Course Exercises Guide

Essentials of Service Development for IBM DataPower Gateway V7.5

Course code WE751 / ZE751 ERC 1.1

IBM Training

August 2016 edition

Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Contents

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

Approach®	CICS®	DataPower®
DB™	developerWorks®	IMS™
Rational®	Redbooks®	Tivoli®
WebSphere®	z/OS®	

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft is a trademark of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Social® is a trademark or registered trademark of TWC Product and Technology, LLC, an IBM Company.

Other product and service names might be trademarks of IBM or other companies.

Exercises description

FLY airline case study

The exercises in this course build upon a common case study: the FLY airline services. The services are composed of a Booking Service web service and a Baggage Service web service. The services are implemented as a BookingServiceBackend MPGW and a BaggageStatusMockService MPGW, both running within the FLYService domain.

The Booking Service has one operation: BookTravel. The SOAP request that is named BookingRequest contains billing details, payment card details, booking type, and the reservation code. The SOAP response is a BookingResponse, which contains the confirmation code and much of the original message. The endpoint is:
`http://<dp_internal_ip>:9080/BookingService/`

The Baggage Service has two operations:

- BaggageStatus. The SOAP request that is named BaggageStatusRequest contains the passenger's last name and their reference number. The SOAP response is BaggageStatusResponse, which contains the status of each bag that is attached to the passenger's reference number.
- BagInfo. The SOAP request that is named BagInfoRequest contains the ID number of the bag in question. The SOAP response is BagInfoResponse, which contains the status of the bag and which passenger it belongs to. The Baggage Service does not have a WSDL, and cannot be proxied by a web service proxy. The endpoint is:
  `http://<dp_internal_ip>:2068/BaggageService/`

Technically, the FLY airline services are self-contained MPGWs that mimic a web services back end that might be on WebSphere Application Server.

This application minimizes its dependencies on data sources by relying on data from a flat file, and allowance of read-only operations.

Exercises

This course includes the following exercises:

- **Exercise 1**: First exposure to the DataPower developer environment

  Import an MPGW and update its handler listening port. Examine some of the configuration. Test the service with a browser and a cURL command. Export the service.

- **Exercise 2**: Creating a BookingService gateway

  Create a simple MPGW that proxies the back-end Booking Services web services. Test with SoapUI.

- **Exercise 3**: Enhance the BookingService gateway

  Enhance the BookingServiceProxy MPGW to validate, filter, and transform the message. Test with SoapUI.

- **Exercise 4**: Adding error handling to a service policy

Add an error policy, error rule, and On Error action to the MPGW. Use the probe and SoapUI to observe the processing behavior.

- **Exercise 5**: Creating cryptographic objects and configuring SSL

  Generate the key materials and create the crypto objects needed to support SSL. Add an HTTPS FSH to the Booking Service MPGW. Use cURL to test the HTTPS FSH. Create a second MPGW that calls the Booking Service MPGW over HTTPS. Create the SSL-related objects. Use SoapUI to test the SSL behavior. Use the system log and the probe to observe the internal behavior.

- **Exercise 6**: Implementing a service level monitor in a multi-protocol gateway

  Create a log target. Create an SLM Resource Class object to identify the type of traffic to monitor. Add an SLM action to the MPGW service policy. Test with a load test from SoapUI. Change the SLM sanction. Test again and see the difference in behavior with the different sanction.

- **Exercise 7**: Using a DataPower pattern to deploy a service

  Import a pattern. Configure the points of variability. Deploy the pattern into a new service. Edit the generated service to examine its configuration. Use SoapUI to test the service.

Note

Most of the exercise steps use the Blueprint Console rather than the WebGUI because the Blueprint Console is replacing the WebGUI in some future release. Because it is a “technical preview”, there are still a few bugs in the interface. The most noticeable bug in V7.5.0.0 is when the scroll bar in a window does not respond, or does not display properly. Until the bug is fixed in a fix pack, you can easily get around the problem. Click somewhere in the window to get focus, and then use the up arrow and down arrow keys to scroll the window contents.

In the exercise instructions, you see that each step has a blank preceding it. You might want to check off each step as you complete it to track your progress.

Most exercises include required sections, which must always be completed. These exercises might be required before doing later exercises. Some exercises also include optional sections that you might want to do if you have sufficient time and want an extra challenge.

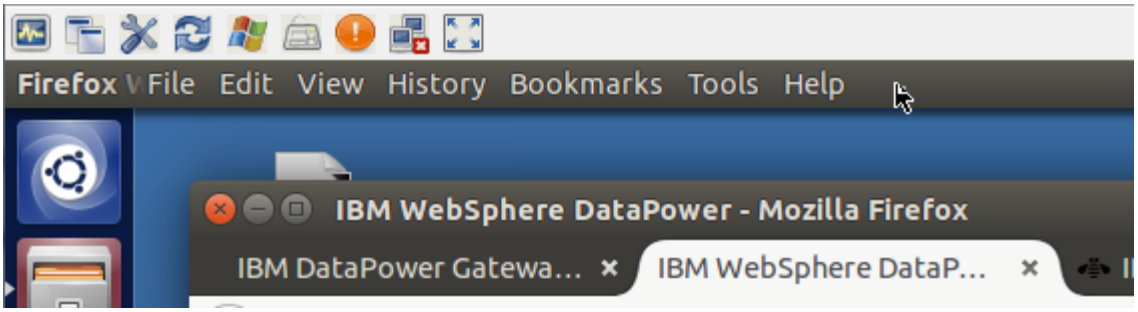
If you are using the IBM remote lab environment:

As of June 2016, the environment that is used to support the IBM-supplied images and DataPower gateways is Skytap. Each student is supplied an Ubuntu student image and a DataPower gateway.

- Ignore all offers to upgrade any of the software on the image. The image and exercise steps are designed to operate at the supplied levels of the contained software.
- The supplied image is Ubuntu 14.04 LTS. The desktop uses Unity, which is different than the more common Gnome desktop. Some hints on using Unity are at: http://www.howtogeek.com/113330/how-to-master-ubuntus-unity-desktop-8-things-you-need-to-know/

- A noticeable difference is that the menus on the windows within the desktop are not typically visible. When a window is the “active” window, that window does not have any menu items, but the application type is displayed in the black bar that spans the top of the desktop. In the following screen capture, observe that the browser window does not have a menu bar, and that its type of “Firefox Web Browser” is listed in the black bar.

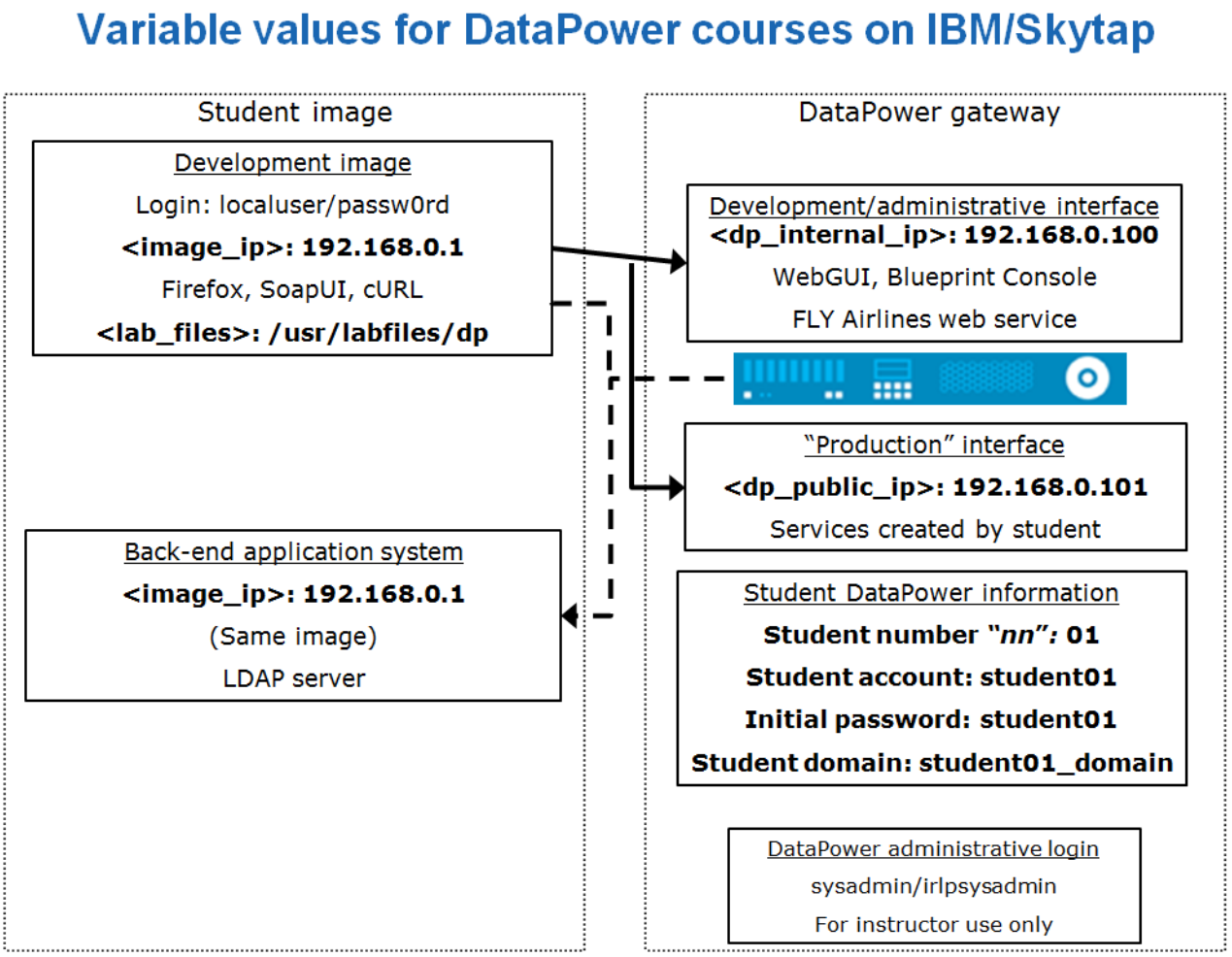
If you hover the mouse over the black bar, the menu items for the active window are displayed.



Another noticeable difference is that when a window is maximized, the **Close**, **Minimize**, and **Restore** buttons are not visible until you hover the mouse in the black bar.

This Unity behavior is discussed in the “Hidden Global Menus” section of the previously mentioned howtogeek.com page.

- The IBM supplied environment has pre-assigned values for some of the variables, such as the IP addresses of the student image and the DataPower gateway, the student number, and the initial password. The following graphic shows those assignments.



Important

Online course material updates might exist for this course. To check for updates, see the Instructor wiki at http://ibm.biz/CloudEduCourses.

Exercise 1. First exposure to the DataPower developer environment

Estimated time

00:45

Overview

In this exercise, you use the WebGUI and Blueprint Console to examine a multi-protocol gateway, edit the gateway, and test the service by using a browser and by using the cURL command.

Objectives

After completing this exercise, you should be able to:

- Log in to the WebGUI
- Use the navigation bar
- Use an object catalog
- Connect to the Blueprint Console
- Import a service
- Edit a multi-protocol gateway
- Review the actions in a policy editor
- Test a service from a browser and a cURL command
- Export a service

Introduction

In this exercise, you are guided through interactions with the WebGUI. You start off by working in the navigation bar. You connect to the other web interface, the Blueprint Console. Because you are going to test a service in a later section, you import and update a Hello World multi-protocol gateway service. You then test the Hello World service, and you look at the responses and the log messages that are generated. Finally, you export the modified service configuration.

Requirements

To complete this exercise, you need:

- Access to the **DataPower** gateway

- **cURL**, to send a request from a terminal window
- Services in the **FLYServices** domain
- Access to the *<lab_files>* directory

Exercise instructions

Preface

- Remember to use the domain and port address that you were assigned in the exercise setup. **Do *not*** use the default domain.
- The references in exercise instructions refer to the following value:
  - <*lab_files*>: Location of the student lab files. Default location is: `/usr/labfiles/dp/`
  - <*dp_internal_ip*>: IP address of the DataPower gateway development and administrative interface.
  - <*nn*>: Assigned student number. If no instructor is present, use “01”.
  - <*studentnn*>: Assigned user name and user account. If no instructor is present, use “`student01`”.
  - <*studentnn_password*>: Account password. In most cases, the initial value is the same as the user name. You are prompted to create a password on first use. Write it down.
  - <*studentnn_domain*>: Application domain that the user account is assigned to. If no instructor is present, use “`student01_domain`”.
  - <*mpgw_helloworld_port*>: `12`*nn*`7`, where “*nn*” is the two-digit student number. This number is the listener port for the HelloWorld MPGW.
  - <*dp_WebGUI_port*>: Default port is 9090, listening port for the WebGUI.

1.1. Initialize the lab environment

Some setup activities are required to properly configure the lab environment and determine IP addresses and ports.

__ 1. If you have not yet performed the setup activities, you must go to **Appendix B: Lab environment setup**. Complete those activities before proceeding.

These activities need to be performed only once for the course.

1.2. Work with the WebGUI home page

In this section, you log in to the WebGUI for your user account. You are guided through accessing some of the options on the WebGUI home page. At the end of this section, you switch to the BluePrint Console.

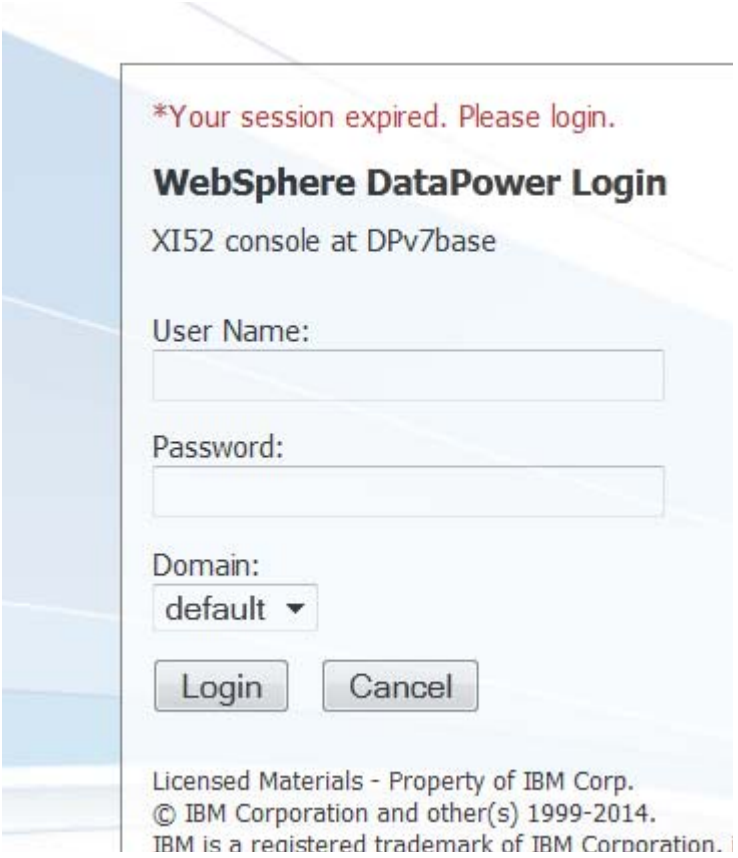
Be sure to look at the Preface section of this exercise for the substitution values that are entered during the exercise.

__ 1. Use a web browser to log in to the DataPower WebGUI.

__ a. Connect to the WebGUI home page:
`https://<dp_internal_ip>:<dp_WebGUI_port>`

Note

If you get an "untrusted connection" message, choose to accept it.



__ b. Enter:

- **User Name**: *<studentnn>*
- **Password**: *<studentnn_password>*
- **Domain**: *<studentnn_domain>*

__ c. If this attempt is the first time that this account is accessed, you are prompted to change the password. Write down the new password somewhere. If you had to create a password, you are logged out of the WebGUI. Log back in, with the new password.

__ 2. The WebGUI home page - the Control Panel view - is displayed. Examine the right side of the banner area of the page. Note the Domain value. Verify that you are *not* in the default domain.

__ 3. Expand the main menu choices in the navigation bar.

__ 4. Examine the Control Panel area of the WebGUI home page. Most of the links here are also available in the navigation bar.

__ 5. Generally, there are several ways to get to many of the functions in the WebGUI. This fact is demonstrated by accessing the Multi-Protocol Gateway catalog in different ways over the next several steps.

In the **Services** area of the Control Panel, click the **Multi-Protocol Gateway** link.

Multi-Protocol Gateway

__ 6. The page changes to the multi-protocol gateway (MPGW) catalog, which lists all the defined MPGWs in this domain. It also has a button to initiate the creation of an MPGW.

You can click the name of any of the MPGWs to view or edit their configurations. Because you are in a fresh domain, no gateways are listed in the catalog yet.

MPGWs are covered in detail in a later unit.

__ 7. Click **Control Panel** in the navigation bar. This link should be visible at the upper-left side of the WebGUI page.

__ 8. You are back on the WebGUI home page, or the Control Panel. This time, select **Services > Multi-Protocol Gateway > Edit Multi-Protocol Gateway**.

__ 9. The multi-protocol gateway catalog page appears as before. It should still be empty. This step demonstrates that there are typically multiple ways to get to many of the configuration pages within the WebGUI.

__ 10. Click **Control Panel**.

__ 11. You are now back on the main web page for the WebGUI. Click **Blueprint Console**.

Information

You can connect directly to the Blueprint Console and bypass the WebGUI by entering:
`https://<dp_internal_ip>:<dp_WebGUI_port>/dp/index.html`

__ 12. This action opens the **Blueprint Console** on a new tab in your browser.

Attention

For the DataPower V7.5.0 firmware, the Blueprint Console is considered a “Technical Preview”. It will replace the WebGUI in a future release. This course will use the Blueprint Console wherever possible to prepare you for future releases. The steps use WebGUI when the Blueprint Console support is not currently available. Regardless of which web interface is used, the contents of the configurations are identical.

__ 13. The Blueprint Console “home” page is the **All Services** page. The Control Panel page does not exist in the Blueprint Console.

Note

From this point forward, you can work in either web interface, although you should work primarily in one or the other.

1.3. Work in the Blueprint Console

In this section, you are introduced to the Blueprint Console. At the end of the section, you import an existing service configuration.

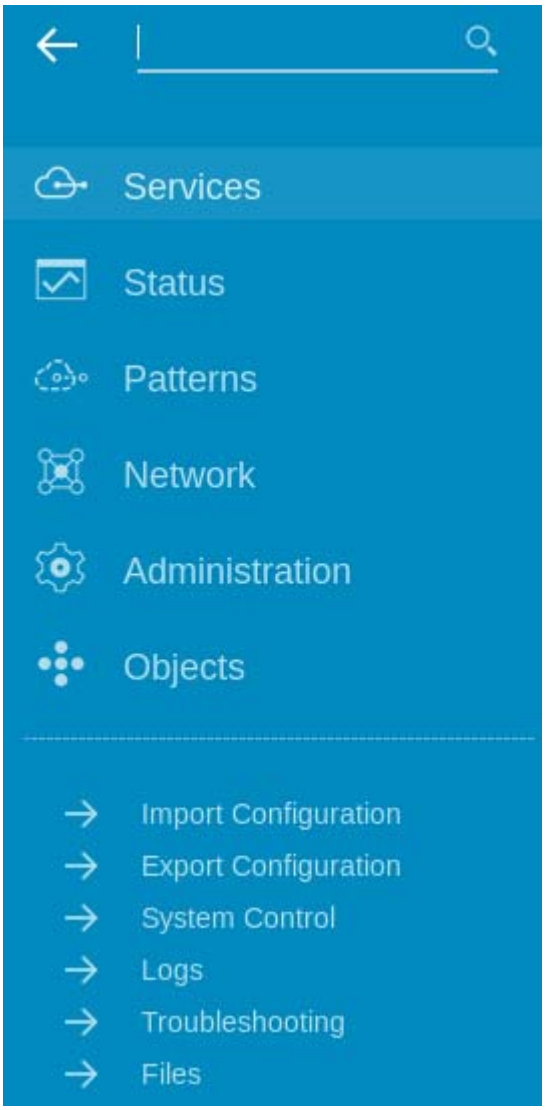
Important

The screen captures are usually for an example user account. Your account name might be different. Be sure to follow the lab instructions for naming your objects and specifying values.

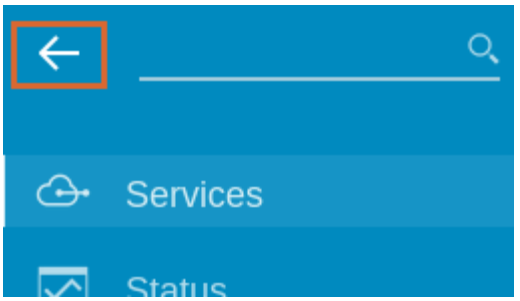
__ 1. Examine the banner of the All Services page. The upper left has the **Open** icon (three horizontal lines) to open the navigation area.

__ 2. Click the **Open** icon.

__ 3. The navigation area expands.

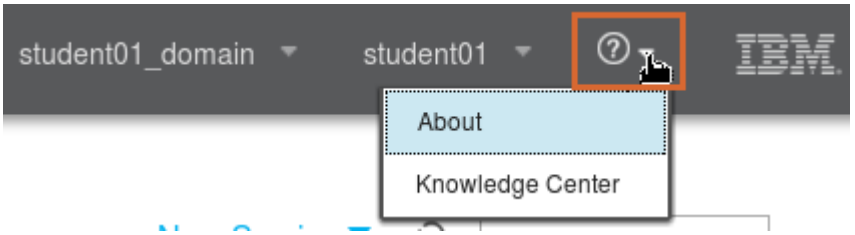


__ 4. You work with this area throughout the lab exercises. For now, use the **Close** icon (left arrow) to close the area.

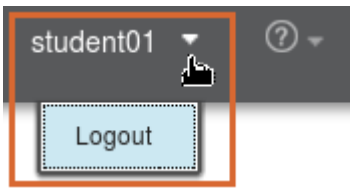


__ 5. The right side of the banner has a question mark. Click the down arrow to see the choices:

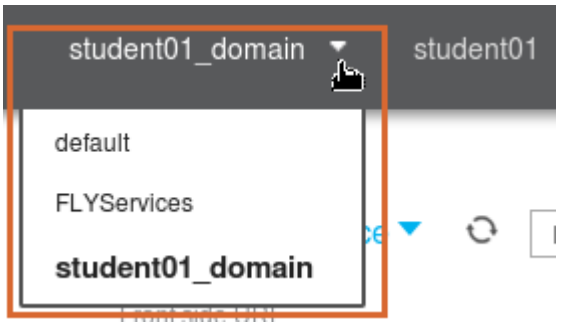
- The **About** choice displays the firmware version.
- The **Knowledge Center** choice links to the DataPower Knowledge Center web page.



__ 6. The next item to the left of the question mark displays the user account. Click the down arrow to see the **Logout** option. Use this option when you want to log out of the Blueprint Console.

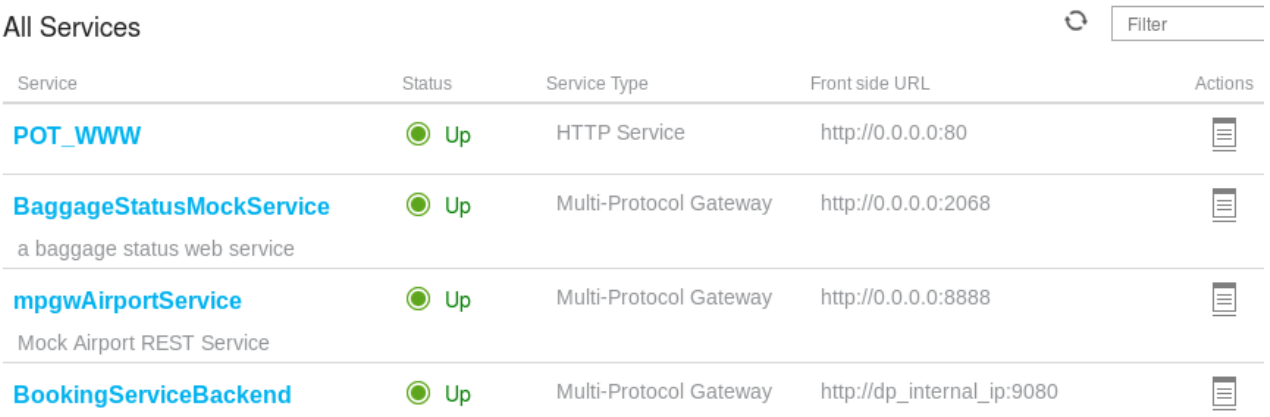


__ 7. The item to the left of the user account lists the domain that you are currently in: **student01_domain** in the screen capture example. Click the down arrow to see the other domains that you have access to.



__ 8. Click **FLYServices**.

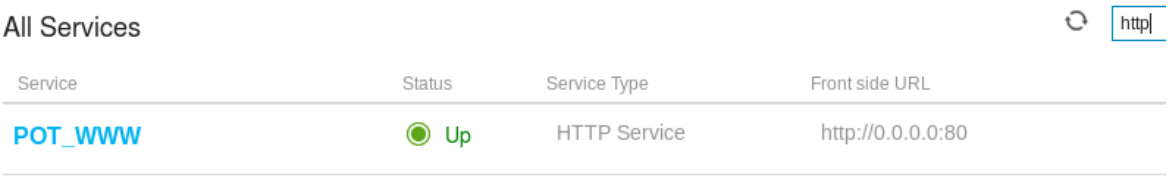
__ 9. The banner now indicates that you are in the FLYServices domain. The All Services page lists an HTTP service and several MPGWs.



The services are grouped by Service Type, alphabetically. If you hover over a column header (Service, Status, Service Type, Front side URL), you see an up/down arrow at the right edge of the column. You can select that to reorder the sort in that column.

In the upper right is a **Filter** field. Enter `http` in that field.

__ 10. The list is filtered for any column that contains ”http”.



The POT_WWW service contains “http” in its Service Type column.

__ 11. An “X” is displayed at the end of the **Search** field. Click **X** to remove the filter. The list should refresh to contain all four services.

__ 12. Right now, you are displaying “all services” regardless of service type. In the navigation area, click **HTTP Service**.

__ 13. You should now see just the single HTTP service. Click **Multi-Protocol Gateway** in the navigation area.

__ 14. The list displays the three MPGWs only. Click **All Services** in the navigation area to see all services regardless of type.

__ 15. The **Actions** column has a single icon, View Log. Since this is a read-only domain, options to delete or create a service do not show.

__ 16. Click **BaggageStatusMockService**.

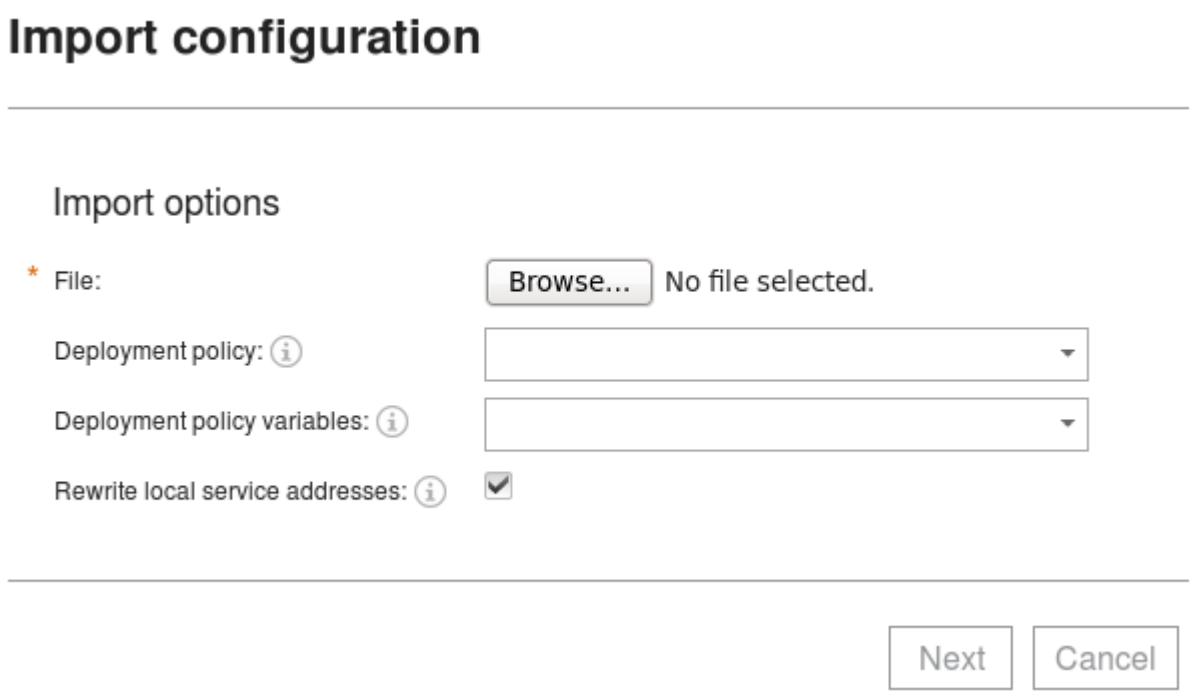
__ 17. The configuration page for the MPGW is displayed. Since you have read-only access, any attempt to modify the configuration displays a “permission denied” error message. The details of configuring an MPGW are covered in later lectures.

__ 18. Switch back to your student application domain - ***<studentnn_domain>*** - by selecting it from the domain indicator in the banner. This domain has full read/write access.

__ 19. The next activity is to import an MPGW configuration into your application domain. Click the **Open** icon to expand the navigation area.

__ 20. Click **Import Configuration**.

__ 21. On the Import Configuration page that appears, click **Browse**.



__ 22. Select the import file: `<lab_files>/intro/HelloWorldMPGW.zip`

__ 23. Leave the other fields at their defaults, and click **Next**.

__ 24. On the next page, you can see the list of objects and embedded files that are in the import file. The new objects and files are automatically checked for import. Notice the XSL style sheet file (`helloxslworld.xsl`) and the JavaScript file (`hellojsworld.js`). Click **Import**.

__ 25. The next page should indicate a successful import. Click **Close**.

__ 26. You now have a HelloWorld MPGW in your **All Services** list. You will investigate it in the next section.

__ 27. Just below the banner, a notification is displayed.

__ 28. Click **Save changes** to commit your new configuration to the domain's configuration file.

Information

When the gateway is started, each domain's configuration is read from a `cfg` file on the gateway file system. When a configuration is completed (by clicking **Apply** or **Done**), that object is now active and usable in the domain. However, it exists only in memory, and is not yet persisted. By clicking **Save changes**, you write all memory-only configurations to the `cfg` file in the file system.

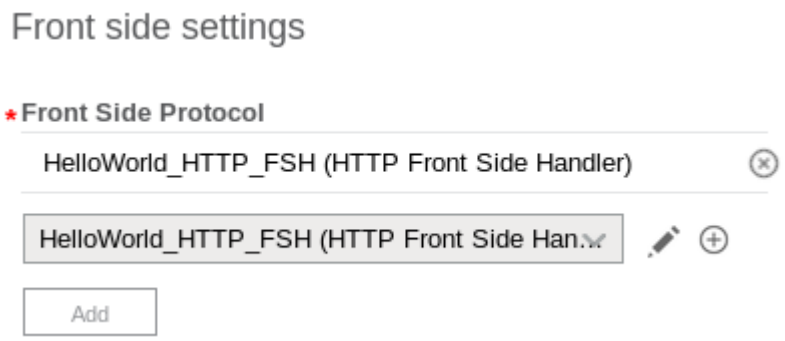
1.4. Examine and edit a service

In this section, you examine the HelloWorld MPGW service that you imported in a previous step. You update the port that this service listens on.

__ 1. From the **All Services** list, select the **HelloWorld** MPGW.

__ 2. The HelloWorld MPGW configuration page is displayed. First, you examine and update the front side protocol handler. For an MPGW, the front side protocol handler defines how the service receives client requests. On the middle right side, find the **Front Side Protocol** section.

__ 3. Click **HelloWorld_HTTP_FSH**. This action causes the handler name to appear in the following selection box.



__ 4. Click the **edit** button (pencil icon).

Information

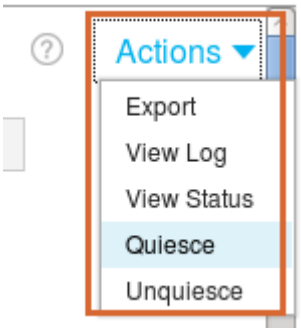
The plus sign icon is used to *create* an object.

The pencil icon is used to *edit* the existing object that is displayed in the selection box.

__ 5. The Front Side Protocol page opens. On the left side is the navigation tree. Only one object is listed, the handler. Notice that it indicates the type of handler: HTTP Front Side Handler.

__ 6. On the upper right is a question mark. Click this icon to display online help for this object.

__ 7. The **Actions** menu lists more operations that you can perform on the object.



__ 8. Beneath the name of the object is the **Main** section. Notice that it has “twistie” icon (down arrow). The twistie can be clicked to expand and contract the section. The HTTP front side protocol handler has only the Main section.

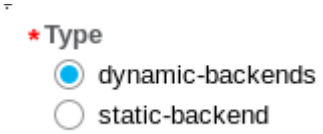
__ 9. A “circled i” icon shows to the right of the field names. You can hover over that icon to get some help for that field.

__ 10. This object defines the port that the service listens on, and what other HTTP options are supported. Change the **Port** to the *<mpgw_helloworld_port>* value.

__ 11. Click **Apply** to save the new handler configuration.

__ 12. The handler window closes. Click **Apply** on the Multi-Protocol Gateway page if it is enabled. This action saves the MPGW with the new definition of the handler.

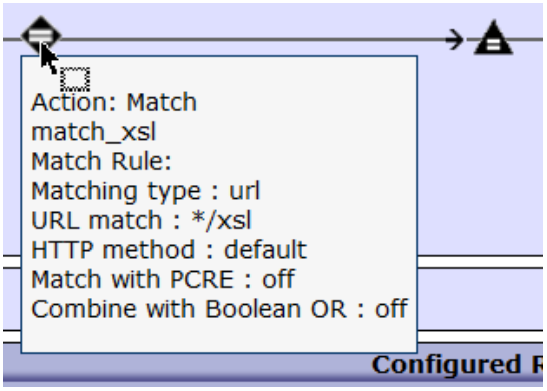
__ 13. On the left side of the page, notice that the **Type** setting is **dynamic-backends**. This setting indicates that the service policy is going to determine the back-end application URL. For this case of the HelloWorld MPGW, no back-end application is called. A response is sent back to the client from the MPGW itself. This situation is called a “loopback” type of service.



__ 14. The last examination of HelloWorld is of the service policy. Look for **HelloWorld** as the **Multi-Protocol Gateway Policy**. Click the **edit** button.

__ 15. A policy editor window opens. In the **Configured Rules** section, click the **xsl_rule**. This action makes the row of the selected rule change to a bold font. More importantly, it makes the selected rule become the rule that is displayed in the rule area.

__ 16. Hover the mouse over the action icon on the left, the Match action.



__ 17. In the hover box, observe the URL match specification: `*/xsl`. This specification indicates that this rule gets control if the received URL ends in "`/xsl`".

__ 18. The next action, which is an equal sign in a triangle, is the action that specifies that this service is a loopback service.

__ 19. The last action, the circle with the curvilinear patterns, indicates that a style sheet is executed. Double-click that action.

__ 20. A Transform action window opens. By the Transform File field, click **View**.

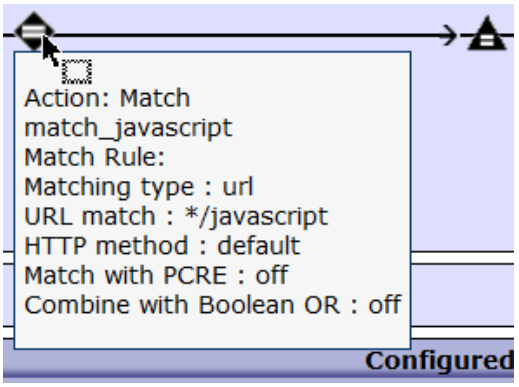
__ 21. A browser window opens that lists the contents of the `helloxslworld.xsl` style sheet. The style sheet writes a message to the DataPower system log, and writes HTML to produce the "hello world" browser page.

__ 22. Close the browser page.

__ 23. Click **Cancel** on the Transform action page.

__ 24. Back on the policy editor in the **Configured Rules** section, click the **javascript_rule**.

__ 25. In the rule area, hover over the Match action.



__ 26. In the hover box, observe the URL match specification: `*/javascript`. This specification indicates that this rule gets control if the received URL ends in "`/javascript`".

__ 27. The next action, which is an equal sign in a triangle, is the action that specifies that this service is a loopback service.

__ 28. The last action, a square with braces (**{}**), indicates that JavaScript is executed. Double-click that action.

__ 29. A Configure GatewayScript action window opens. By the GatewayScript file field, click **View**.

__ 30. A browser window opens that lists the contents of the `hellojsworld.js` file. The code writes a message to the DataPower system log, and writes HTML to produce the “hello world” browser page.

__ 31. Close the browser page.

__ 32. Click **Cancel** on the GatewayScript action page.

__ 33. Close the policy editor window.

__ 34. In the notification area beneath the banner, click **Save changes** to write the new configuration to the configuration startup file.

Information

If you get a window that states something like “Other domains contain unsaved changes” when you attempt to save your configuration, select your domain and proceed.

1.5. Test a service

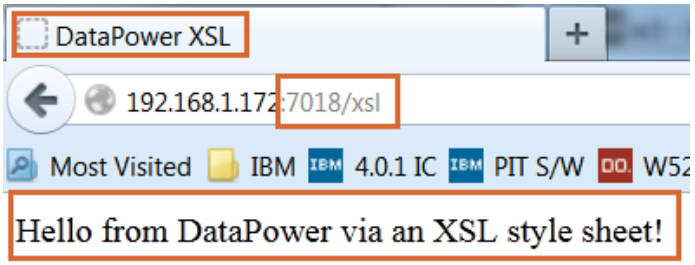
In this section, test the service, both from a browser and from the cURL command.

__ 1. Open a browser.

__ 2. Enter the URL to request the XSL version of HelloWorld (be careful that you use *http*):

`http://<dp_public_ip>:<mpgw_helloworld_port>/xsl`

__ 3. The service responds.



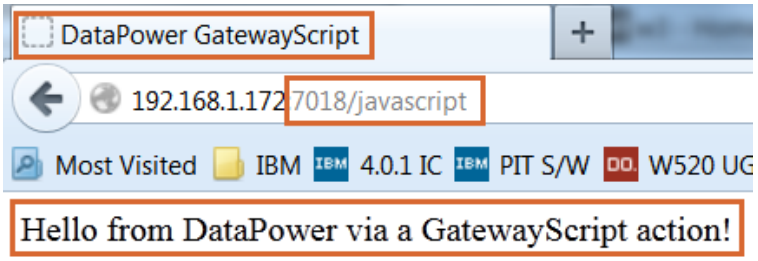
__ 4. The page title comes from the `<title>` HTML element. The page contents come from the HTML `<body>`. Notice the sample URL, especially the URI section.

__ 5. If you want, you can go back into the policy editor of the service and review the XSL style sheet.

__ 6. Enter the URL to request the JavaScript version of HelloWorld:

`http://<dp_public_ip>:<mpgw_helloworld_port>/javascript`

__ 7. The service responds.

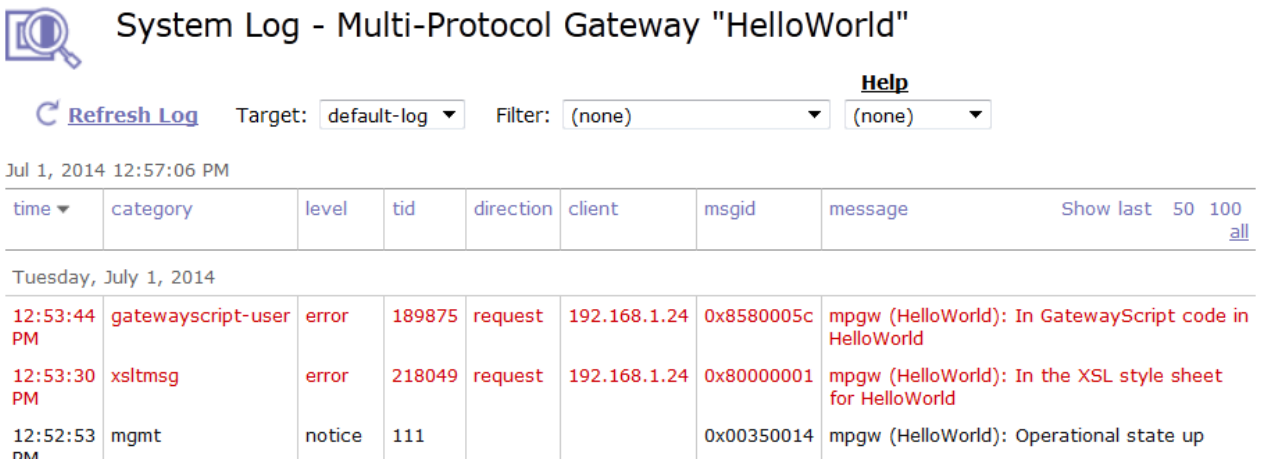


__ 8. As before, the page title comes from the `<title>` HTML element. The page contents come from the HTML `<body>`. Notice the sample URL, especially the URI section.

__ 9. If you want, you can go back into the policy editor of the service and review the GatewayScript code.

__ 10. On the Multi-Protocol Gateway page, click **Actions > View Log**.

__ 11. A window opens that displays the system log for entries that relate to this MPGW. The screen capture shows the right side of the log. Look in the **messages** column. Notice the entries that were generated in the XSL and in the javaScript code.



__ 12. Leave the log window open.

__ 13. Open a terminal window.

__ 14. Enter a cURL command to send an HTTP request for the XSL web page:

```
curl -G http://<dp_public_ip>:<mpgw_helloworld_port>/xsl
```

Information

cURL is a command line tool to send and receive HTTP requests. In later exercises, you use another tool that is called SoapUI to send more complex requests.

__ 15. You get a text response that contains the HTML to construct the Hello World XSL page.

__ 16. Use the same terminal window to enter a cURL command to send an HTTP request for the JavaScript web page:

```
curl -G http://<dp_public_ip>:<mpgw_helloworld_port>/javascript
```

__ 17. You get a text response that contains the HTML to construct the Hello World JavaScript page.

__ 18. You should notice a difference in the text between the HTML text that is generated from the raw JavaScript, and the HTML text that is generated from the XSL style sheet.

__ 19. If you want, you can switch to the log window, and click **Refresh Log**. You should see the two new, but similar messages.

__ 20. Close the log window.

1.6. Export a configuration

You export your updated configuration, and examine its contents.

__ 1. Click **Open > Export Configuration**.

__ 2. An Export Configuration window opens. Select **Export data for select configurations from the current domain** and click **Next**.

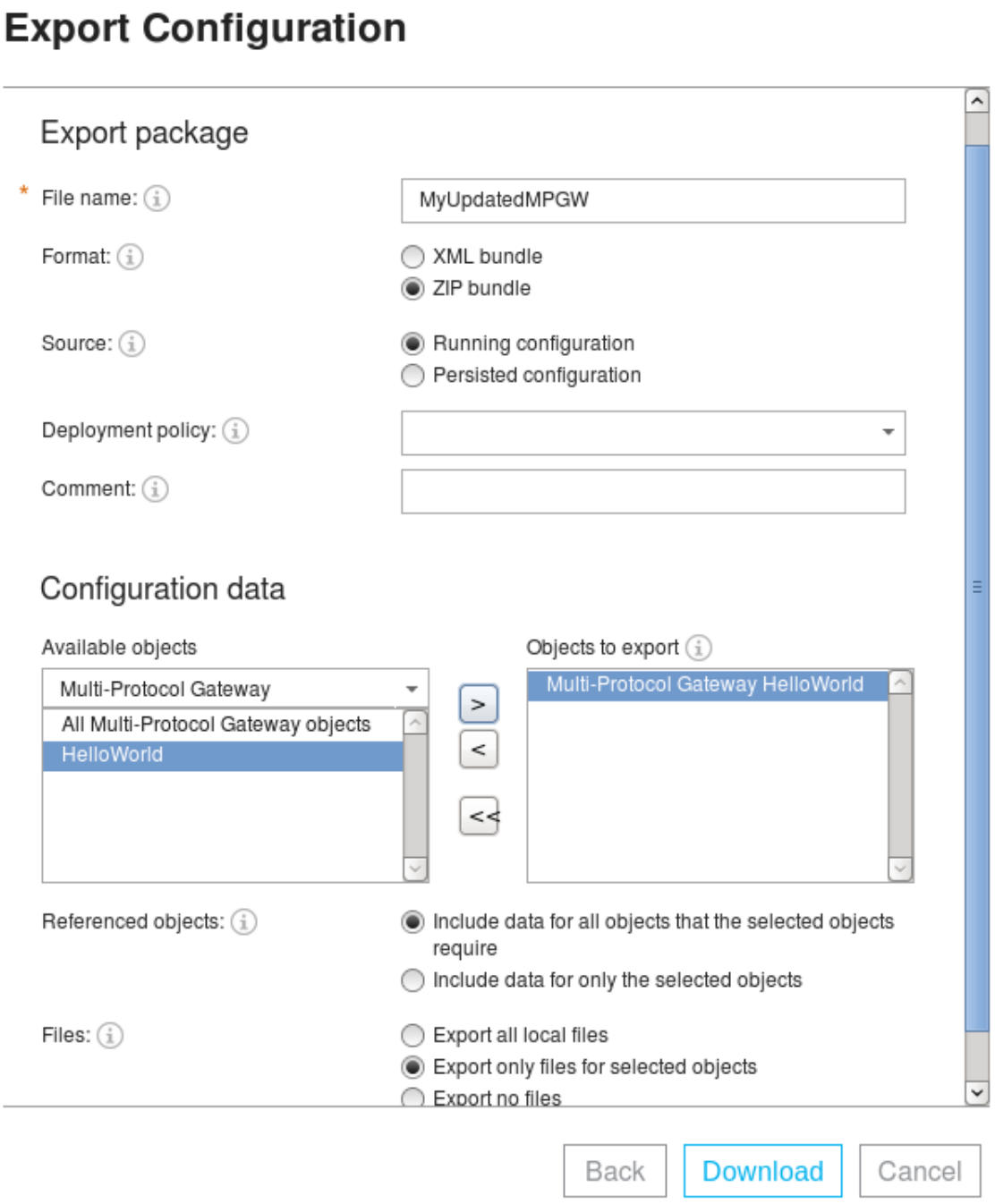
__ 3. Enter a file name of `MyUpdatedMPGW`.

__ 4. Leave the format as **Zip bundle** and source as **Running configuration**.

__ 5. For Available objects, select **Multi-Protocol Gateway**. In the lower list, select **HelloWorld**.

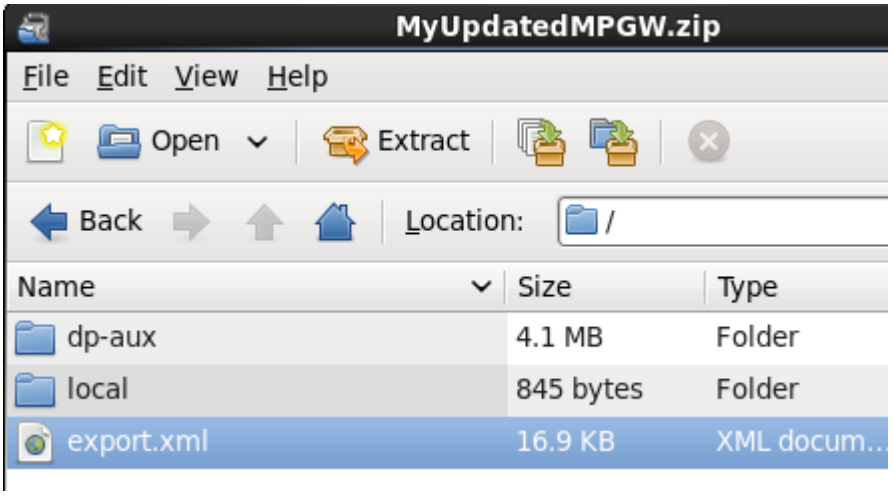
__ 6. Click the right arrow button to move **HelloWorld** into the **Objects to export** list.

__ 7. Leave the remaining selections as-is, and click **Download**.



__ 8. The objects are exported to a zip file. Click **Save File**.

__ 9. The zip file downloads to the browser. Open the zip file (which might happen automatically). You can use File Roller to expand the file.



The `dp-aux` folder contains DataPower environment settings.

The local folder contains the files from the `local:///` directory: `helloxslworld.xsl` and `hellowjsworld.js`.

The `export.xml` file contains the XML-formatted configuration for the MPGW and its referenced object. If you examine the contents, you can find a <LocalPort> element that contains your updated port number.

The zip file can be sent to a source repository, can be imported into another domain, or can be imported on another gateway.

__ 10. Close the editor and file browser if open.

__ 11. Click **All Services** in the navigation area.

__ 12. The HelloWorld MPGW now shows in the **All Services** list. Notice that the Actions column has a delete (trash can) icon.

If you want, you can log out of the Blueprint Console and the WebGUI.

End of exercise

Exercise review and wrap-up

In this exercise, you maneuvered around the WebGUI, worked in the Blueprint Console, imported a service, edited the service, and tested it from a browser and a cURL command, and exported the service.

Exercise 2. Creating a BookingService gateway

Estimated time

00:45

Overview

This exercise shows you how to create a basic multi-protocol gateway (MPGW). You learn the basic steps necessary to implement a message flow within the DataPower gateway. You use the SoapUI tool to send a SOAP message to the MPGW that you created. After DataPower processes the message, a response SOAP message is sent back to SoapUI and displayed in the response window.

Objectives

After completing this exercise, you should be able to:

- Create a multi-protocol gateway
- Test the message flow by using the SoapUI graphical test tool

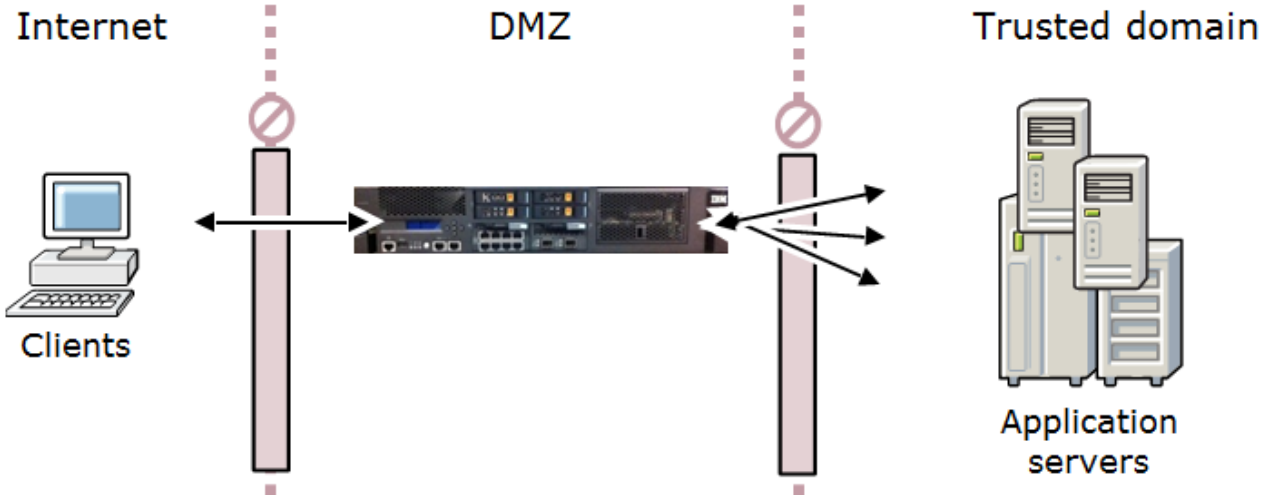
Introduction

The DataPower gateway provides the capabilities to set up multiple MPGWs, or virtual firewalls, to protect back-end applications. Each of these firewalls operates on a port with a policy that consists of a set of rules. Each rule contains actions that complete one or more functions.

An MPGW has functions similar to a proxy server. The client accesses the MPGW as if it is the actual web service. The client does not know the real location of the web service system. The gateway can examine the incoming request from a client. The message content can be transformed or enhanced. The message itself can be rejected for failed authentication or malformed content. Finally, the message can then be forwarded to the actual web service that is running in the trusted domain. The MPGW service can also be configured to accomplish content-based routing, where an appropriate back-end server is selected based on the contents of the message or the identity of the client.

In this exercise, the MPGW performs some basic functions: it proxies the client requests to the actual back-end service, and it verifies that the message is a valid SOAP message.

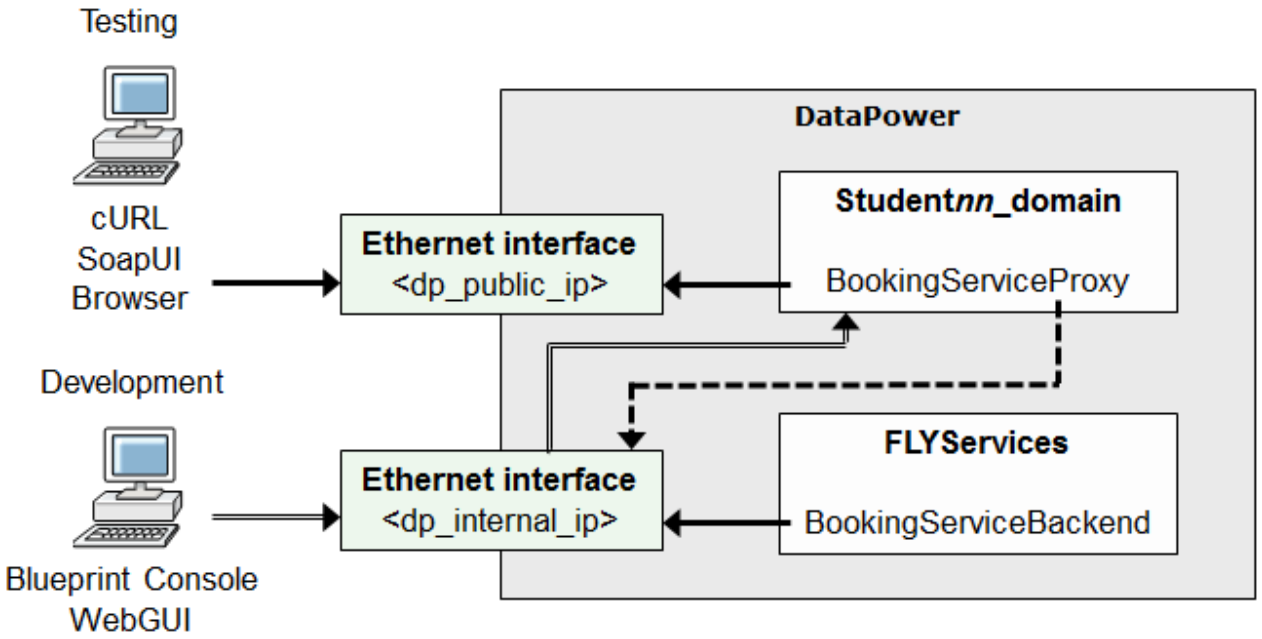
A typical deployment topology is displayed here.



For the exercises in the lab, the topology is simplified:

- The typical back-end application service that is running on a server in the trusted domain is implemented as a service on the gateway itself. A supplied MPGW named BookingServiceBackend listens on port 9080 of the <dp_internal_ip> Ethernet interface in the FLYServices domain. This service listens on the <dp_internal_ip> interface because it is supposed to be hidden from the clients.
- The student develops DataPower services by using the WebGUI and Blueprint Console. The WebGUI and Blueprint Console are enabled on port 9090 of the <dp_internal_ip> Ethernet interface. The student works in the student-specific domain.
- When the student creates the BookingServiceProxy MPGW in this exercise, the MPGW is visible to clients on the <dp_public_ip> Ethernet interface.

- The BookingServiceBackend web service of BookingServiceProxy listens on the <dp_internal_ip> interface. Therefore, the back-end URL of BookingServiceProxy points to that IP address.



Requirements

To complete this exercise, you need:

- Access to the DataPower gateway
- SoapUI, to send requests to the DataPower gateway
- The BookingServiceBackend web service that runs on the DataPower gateway in the `FLYservices` domain
- Access to the `<lab_files>` directoryExercise instructions

You configure an MPGW to proxy messages to a preconfigured back-end web service.

Preface

- Before starting this lab, complete the steps in Exercise 1: First exposure to the DataPower developer environment.
- Remember to use the domain and port address that you were assigned in the exercise setup. **Do *not*** use the default domain.
- The references in exercise instructions refer to the following values:
  - `<lab_files>`: Location of the student lab files. Default location is: `/usr/labfiles/dp/`
  - `<image_ip>`: IP address of the student image (use `/sbin/ifconfig` from a terminal window to obtain the value).

- <*dp_internal_ip*>: IP address of the DataPower gateway development and administrative functions that are used by internal resources such as developers.
- <*dp_WebGUI_port*>: Default port is 9090, listening port for the WebGUI.
- <*dp_public_ip*>: IP address of the public services on the gateway that is used by customer and clients.
- <*nn*>: Assigned student number. If no instructor is present, use “01”.
- <*studentnn*>: Assigned user name and user account. If no instructor is present, use “`student01`”.
- <*studentnn_password*>: Account password. In most cases, the initial value is the same as the user name. You are prompted to create a password on first use. Write it down.
- <*studentnn_domain*>: Application domain that the user account is assigned to. If no instructor is present, use “`student01_domain`”.
- <*was_server_port*>: Port number that the back-end web services listen on. The default port is 9080.
- <*mpgw_booking_port*>: `12`*nn*`1`, where “*nn*” is the two-digit student number. This number is the listener port for the MPGW that proxies the BookingService web service.

2.1. Initialize the lab environment

Some setup activities are required to properly configure the lab environment and determine IP addresses and ports.

__ 1. If you have not yet performed the setup activities, you must go to **Appendix B: Lab environment setup**. Complete those activities before proceeding.

These activities need to be performed only once for the course.

2.2. Create a basic MPGW to validate SOAP messages

In this exercise, you create an MPGW that proxies a SOAP message to a back-end web service. The MPGW is tested with the SoapUI tool by sending already created SOAP messages to the back-end web service.

This lab implements the scenario that is described in three essential steps:

- Section 1, "Create a multi-protocol gateway for the Booking service"
- Section 2, "Configure an HTTP front side protocol handler"
- Section 3, "BookingServiceProxy MPGW testing"

Section 1: Create a multi-protocol gateway for the Booking service

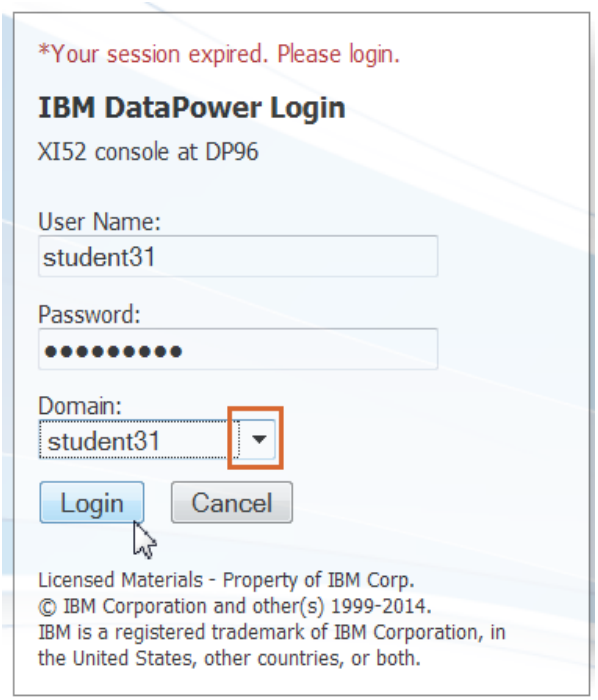
Configure a new multi-protocol gateway on the IBM DataPower Gateway gateway to forward requests to the `BookingService` web service.

__ 1. Use the Firefox icon that is on the desktop to start the Firefox browser.

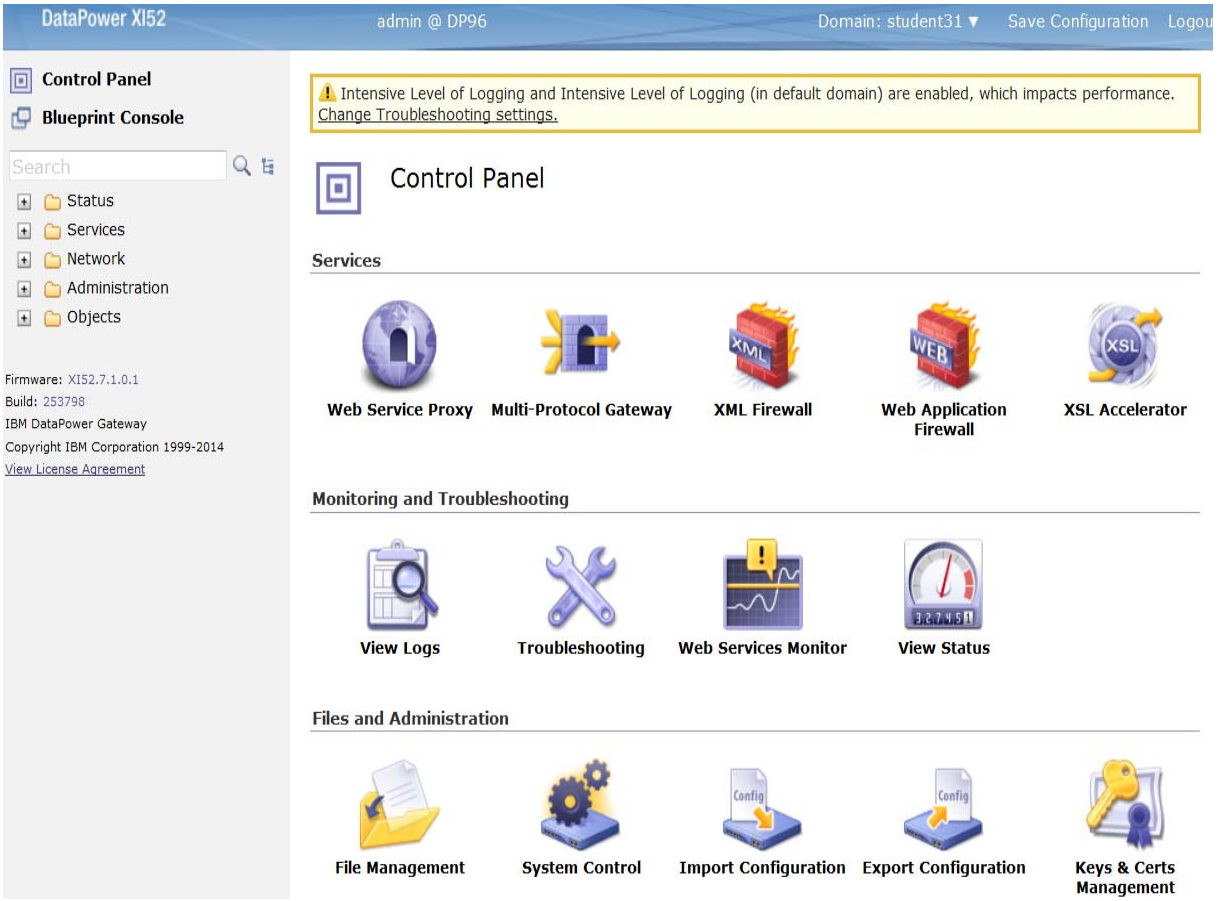
__ 2. Click the DataPower link that is in the toolbar. Ensure that the URL is `https://<dp_internal_ip>:<dp_WebGUI_port>`

__ 3. Enter the **User Name** of <studentnn> and the **Password** of <studentnn_password>.

__ 4. Select the <*studentnn_domain*> domain from the **Domain** list, then click **Login**.



__ 5. After a successful login to DataPower, the main console appears.

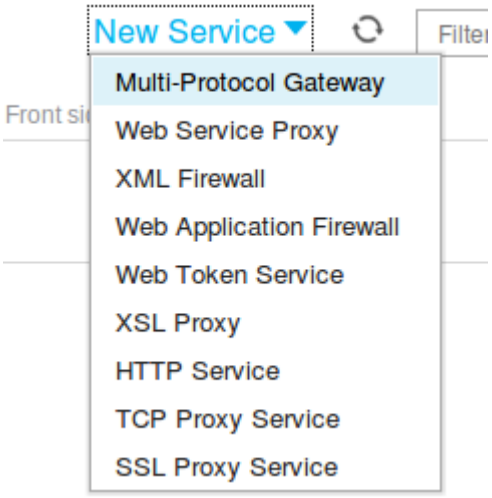


Information

If your gateway has the B2B module that is installed, the Control Panel shows an extra row of B2B icons. This potential difference has no effect on the exercises.

__ 6. In this exercise, you use the Blueprint Console to configure the gateway. Click the **Blueprint Console** in the navigation bar.

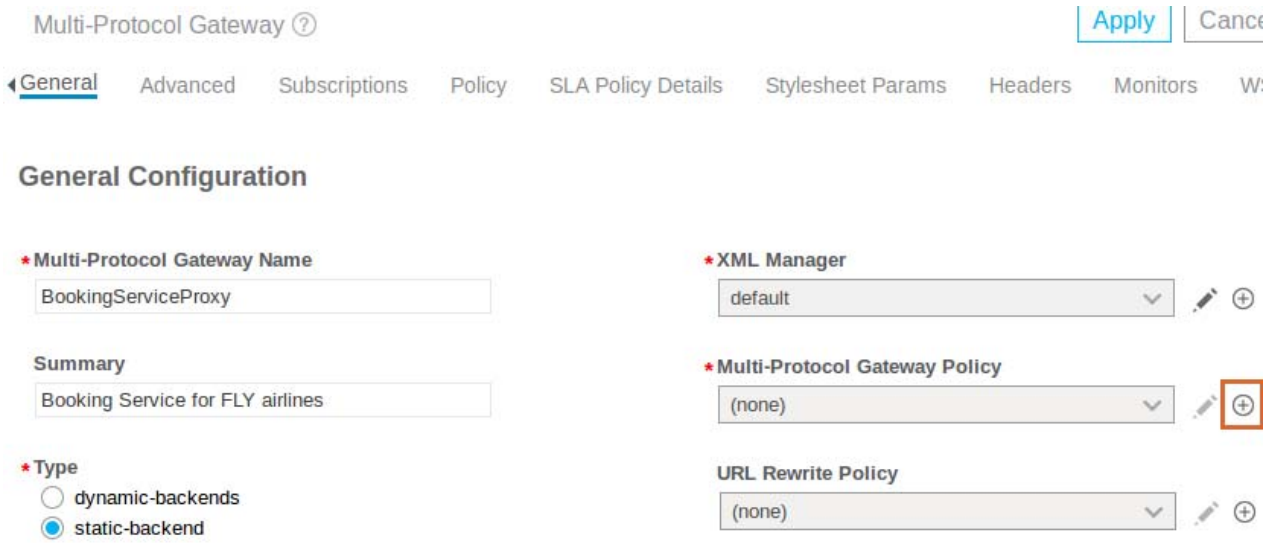
__ 7. The Blueprint Console tab opens, with **All Services** displayed. On the right, click **New Service > Multi-Protocol Gateway**.



__ 8. Enter `BookingServiceProxy` as the name of the new gateway. You can optionally enter a description for the gateway in the summary field, such as `Booking Service for FLY airlines`.

__ 9. In the **XML Manager** field, leave the **default** XML manager selected.

__ 10. In the **Multi-Protocol Gateway Policy** field, click the circled plus sign icon (“new”) to create a multi-protocol gateway policy.



__ 11. Enter `BookingServicePolicy` as the processing policy name.

__ 12. Click **Apply Policy**.

__ 13. Configure a message processing rule to forward any client request message to the back-end service.

__ a. In the processing rule editor for `BookingServicePolicy`, click **New Rule**.

__ b. Set the processing rule direction to **Client to Server**.

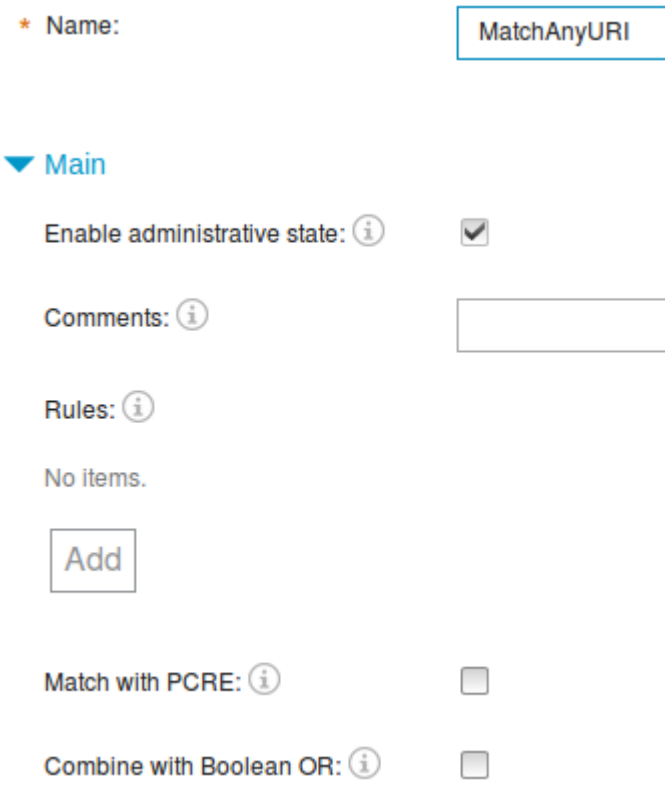
__ c. Double-click the **Match** action icon to configure it. The Match icon is automatically placed on a rule when a rule is created. Any actions on the rule path that are not configured yet are highlighted in yellow.

__ d. Click the circled plus sign icon (“new”) to add a **matching rule**.

__ e. Create a matching rule with the following settings:

- **Name**: `MatchAnyURI`
- **Match with PCRE** (Perl-compatible regular expression): **not selected**

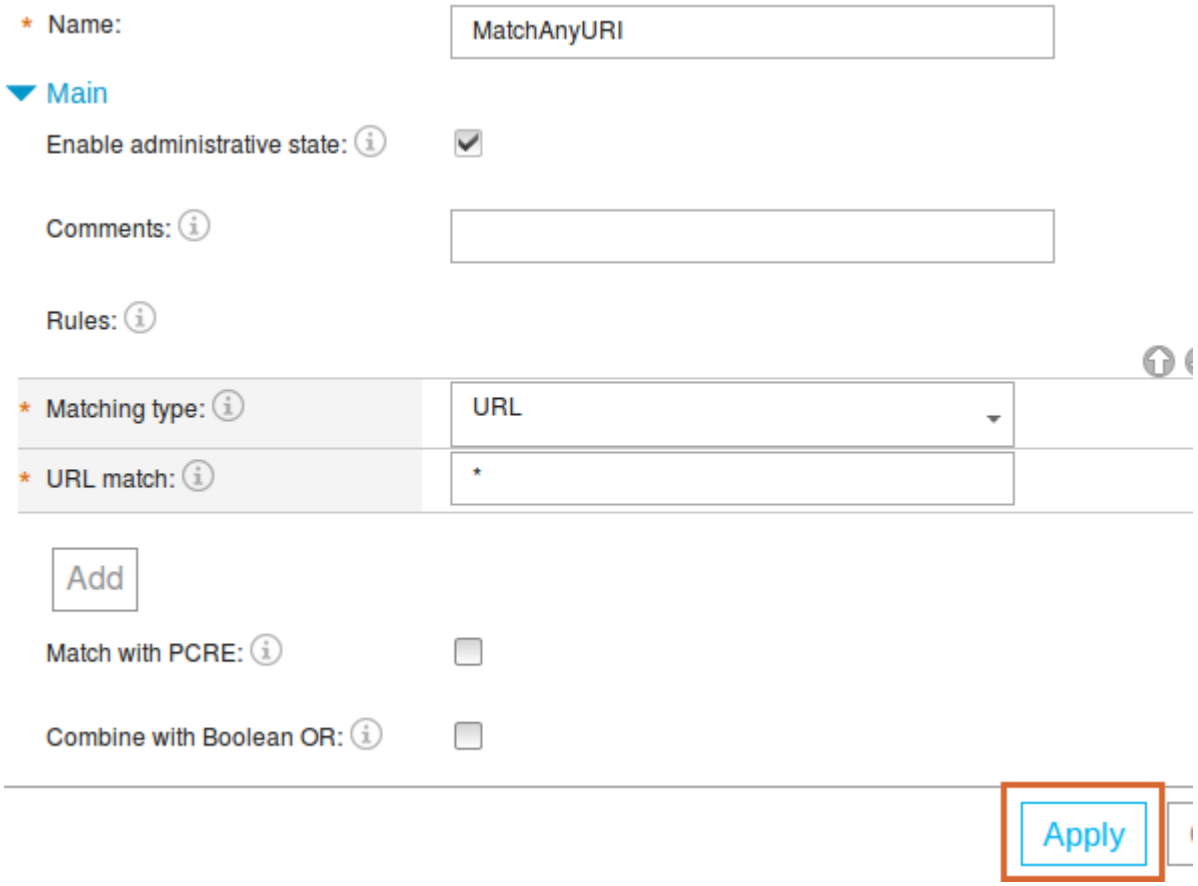
- **Combine with Boolean OR**: **not selected**



__ f. Under the Rules section click **Add** to add a rule.

__ g. Set the **Matching Type** to **URL** and the **URL match** property to: *****

__ h. Click **Apply** to save the matching rule configuration.

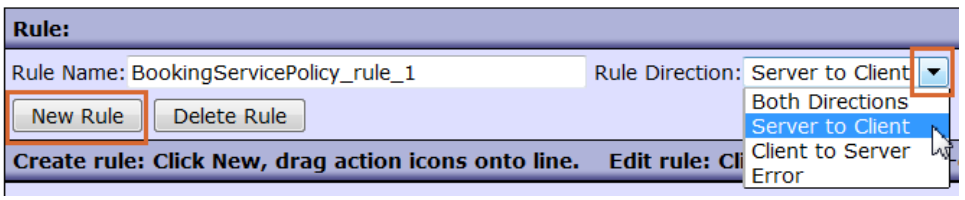


__ i. Click **Done** to use the new `MatchAnyURI` matching rule. The yellow highlighting disappears from the Match action icon.

__ 14. Create a second document processing rule. This response rule processes the message from the back-end server on its way back to the client.

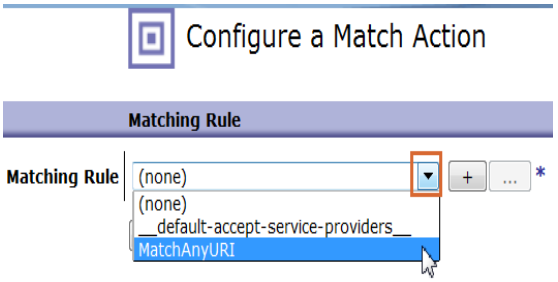
__ a. In the processing rule editor for BookingServicePolicy, click **New Rule**.

__ b. Set the processing rule direction to **Server to Client**.



__ c. Double-click the **Match** action icon to configure it.

__ d. From the **Matching Rule** list, select the previously created **MatchAnyURI**.

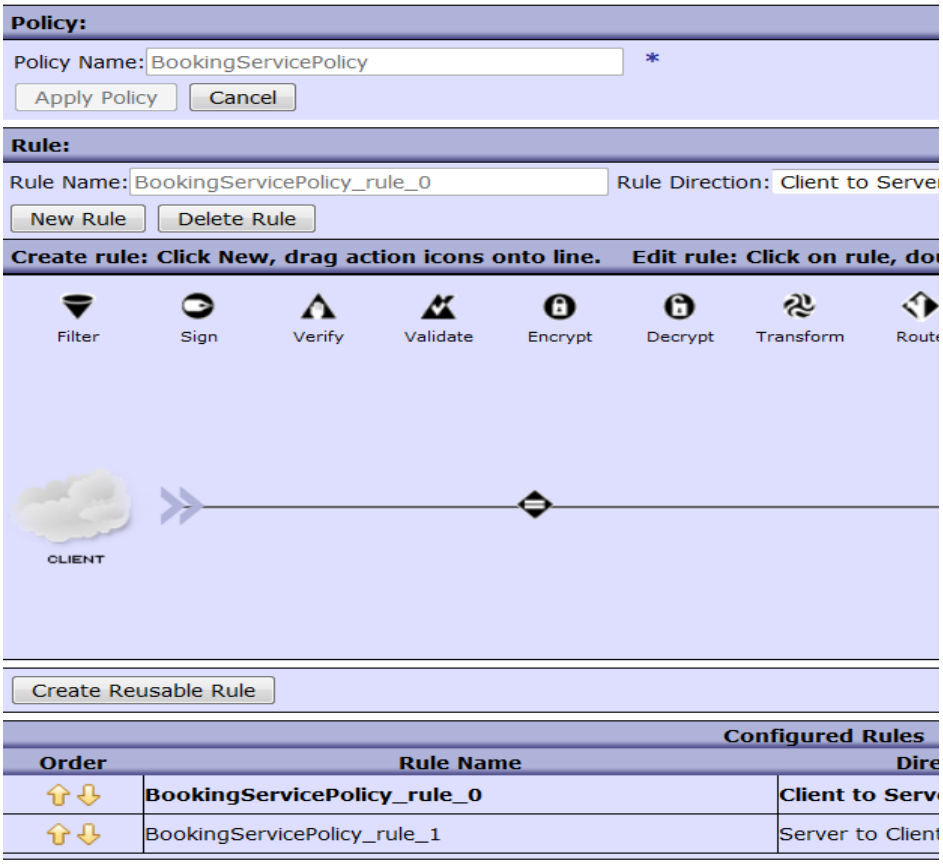


__ e. Click **Done** to use the `MatchAnyURI` matching rule.

__ f. Click **Apply Policy** to save the changes.

__ g. Notice that a **Results** action is automatically added to each rule. Because no other actions exist in the rules, the Results action copies the message content that is passed in to the rule to the output context of the rule.

__ h. Confirm that the new document processing rules are in the list of configured rules.



__ 15. Click **Close Window** to close the `BookingServicePolicy` document policy editor.

Note

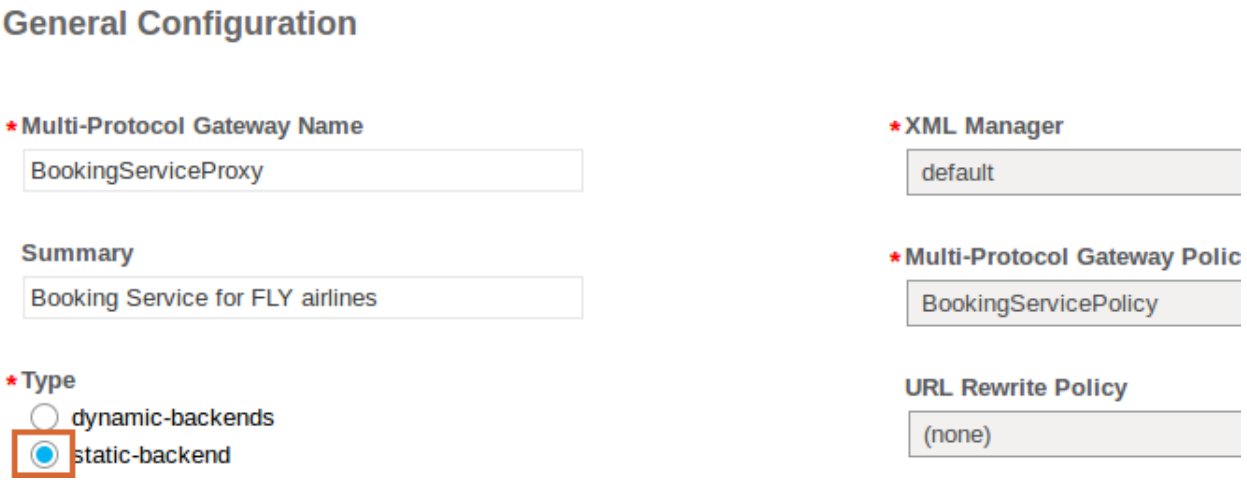
The multi-protocol gateway policy has two document processing rules that are defined:

- **A Request Rule** that accepts request messages from the client with a URL path of any URI and forwards the messages to the back-end server.
- **A Response Rule** accepts any response message from the back-end server and forwards the response to the client.

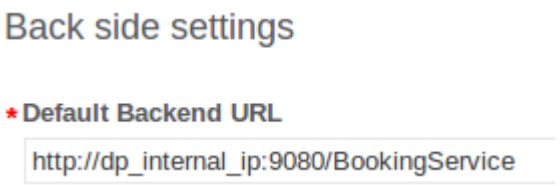
The remaining steps in this section assign the address of the actual `FLYService` web service and various quality of service settings.

__ 16. Set a static back-end connection to the `BookingServiceProxy` web service.

__ a. On the Multi-Protocol Gateway page, select **static-backend** as the back-end connection type.



__ b. Set the back-end URL to: `http://dp_internal_ip:9080/BookingService`



Important

The value **dp_internal_ip** is a predefined “Host Alias” setting, and an administrator typically defines it. The setting assigns an IP address that is associated with any reference to the “dp_internal_ip” variable. This technique makes it easier to migrate DataPower services across environments, such as from Development to Test. In this case study, the back-end web service is running as a service on the gateway itself. In a typical situation, the back-end URL would point to a server that is running somewhere within the trusted domain.

__ 17. Make sure that the **Response Type** field is set to **SOAP**.

Information

The **Request Type** and **Response Type** settings determine the validation steps that the gateway applies to incoming and outgoing messages.

- **JSON** validates the message as well-formed JSON data.
- **Non-XML** represents flat file text or any binary file format. The gateway passes the message itself without modification. However, processing rules can authenticate, authorize, or route the message to another destination.
- **Pass through** literally transmits the message through the gateway without any processing or modification.
- **SOAP** validates the message against the SOAP schema in use. In effect, the gateway checks whether the message contains a valid SOAP envelope for web service messages.
- **XML** verifies that the message contains a well-formed XML document.

__ 18. Make sure that the **Back attachment processing format** is set to **Dynamic**.

Information

Instead of converting large amounts of binary data into text, many web service engines allow binary data to be stored as attachments to the SOAP message. Two common binary data encoding schemes are used in the industry:

- **Multipurpose Internet Mail Extensions (MIME)** defines an encoding format for sending binary information over Internet email messages. Some web services use this scheme to efficiently

transmit binary files as an attachment on a text-based SOAP message. Most web services engines, including the IBM WebSphere web services run time, support this standard.

- **Direct Internet Message Encapsulation (DIME)** was a separate Internet standard that Microsoft proposed as a simpler method to encapsulate binary information in a web service message. Although some web services supported DIME, the newer SOAP Message Transmission Optimization Mechanism (MTOM) specification now supersedes DIME.

Unfortunately, most vendors support only one scheme or the other. DataPower services, such as the multi-protocol gateway, solve this problem by automatically converting attachments between these two types.

The **front attachment processing format** setting determines how the service interprets attachments that are sent from the client. On the other side, the **back attachment processing format** determines how the service encodes attachments in outgoing messages from the service to the back-end resource.

For example, to convert message attachments from a DIME format to MIME, set the **front attachment processing format** to **DIME** and the **back attachment processing format** to **MIME**.

These settings affect messages that contain attachments.

__ 19. Leave the **Back Side Timeout** value set to **120** seconds.

__ 20. Leave the **Stream Output to Back** set to **Buffer Messages**.

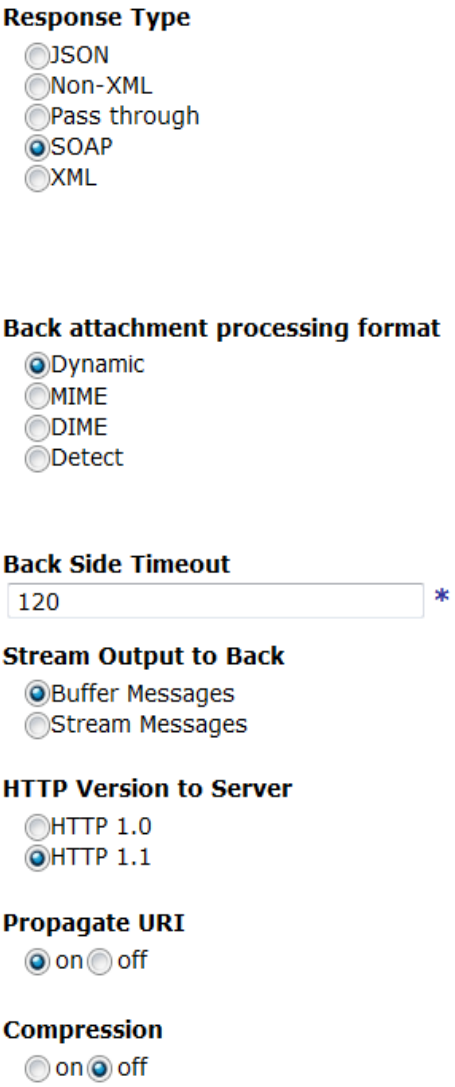
Information

The other setting, **Stream Messages**, continuously sends parts of the request message to the back-end resource unless the gateway encounters some information that causes the message to fail validation.

__ 21. Leave the **HTTP Version to Server** set to **HTTP 1.1**.

__ 22. Leave the **Propagate URI** setting to **on**.

__ 23. Leave the **Compression** set to **off**.
Your back-end resource settings look like the following screen capture.



Section 2: Configure an HTTP front side protocol handler

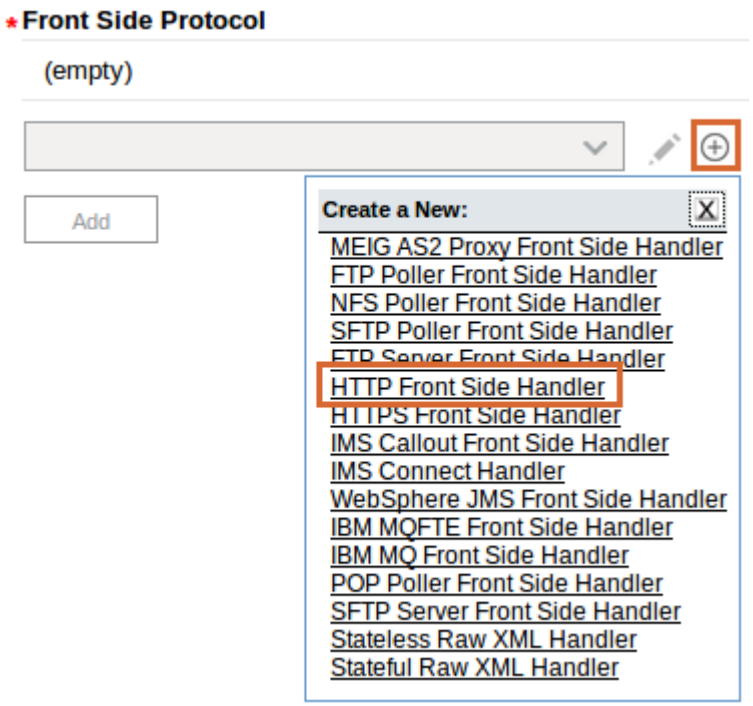
The multi-protocol gateway is configured to forward requests to the FLYService web service. However, the gateway does not support any incoming requests at this moment.

Create a front side protocol handler to receive client requests over an HTTP connection.

__ 1. Create an HTTP front side protocol handler that is named HTTP_12<*nn*>1.

__ a. Beside the **Front Side Protocol** list, click the circled plus sign icon (“new”).

__ b. Select **HTTP Front Side Handler** from the list.



__ c. Configure the new HTTP front side handler with the following values. Leave all other settings at the default values.

- **Name**: `HTTP_12<nn>1` (The format of the name is used to identify the port number that is configured within the object by looking at the name.)
- **Local IP address**: *<dp_public_ip>* (You can choose to use the host alias variable “dp_public_ip” as the address, rather than the hardcoded IP address.)
- **Port Number**: *<mpgw_booking_port>* as assigned by the instructor.

The following screen capture is an example of the settings for student99, and that uses the host alias for the IP address rather than a hardcoded address.

Note

The local IP address value of `<dp_public_ip>` indicates that the front side protocol handler can receive messages on the Ethernet interface on the gateway that is defined for access from the public. You can specify a **Host Alias** with this name so a specific IP address is not indicated in the handler, which makes the service more portable across deployed appliances.

__ d. Click **Apply** to save the changes that are made to the HTTP front side handler.

__ 2. Apply the multi-protocol gateway.

__ a. Click **Apply** to save the changes that are made to the multi-protocol gateway (at the upper right).

__ b. Verify that the Multi-Protocol Gateway status is **up**.

__ c. In the notification area at the top of the page, click **Save Changes**.

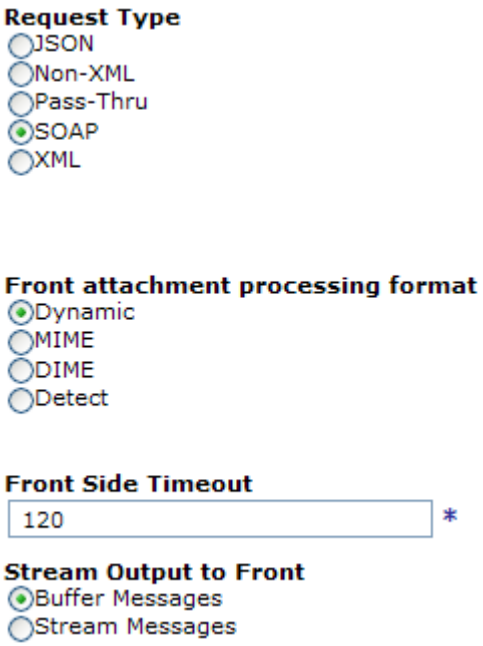
__ 3. Examine the multi-protocol gateway settings for all front side protocol handlers.

__ a. Locate the **Request Type** setting below the Front Side Protocol list. Verify that the expected request message type is **SOAP**, matching the setting for the Response Type.

__ b. Leave the **Front attachment processing format** as **Dynamic**.

__ c. The **Front Side Timeout** value determines the number of seconds that any front side handler idles before abandoning a transaction. Leave this value at **120** seconds, or 2 minutes.

__ d. The **Stream Output to Front** setting determines whether the gateway can start sending portions of the response message back to the client. Leave this setting at **Buffer Messages**.



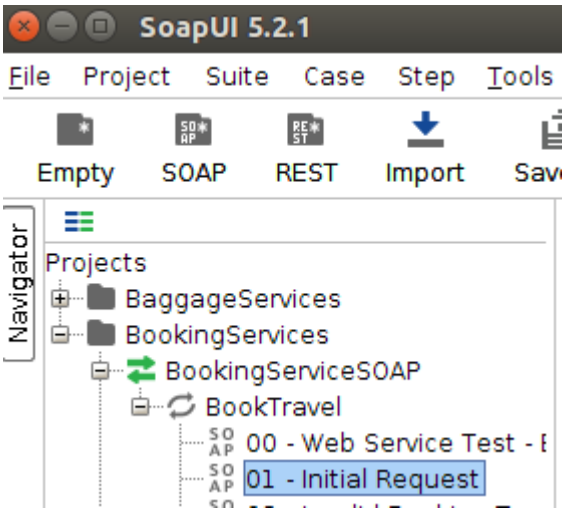
__ e. The multi-protocol gateway `BookingServiceProxy` is now ready for testing.

Section 3: BookingServiceProxy MPGW testing

Test the MPGW configuration by using the SoapUI tool. The following steps ensure that the back-end web service is operational. In addition to testing the availability of the web service, it is also a useful troubleshooting technique to verify network connectivity to the back-end web service.

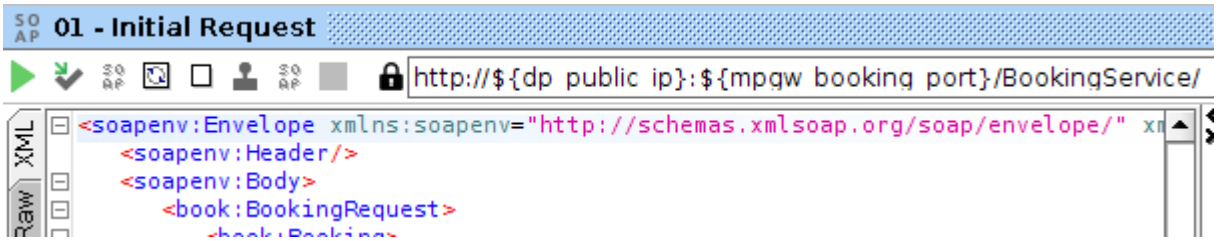
__ 1. On the desktop, locate and start the SoapUI tool if it is not already running.

__ 2. In the project tree, expand the **BookingServices** project until **01 – Initial Request** is visible.



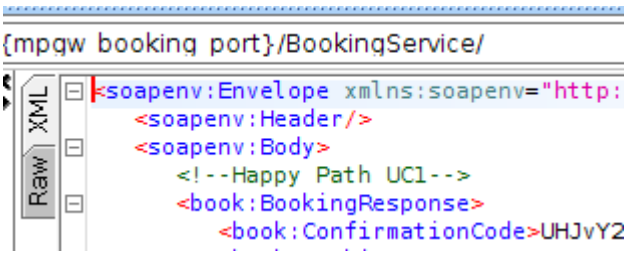
__ 3. Double-click **01 – Initial Request** to open the request window. If a double-click does not work, right-click the request and click **Show Request Editor**.

__ 4. In the URL address field, verify that the pre-filled URL matches the following string: **http://${dp_public_ip}:${mpgw_booking_port}/BookingService** . The string “${xxx}” indicates that a SoapUI global property is referenced. Recall that you set these properties to your specific values in the Lab Setup exercise.



__ 5. Click the green **Submit** arrow that is to the left of the URL address field to send an HTTP POST request to the BookingServiceProxy. The pane on the left side of the Request window contains the SOAP message that is sent as part of the POST.

__ 6. If everything worked properly, you should see <book:BookingResponse> xml tree on the **Response** tab.



If you received an error, you can try to determine the cause by looking at the logs. There’s a convenient **View Log** link that is found towards the top of the multi-protocol gateway configuration page. You can also view the logs from the Control Panel by clicking the **View Logs** icon.

At this point, you created a multi-protocol gateway service that acts as a proxy and verified that it works. In later exercises, you add more functions to the service.

End of exercise

Exercise review and wrap-up

In this exercise, you created a multi-protocol gateway.

The gateway created was the BookingServiceProxy that is built upon through this course, adding more functions. The MPGW was configured with a request and response rule that matches all incoming URIs.

You were also introduced to the SoapUI tool, which is used throughout this course as a testing tool. SoapUI sends SOAP messages to the DataPower gateway, and you can focus on DataPower development instead.

Exercise 3. Enhancing the BookingService gateway

Estimated time

01:00

Overview

This exercise shows you how to add validation, filtering, and transformation to a multi-protocol gateway (MPGW). These steps involve working with the actions and policy editor. You use the system log to debug the service behavior.

Objectives

After completing this exercise, you should be able to:

- Perform advanced configuration of an MPGW
- Configure a document processing policy with more actions
- Test the MPGW policy by using the graphical SoapUI tool
- Perform basic debugging by using the system log

Introduction

In most cases, multi-protocol gateways (MPGWs) are designed to do much more than proxy a request to a back-end application server. MPGWs are used to validate the input message so that only appropriate messages reach the back-end servers. In this exercise, you use several techniques to check messages. Schema validation and SOAP envelope validation confirm the basic format of the message. Filtering checks the contents of the message, and decides on whether to allow the message to continue to be processed. Lastly, you might want to change the format and contents of the message, either on the input side or on the response side. In this exercise, you change the response message to decode a field, obscure a sensitive field, and eliminate unneeded fields.

Requirements

To complete this exercise, you need:

- Access to the DataPower gateway
- Completion of the previous exercises (see the Preface section in the exercise instructions for details)
- SoapUI, to send requests to the DataPower gateway

- The BookingServiceBackend web service that runs on the DataPower gateway in the `FLYServices` domain
- Access to the `<lab_files>` directory

Exercise instructions

In this exercise, you adjust the configuration of your BookingServiceProxy MPGW.

Preface

- Before starting this lab, complete the steps in Exercise 2: Creating a BookingService gateway.
- Remember to use the domain and port address that you were assigned in the exercise setup. **Do *not*** use the default domain.
- The references in exercise instructions refer to the following values:
  - <*lab_files*>: Location of the student lab files. Default location is: `/usr/labfiles/dp/`
  - <image_ip>: IP address of the student image (use `/sbin/ifconfig` from a terminal window to obtain value).
  - <*dp_internal_ip*>: IP address of the DataPower gateway development and administrative functions that are used by internal resources such as developers.
  - <*dp_public_ip*>: IP address of the public services on the gateway that is used by customer and clients.
  - <*dp_WebGUI_port*>: Port of the WebGUI and Blueprint Console. The default port is `9090`.
  - <*nn*>: Assigned student number. If no instructor is present, use “`01`”.
  - <*studentnn*>: Assigned user name and user account. If no instructor is present, use “`student01`”.
  - <*studentnn_password*>: Account password. In most cases, the initial value is the same as the user name. You are prompted to create a password on first use. Write it down.
  - <*studentnn_domain*>: Application domain that the user account is assigned to. If no instructor is present, use “`student01_domain`”.
  - <*FLY_booking_port*>: Port number that the back-end BookingServices web services listen on. The default port is 9080.
  - <*mpgw_booking_port*>: `12`*nn*`1`, where “*nn*” is the two-digit student number. This number is the listener port for the MPGW that proxies the BookingService web service.

3.1. Initialize the lab environment

Some setup activities are required to properly configure the lab environment and determine IP addresses and ports.

__ 1. If you have not yet performed the setup activities, you must go to **Appendix B: Lab environment setup**. Complete those activities before proceeding.

These activities need to be performed only once for the course.

3.2. Add more capability to the BookingServiceProxy MPGW

The BookingServiceProxy multi-protocol gateway is expanded to include more capabilities: Schema validation, SOAP envelope validation, content-based filtering, SQL injection protection, and message transformation by using a style sheet. You add new actions to the service policy of the BookingServiceProxy, and test the results with the SoapUI tool.

The final result is an airline reservation that is booked by using the DataPower BookingServiceProxy MPGW.

This lab implements the scenario that is described in six essential steps:

Section 1: Schema Validation

An XML schema describes the structure of an XML document. Validating an XML document against a schema is one step to assuring that the structure and content of the document is valid and safe. The process of validating an XML document against a schema is generally considered to be processor intensive, resulting in increased server load. For this reason, organizations often disable schema validation to reduce load (and cost) on application servers, especially when they are running on a mainframe. This approach is generally considered a security risk.

Information

Although it is not covered in this exercise, a similar capability exists for validation of JSON input.

IBM DataPower Gateway appliances solve this problem by providing near wirespeed schema validation to messages before they reach the application server. Messages that fail validation are rejected by default (this behavior can be customized).

In this section, you add a Validate action to your existing request rule within the service policy. The Validate action determines the schema from a WSDL that you upload.

__ 1. Connect to the DataPower WebGUI:
`https://<dp_internal_ip>:<dp_WebGUI_port>`

__ 2. Enter the User Name of **<*studentnn*>**, Password of **<*studentnn_password*>**, and Domain of **<*studentnn_domain*>**.

__ 3. Click **Login**.

__ 4. Click **Blueprint Console**.

__ 5. The All Services page is displayed. Click the **BookingServiceProxy** multi-protocol gateway link.

__ 6. Your MPGW is displayed. You need to edit the service policy to add the new behavior. Click the pencil (edit) button in the **Multi-Protocol Gateway Policy** field.

__ 7. Expand the policy editor so that you can see all the configured rules at the bottom. Make sure that the **Client to Server** rule is selected (it is bold). Currently, only a Match action and Results action are in the rule.

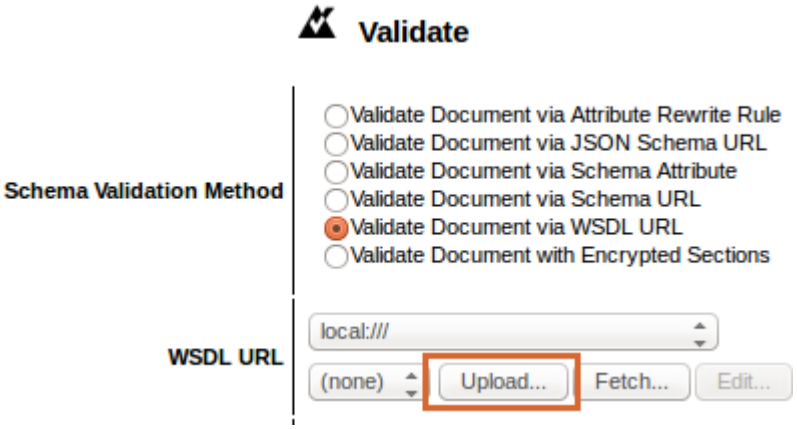
__ 8. Click and drag a **Validate** action and drop it to the right of the Match action.

__ 9. Configure the new Validate action (outlined in yellow).

__ a. Double-click the **Validate** action to open the **Validate** action configuration window.

__ b. Select the **Validate Document via WSDL URL** in the Scheme Validation Method section.

__ c. Click **Upload** to upload the appropriate WSDL document.



__ d. Click **Browse**.

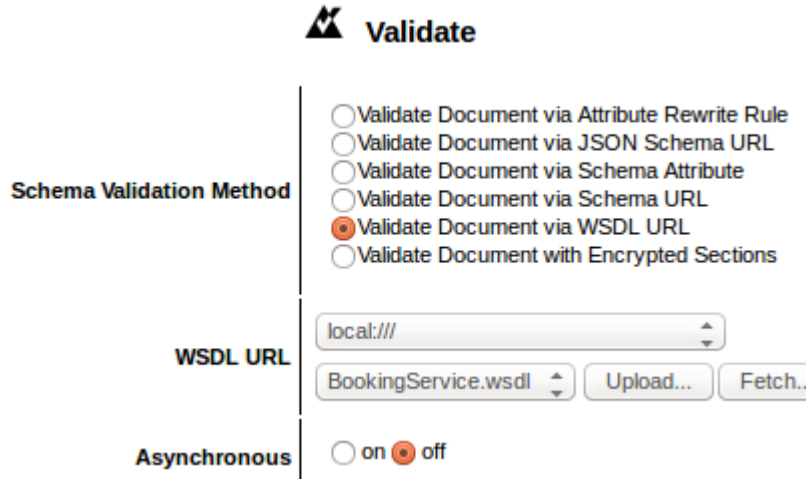
__ e. Select **BookingService.wsdl** in the `<lab_files>/BookingService/` directory.

__ f. Click **Open**.

__ g. Click **Upload**.

__ h. Click **Continue** on the “Upload successful” page.

__ i. After uploading the WSDL file, make sure that the Validate configuration window contains the required values.



__ j. Click **View**. A window opens that displays the `BookingService.wsdl` file.

Scroll to the **BookingType** element. You can widen the window to eliminate the word wrap. Notice that the BookingType element restricts its values to "I" for Internal and "E" for External. This schema restriction is tested in the following section.

```
<xsd:element name="ReservationCode" type="xsd:string" />
<xsd:element name="BookingType">
        <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="I" />
                        <xsd:enumeration value="E" />
                </xsd:restriction>
        </xsd:simpleType>
</xsd:element>
```

__ k. Close the viewing window.

__ l. Click **Done**. The Validate action is now configured.

__ 10. Click **Apply Policy** at the top of the policy editor to activate your changes.

__ 11. Click **Close Window** in the upper-right corner of the policy editor.

__ 12. Click **Apply** on the Multi-Protocol Gateway window if it is enabled.

Section 2: Schema Validation Test

__ 1. If the SoapUI tool is not already open, open it from the SoapUI icon that is on the desktop.

__ 2. In the SoapUI window, expand the BookingServices project until you can see the **01 – Initial Request** request, if it is not already open.

__ 3. The URL address field should still contain your specific port number. Click the green **Submit** arrow to POST the request again. The request should be successful as it was before. This result indicates that the message successfully passed schema validation.

__ 4. If you want, close the “01 – Initial Request” window.

__ 5. Open **02 – Invalid Booking Type**.

In this sample payload you see that the <book:BookingType> has the value of EXTERNAL, which is not valid against the schema.

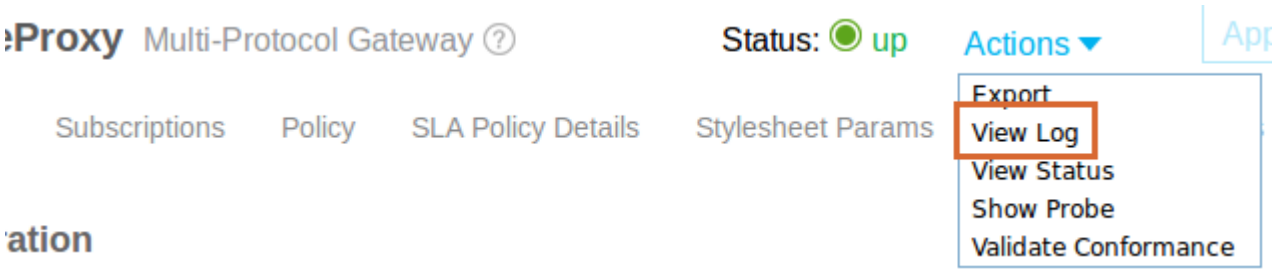
__ 6. Click the green **Submit** arrow to send the request to your MPGW.

__ 7. Because “EXTERNAL” is not a valid BookingType, it failed schema validation that resulted in a SOAP fault back to the client.

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Body>
      <env:Fault>
         <faultcode>env:Client</faultcode>
         <faultstring>Internal Error (from client)</faultstring>
      </env:Fault>
   </env:Body>
</env:Envelope>
```

The returned error message indicates that an internal error occurred but no other details are provided. This generic error response is by design to prevent malicious attackers from gaining detailed information about the underlying service. You can see detailed information about the failure in the DataPower log.

__ 8. On the Multi-Protocol Gateway configuration page, click the **Actions > View Log** link from the upper-right side of the page.



__ 9. A log window opens. It reveals the underlying reason for the “Internal Error” message. A later lecture discusses how to customize the error messages to the client.

9:14:19 AM	mpgw	error	67072	error	172.16.80.145	0x00230001	mpgw (BookingServiceProxy): Dynamic Execution Error
9:14:19 AM	multistep	error	67072	request	172.16.80.145	0x80c00009	mpgw (BookingServiceProxy): request BookingServicePolicy_rule_0 #1 validate: 'INPUT wsdl local:///BookingService.wsdl' failed: http://172.16.78.23:12311/BookingService/: cvc-simple-type 1: element {http://www.ibm.com/datapower/FLY/BookingService/}BookingType value 'EXTERNAL' is not a valid instance of the element type
9:14:19 AM	multistep	error	67072	request	172.16.80.145	0x01d30003	mpgw (BookingServiceProxy): Schema Validation Error
9:14:19 AM	schema	error	67072	request	172.16.80.145	0x80c00010	mpgw (BookingServiceProxy): Processing of 'local:///BookingService.wsdl' stopped: http://172.16.78.23:12311/BookingService/: cvc-simple-type 1: element {http://www.ibm.com/datapower/FLY/BookingService/}BookingType value 'EXTERNAL' is not a valid instance of the element type

__ 10. Close the log window.

Section 3: SOAP Envelope Schema Validation

The Multi-Protocol Gateway service that you configured expects requests and responses to conform to SOAP standards. This setting is found towards the middle of the Multi-Protocol Gateway main configuration page (see following screen capture).

__ 1. In SoapUI, close the **02 – Invalid Booking Type** and open **03 – Missing SOAP Envelope**.

__ 2. Note that the input document does not have an enclosing SOAP envelope. Click the green **Submit** arrow to POST the XML to BookingServiceProxy.

__ 3. The request should fail again. To see details about the failure, click the **Actions > View Log** link on the Multi-Protocol Gateway configuration page.

error	59857	error	172.16.80.145	0x00d30002	mpgw (BookingServiceProxy): Invalid SOAP envelope
error	59857	request	172.16.80.145	0x80c00008	mpgw (BookingServiceProxy): rule (BookingServicePolicy_rule_0): Implied action Parsing input as SOAP, attempt pipeline. failed: http://172.16.78.23:12311/BookingService/:1: cvc-wildcard 2: unrecognized element {http://www.ibm.com/datapower/FLY/BookingService}BookingRequest
error	59857	request	172.16.80.145	0x80e003aa	mpgw (BookingServiceProxy): http://172.16.78.23:12311/BookingService/:1: cvc-wildcard 2: unrecognized element {http://www.ibm.com/datapower/FLY/BookingService}BookingRequest

__ 4. If you want, you can close the log window. If you keep it open, you need to click **Refresh Log** to get the newest entries.

Section 4: Content-based Filtering

You can easily extend the built-in threat protection by defining custom filters. A custom filter is an XSL template that makes an “accept” or “reject” decision based on defined custom logic.

The “accept” and “reject” decision are accomplished by using special built-in extension functions for XSL. The <dp:accept> and <dp:reject> extension functions are used to tell the processing rule how to proceed with the message. The following XSL template inspects the <ReservervationCode> element to make sure that it starts with the string “JK”.

Listing of file: ReservationCode_Filter.xsl

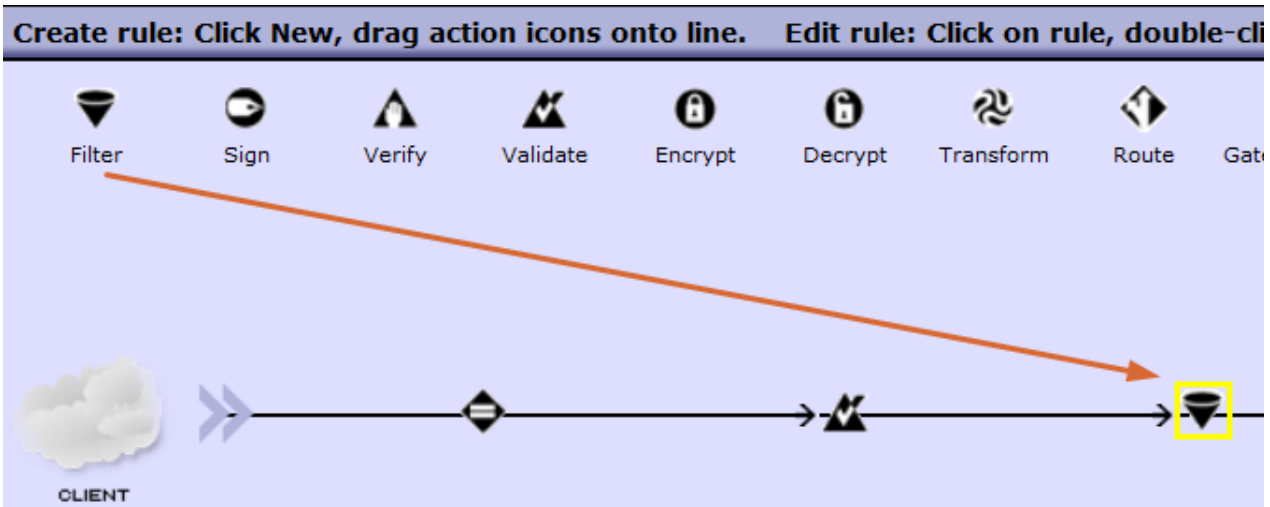
```
<xsl:template match="/">
    <xsl:choose>
        <xsl:when test="starts-with(//book:ReservationCode, 'JK')">
            <dp:accept />
        </xsl:when>
        <xsl:otherwise>
            <dp:reject>Reservation Code was not for FLY Airlines</dp:r

        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Now you add a filter action to your processing rule.

__ 1. Edit the MultiProtocol Gateway Policy.

__ 2. In the policy editor window, select the request rule that contains the Validate action. This selection makes the rule active in the edit area.

__ 3. Drag a **Filter** action onto the rule as shown in the following screen capture. Be sure to drag the Filter action after the Validate action.



__ 4. Double-click the yellow outlined filter action to complete its configuration.

__ a. Click **Upload** to upload the appropriate file.

__ b. Click **Browse**.

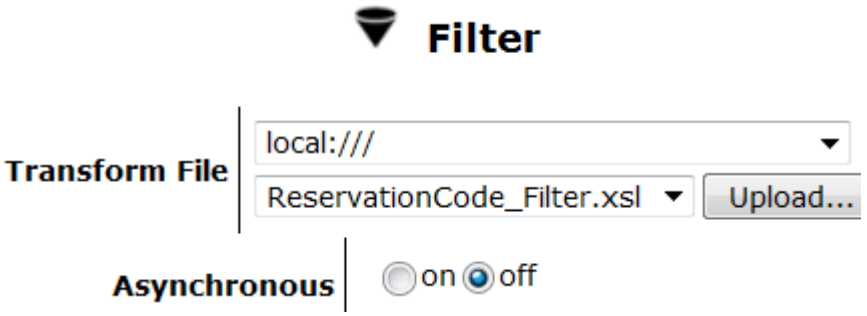
__ c. Select **ReservationCode_Filter.xsl** in the *<lab_files>*/BookingService/ directory.

__ d. Click **Open**.

__ e. Click **Upload**.

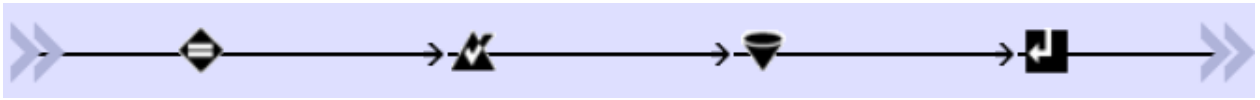
__ f. Click **Continue** on the “Upload successful” page.

__ g. After uploading the transform file, make sure that the Filter window contains the required values.



__ 5. In the Configure Filter Action window, click **Done**.

The processing policy should now look like the following screen capture.



__ 6. Click **Apply Policy** to make your changes active.

__ 7. In SoapUI, close **03 – Missing SOAP Envelope** and open **04 – ReservationCode Invalid**. Notice that the reservation code does not start with “JK”.

__ 8. Click the green **Submit** arrow to POST the request to BookingServiceProxy. You should receive a SOAP fault with an error message as shown in the following image.

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Body>
      <env:Fault>
         <faultcode>env:Client</faultcode>
         <faultstring>Reservation Code was not for FLY Airlines (from client)</faultstring>
      </env:Fault>
   </env:Body>
</env:Envelope>
```

Section 5: SQL Injection Threat Filtering

SQL Injection is an attack technique that is used to exploit websites and services that construct SQL statements from user-supplied input. For example, assume that a web service expects a SOAP request that contains a <last-name> element that is used for looking up a customer.

```
<soap:Body>
   <customer-lookup>
      <last-name>KAPLAN</last-name>
   </customer-lookup>
</soap:Body>
```

The web service uses an SQL statement with substitution parameters similar to the following SQL snippet:

```
SELECT * FROM EMPLOYEE WHERE LASTNAME = ?
```

After the substitution takes place, the resultant SQL statement will be:

```
SELECT * FROM EMPLOYEE WHERE LASTNAME = 'KAPLAN'
```

However, if the value submitted in the <last-name> element contained a malicious SQL injection threat, it might look like this sample code:

```
<soap:Body>
   <customer-lookup>
      <last-name>KAPLAN' OR '1'='1</last-name>
   </customer-lookup>
</soap:Body>
```

The SQL statement would become:

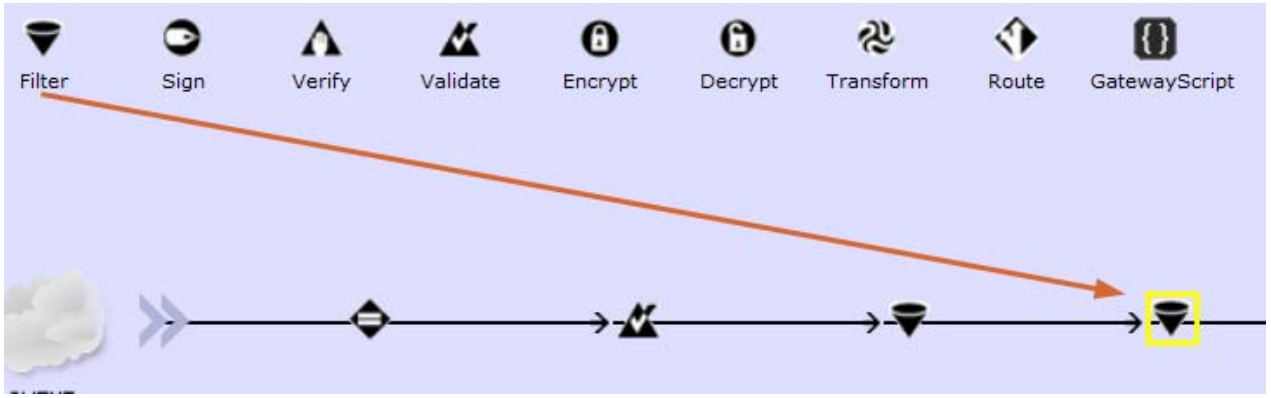
```
SELECT * FROM EMPLOYEE WHERE LASTNAME = 'KAPLAN' OR '1' = '1'
```

The service returns the details about ALL employees, since the WHERE clause evaluates to true for every record in the EMPLOYEE table (because of the '1' = '1' clause).

IBM DataPower Gateway Appliances can protect against such SQL injection threats by using a special SQL injection threat filter. It works the same way as the filter you tried in the previous steps, except that the logic is a bit more complex.

The SQL Injection Threat filter has two parts: the base style sheet filter (that uses <accept/> and <reject/>), and an XML file that contains the various patterns to search for. Keeping the patterns in a separate XML file makes it easier to create more customized patterns.

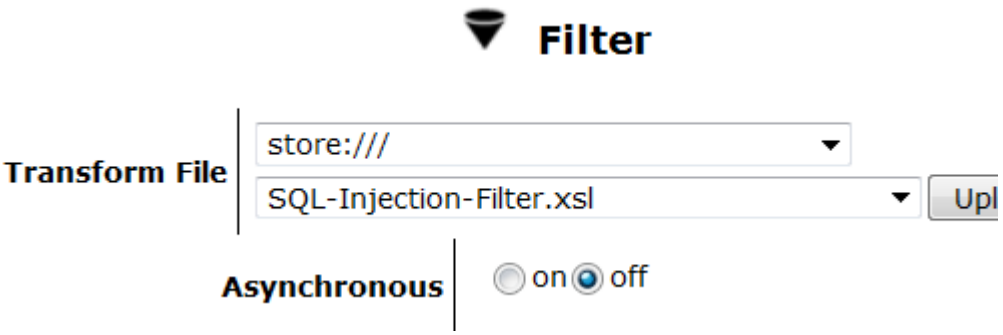
__ 1. In the policy editor window, drag another **Filter** action onto the processing rule to the right of the previously added filter action.



__ 2. Double-click the yellow outlined filter action to complete its configuration.

__ 3. Change the upper drop-down box for the **Transform File** to show: **store:///**

__ 4. In the lower drop-down box, select: **SQL-Injection-Filter.xsl**



__ 5. Click **Done**.

__ 6. Click **Apply Policy** to activate these changes.

The policy now protects against malicious SQL injection threats. Your next test contains a SOAP message with an SQL injection threat in it. The contents of the book:HolderName element contain the threat:

```
<book:PaymentCardDetails>
   ...
      <book:HolderName>Olivia Holdt' or '1'='1'</book:HolderName>
```

__ 7. In SoapUI, close the **04 – ReservationCode Invalid** and open **05 – SQL Injection**. Notice the injection attack as shown previously in the test case.

__ 8. Click the green **Submit** arrow to POST the message to BookingServiceProxy.

__ 9. The request should fail due to “Message contains restricted content (from client)”.

__ 10. If you want, you can open the log. Notice the error messages that indicates the SQL injection attack detection.

Section 6: Transforming with XSL

At the heart of DataPower Gateway appliances is a high-speed XSL compiler and execution engine. In fact, most of the built-in functions are engineered by using XSL. Some of the built-in style sheets can be found in the *store* directory. XSL developers can easily copy and modify the IBM provided style sheets to create functions or support emerging standards before IBM makes them available.

When a style sheet is referenced for the first time, it is compiled by using a patented optimizing XSL compiler for execution on specialized IBM DataPower hardware, then cached in memory for high-speed recall and execution.

IBM augmented XSL with a rich set of *extension functions* that enable you to easily add complex processing functions to your processing rules. For example, extension functions exist for performing base-64 encoding and decoding, encryption and decryption, and date/time functions. Functions also exist for communicating with off-box web services and LDAP servers.

Information

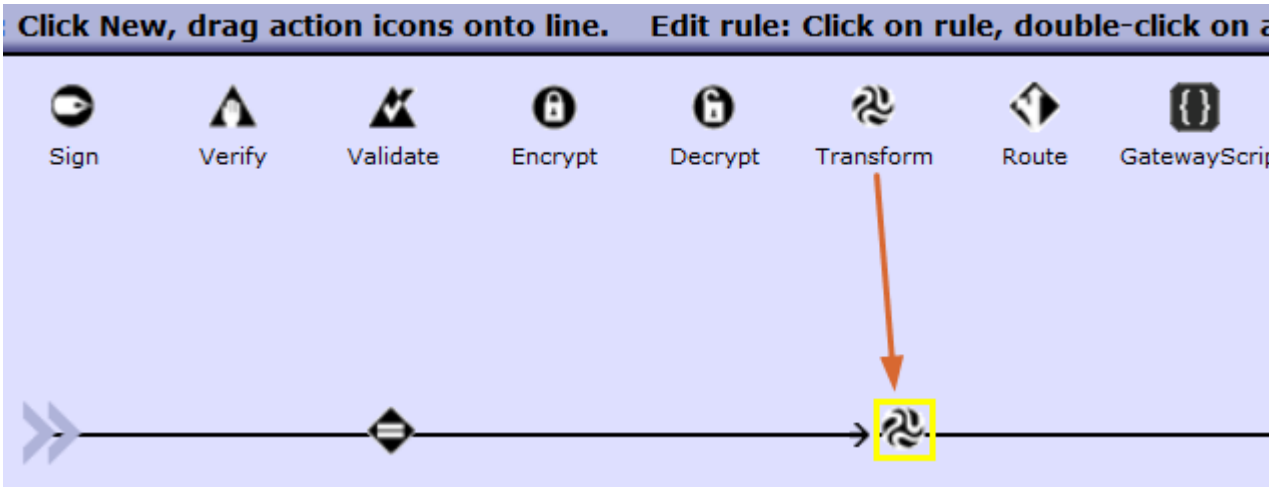
With each release of the firmware, more of the extension functions are mirrored in GatewayScript functions.

In this section, you are introduced to how XSL templates are used within processing rules.

In the following steps, you add a transform action to the response (server to client) rule instead of the request rule. Therefore, because the transform action modifies the overall structure of the message, it does not match the schema that the backend service is expecting, the request fails. To avoid this situation, you modify the response that is returned to SoapUI.

__ 1. In the policy editor, towards the bottom, click the **Server to Client** rule to make it the active rule in the editor.

__ 2. Click and drag a **Transform** action and drop it after the match action.



__ 3. Double-click the yellow outlined transform action to display its configuration settings.

__ 4. Click **Upload** to upload the appropriate **Transform File**.

__ 5. Click **Browse**.

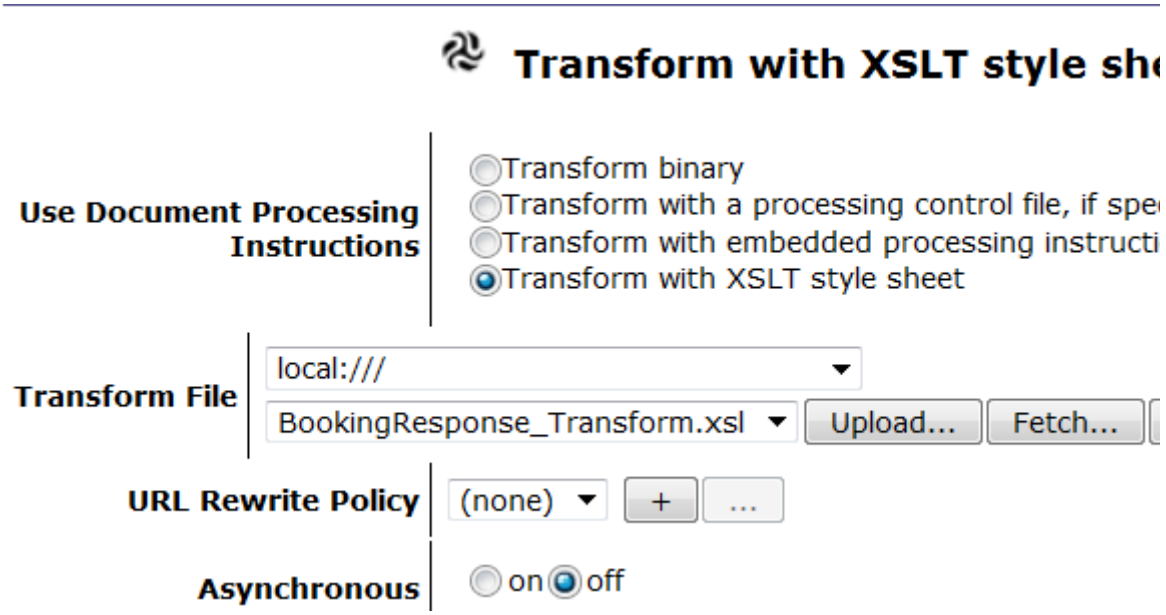
__ 6. Select **BookingResponse_Transform.xsl** in the `<lab_files>/BookingService/` directory.

__ 7. Click **Open**.

__ 8. Click **Upload**.

__ 9. Click **Continue** on the “Upload successful” page.

__ 10. After uploading the transform file, make sure that the Filter window contains the required values.



__ 11. Click **Done** to save the transform action.

__ 12. Click **Apply Policy** to apply the changes to the overall policy.

__ 13. Click the **Close Window** link to close the policy editor.

__ 14. If needed, click **Apply** on the Multi-Protocol Gateway configuration page.

You're now ready to run another transaction through your multi-protocol gateway service. Before you do that, look at what the XSL template does to the message.

In all the successful responses that you received so far, all of the request details were included in the response, and the <book: ConfirmationCode> was base64 encoded.

The BookingResponse_Transform.xsl template looks for a few different patterns:

- When a book:Expiry or book:CVV tags are found, no additional steps are taken and these tags and any children of these tags are removed from the output.
- When a book:ConfirmationCode tag is encountered, it changes it into a <book:ConfirmationText> tag and then decodes the original tag's value. dp:decode() is an extension function that performs the base64 decoding.
- When a book:Number tag is encountered, it uses the substring() function to get the last four numbers and replace the other numbers with asterisks.
- For anything else that does not match, an identity transform is used to copy it to the output document.

Partial listing of file: `BookingResponse_Transform.xsl`

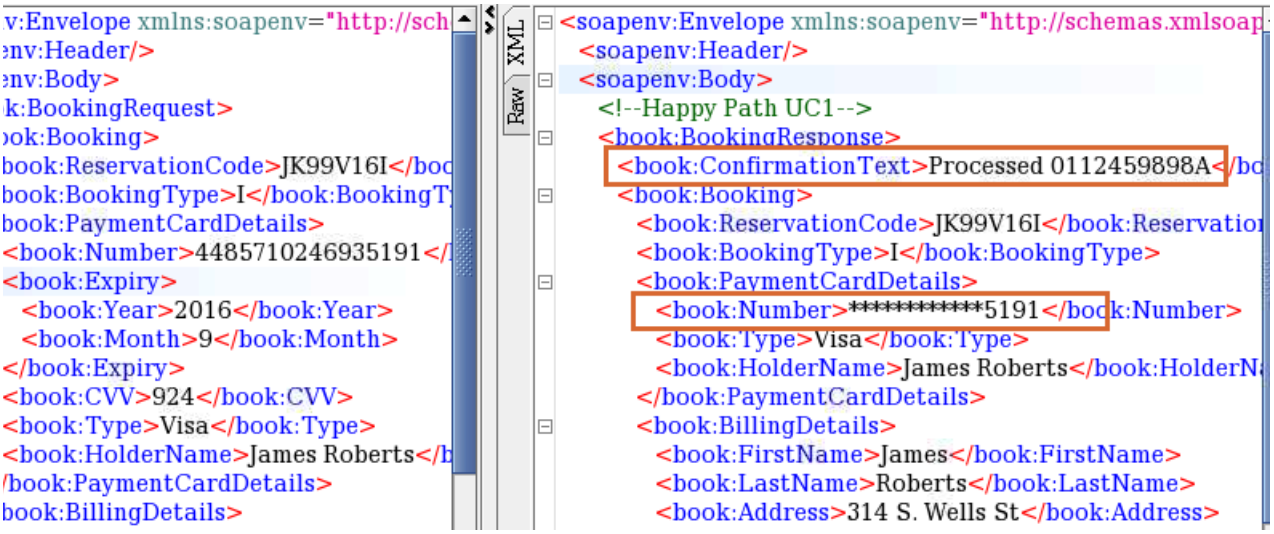
```
<xsl:template match="//book:Expiry | //book:CVV">
   <!--DO NOTHING-->
</xsl:template>
<xsl:template match="book:Number">
   <xsl:element name="book:Number">
      <xsl:variable name="stl" select="string-length(.)"/>
      <xsl:text>**********</xsl:text>
      <xsl:value-of select="substring(.,($stl)-3,($stl))"/>
   </xsl:element>
</xsl:template>
```

Information

This transform can be done by using GatewayScript. Because the input document is XML, XSL is a better choice.

__ 15. In SoapUI, close **05 – SQL Injection** and open **01 – Initial Request**.

__ 16. Click the green **Submit** arrow to POST the message to BookingServiceProxy, then inspect the response. Notice that the book:ConfirmationCode tag was replaced with a book:ConfirmationText tag, and that its contents are no longer base64 encoded. Also, the book:Expiry and book:CVV is removed and the book:Number is masked.



__ 17. When everything is working properly, you can save your configuration by clicking **Save changes** in the message area at the top of the page.

End of exercise

Exercise review and wrap-up

In this exercise, you configured the BookingServiceProxy MPGW.

The BookingServiceProxy multi-protocol gateway is expanded to include more functions. First, a Validate action is configured, then tested, to perform schema validation. Next, SOAP envelope schema validation is configured and tested. Thirdly, the MPGW is configured for Content Based Filtering. Then, the MPGW is configured for SQL injection threat protection. Finally, the BookingServiceProxy is configured to transform the response message by using a custom XSL style sheet and XPath.

The final result is an airline reservation that is booked by using the DataPower BookingServiceProxy.

Exercise 4. Adding error handling to a service policy

Estimated time

00:45

Overview

In this exercise, you add an On Error action and an error rule to a service policy, and create an error policy at the service level.

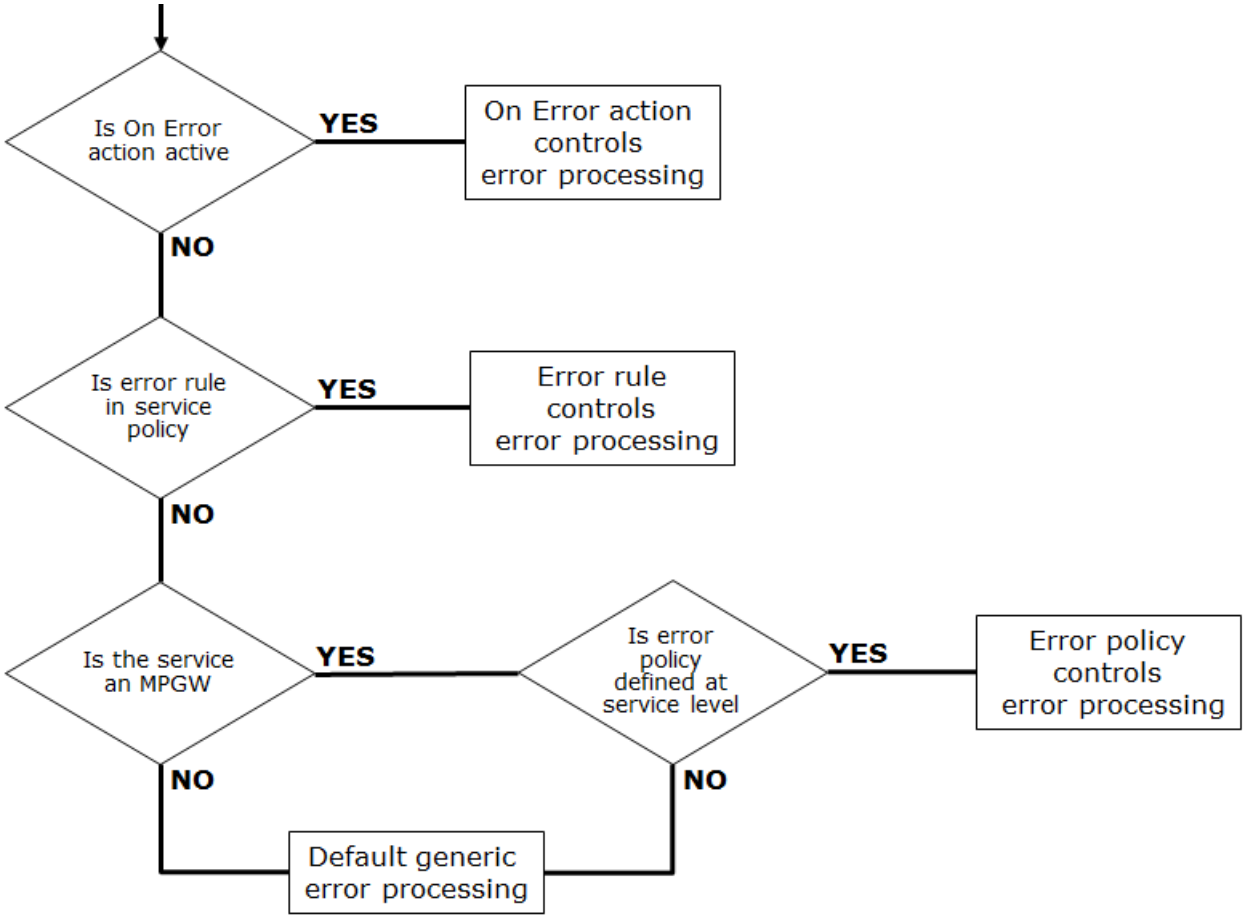
Objectives

After completing this exercise, you should be able to:

- Configure an error policy at the MPGW service level
- Configure a service policy with an On Error action
- Configure a service policy with an Error rule

Introduction

Message processing within a service is expected to occasionally have errors. Policies can use **On Error** actions and **error rules** to deal with errors that occur within rule processing. Also, one can use an error policy at the MPGW service level to deal with errors that are not handled at the service policy level. This exercise creates an error policy, and adds an error rule and an **On Error** action to provide documentation on invalid messages that are sent to the MPGW.



Requirements

To complete this exercise, you need:

- Access to the DataPower gateway
- Completion of the previous exercises (see the Preface section in the exercise instructions for details)
- SoapUI, to send requests to the DataPower gateway
- The BookingServiceBackend web service that runs on the DataPower gateway in the `FLYServices` domain
- Access to the `<lab_files>` directory

Exercise instructions

Preface

- Before starting this lab, complete the steps in Exercise 1: First exposure to the DataPower development environment.
- This exercise also depends on the previous completion of Exercise 3: Enhancing the BookingService gateway.
- Remember to use the domain and port address that you were assigned in the exercise setup. **Do *not*** use the default domain.
- The references in the exercise instructions refer to the following values:
  - <*lab_files*>: Location of the student lab files. Default location is: `/usr/labfiles/dp/`
  - <image_ip>: IP address of the student image (use `/sbin/ifconfig` from a terminal window to obtain value).
  - <*dp_internal_ip*>: IP address of the DataPower gateway development and administrative functions that are used by internal resources such as developers.
  - <*dp_public_ip*>: IP address of the public services on the gateway that is used by customer and clients.
  - <*dp_WebGUI_port*>: Port of the WebGUI. The default port is `9090`.
  - <*nn*>: Assigned student number. If no instructor is present, use “`01`”.
  - <*studentnn*>: Assigned user name and user account. If no instructor is present, use “`student01`”.
  - <*studentnn_password*>: Account password. In most cases, the initial value is the same as the user name. You are prompted to create a password on first use. Write it down.
  - <*studentnn_domain*>: Application domain that the user account is assigned to. If no instructor is present, use “`student01_domain`”.
  - <*FLY_booking_port*>: Port number that the back-end BookingServices web services listen on. The default port is 9080.
  - <*mpgw_booking_port*>: `12`*nn*`1`, where “*nn*” is the two-digit student number. This number is the listener port for the MPGW that proxies the BookingService web service.

4.1. Initialize the lab environment

Some setup activities are required to properly configure the lab environment and determine IP addresses and ports.

__ 1. If you have not yet performed the setup activities, you must go to **Appendix B: Lab environment setup**. Complete those activities before proceeding.

These activities need to be performed only once for the course.

4.2. Add error processing

For this part, you add default error processing to the multi-protocol gateway by creating an error policy. The error policy takes control when an error occurs that conforms to the error match condition you create. Error policy is where the default error processing is configured, allowing more granular error processing to occur at the error rule and On Error action levels.

Next, you add error handling to an existing document processing policy. This error rule takes control when your validation or transformation rule has an error. The error rule contains a **Transform** action that produces an HTML document that indicates the error.

Finally, you add an **On Error** action to the existing request rule. **On Error** actions specify the processing rule that is called during any errors in subsequent actions. The request rule to specify different error handling, processing rules at different steps within the request rule, is allowed. This action also applies to response rules.

This exercise is divided into the following sections:

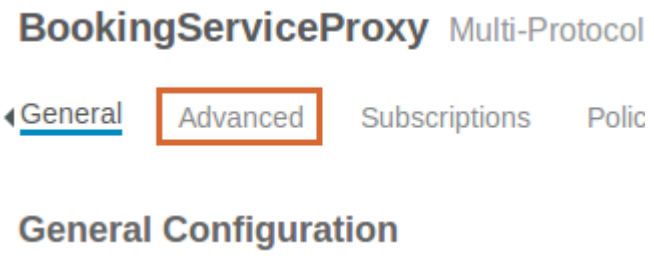
Section 1: Add an Error Policy

__ 1. Log in to your domain in the WebGUI.

__ 2. Switch to the Blueprint Console.

__ 3. Open your multi-protocol gateway **BookingServiceProxy** for editing.

__ 4. On the Configure Multi-Protocol Gateway page, click the **Advanced** tab.



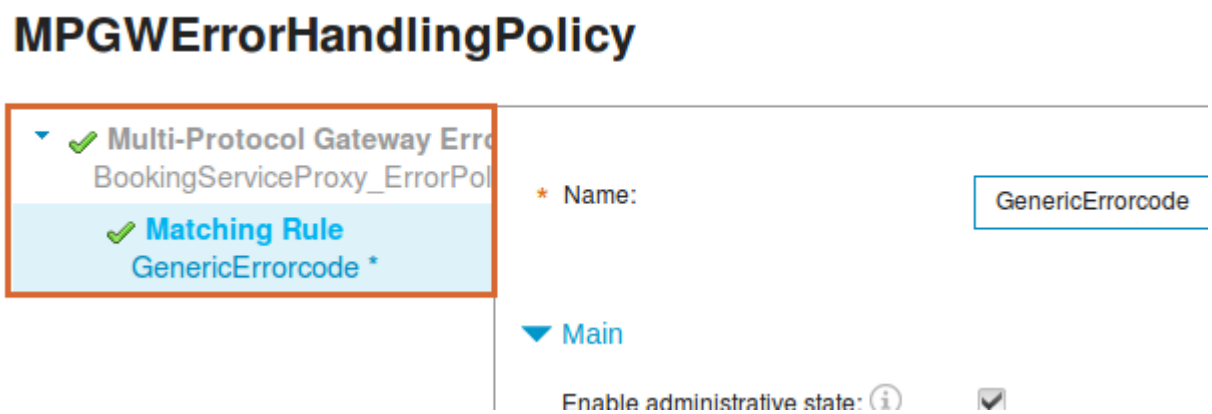
__ 5. Scroll down, and add an Error Policy by clicking new (+).

__ 6. Name the new error policy: `BookingServiceProxy_ErrorPolicy`

__ 7. Click **Add** to create a Policy Map.

__ 8. For the Matching Rule, click new (+) to create a matching rule.

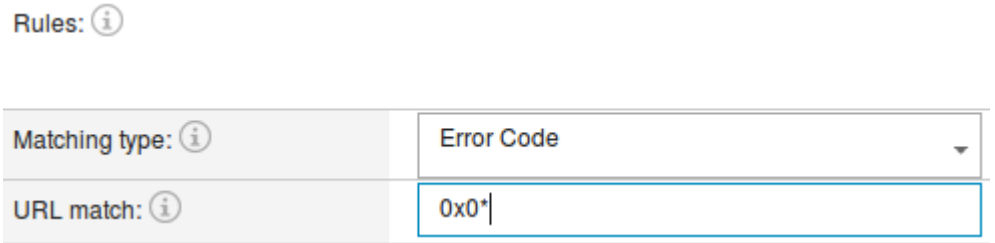
__ 9. Notice that an object tree appears on the left. Because the error policy object references the matching rule object, the tree is presented to show you where in the containment you are.



__ 10. In the **Main** area of the Matching Rule configuration page, enter a name of: `GenericErrorcode`

__ 11. In the **Rules** section, click **Add** to add a rule.

__ 12. In the Edit Rules window, select a matching type of **Error Code**, and enter an error code of `0x0*` (zeros). This value matches all generic error codes.



Note

In firmware V7.5.0.0, the values field remains labeled as “URL match” instead of “Error code”. This label mismatch does not cause a problem.

__ 13. Click **Apply** to save the new matching rule.

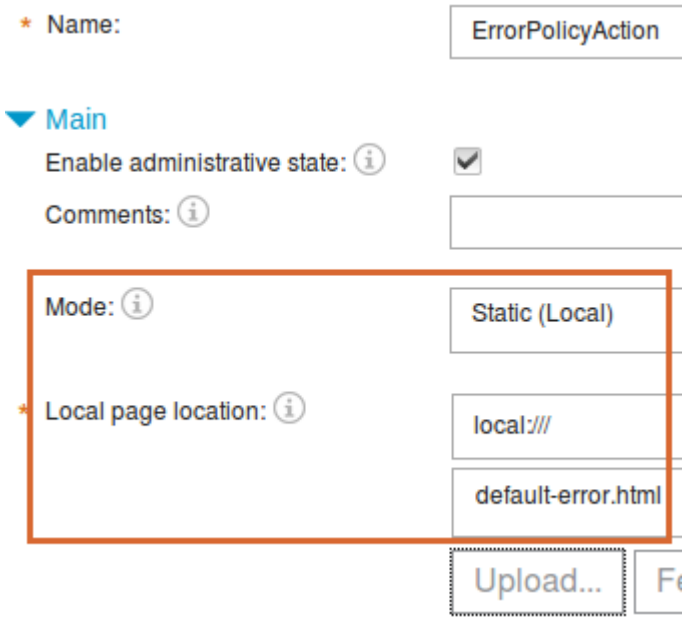
__ 14. In the Policy Maps area, click new (+) to create an **Error Action**.

__ 15. In the **Main** area of the page, enter a name of: `ErrorPolicyAction`

__ 16. Select a Mode of **Static (Local)**.

__ 17. In the **local page location** section, click **Upload > Browse**, and go to the file: *<LAB_FILES>*`/errors/default-error.html`

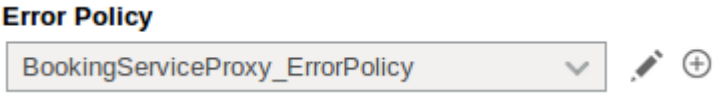
__ 18. Click **Open** and **Upload** to retrieve the html file. Because a static mode was selected, the identified file is a static HTML file that is returned to the client.



__ 19. Click **Apply** to exit the error policy window.

__ 20. The error policy page closes.

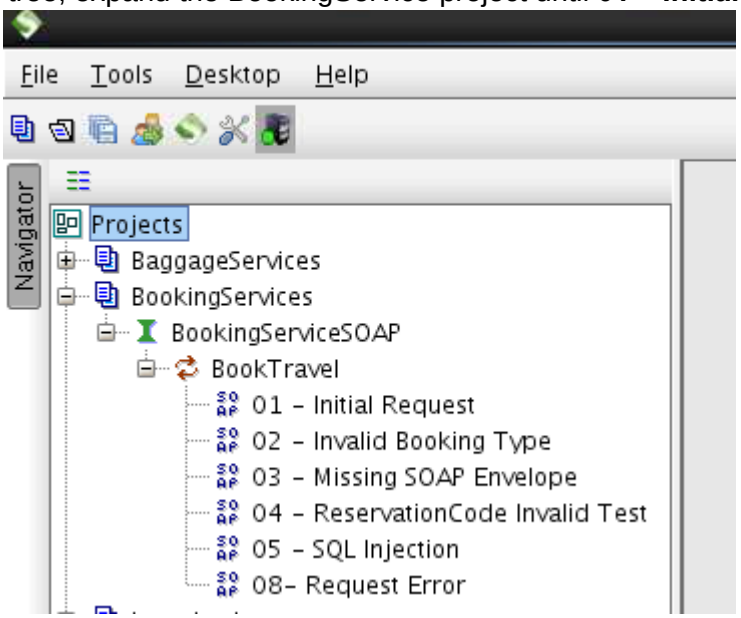
__ 21. Click **Apply** again to save the Error Policy in the multi-protocol gateway.



Section 2: Test the default error policy

__ 1. If SoapUI is not already open, open SoapUI by clicking the icon that is on the desktop.

__ 2. In the project tree, expand the BookingService project until **01 – Initial Request** is visible.

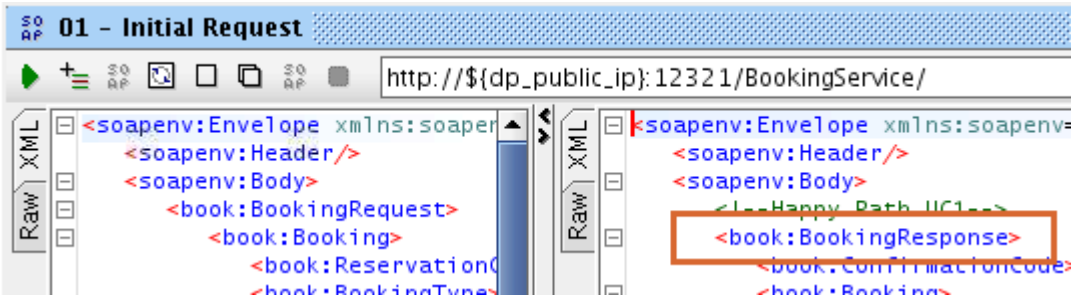


__ 3. Double-click **01 – Initial Request** to open the request window.

__ 4. In the URL address field, verify that it still contains **http://${dp_public_ip}:*${mpgw_booking_port}*/BookingService**

__ 5. Click the green **Submit** arrow to POST the request to BookingServiceProxy.

__ 6. Because no errors exist in the message that is submitted to DataPower, you see the expected <book:BookingResponse> XML tree in the response tab.



__ 7. Close the **01 – Initial Request** window.

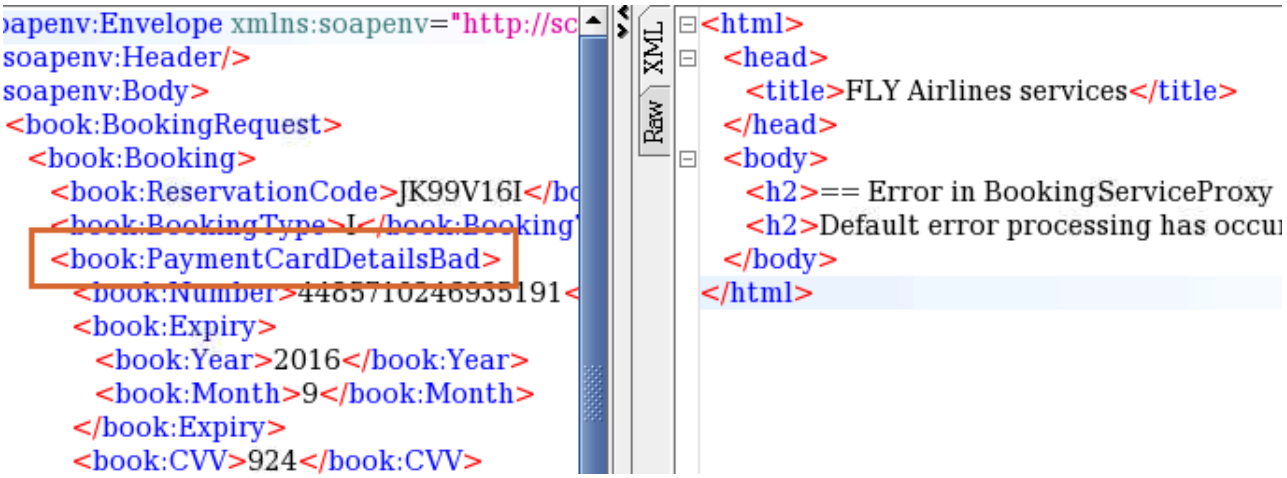
__ 8. Double-click **08 – Request Error** to open the request window.

__ 9. The URL address field should be: **http://${dp_public_ip}:*${mpgw_booking_port}*/BookingService**

__ 10. Look at the new input message on the request tab. Notice the incorrect element **<book:PaymentCardDetailsBad>**. This element fails the schema validation.

__ 11. Click the green **Submit** arrow to send the error request.

__ 12. The request error sends up a SOAP message with an invalid element <PaymentCardDetails**Bad**> so an error is detected and the `default_error.html` is returned. The HTML is in the response tab.



In later sections, you see how you can retrieve transaction-specific information in a style sheet and present it to a client.

Important

Notice the value of the second header (<h2>) which states the “Default error processing has occurred”. The HTML that was in the error policy for the MPGW generated this text. This distinction in error processing is important. As you configure error rules and an On Error action, you use style sheets to report the error message. Because the Error Policy is the default error processing, use of the On Error action and error rule override the Error Policy. Therefore, you can confirm other error processing that takes precedence by a different error message.

Section 3: Create the error rule and add it to the service policy

__ 1. Examine the `custom-error.xsl` file that is used to transform error messages.

__ a. In the Ubuntu File Browser, go to: `<lab_files>/errors`. The File Browser icon is in the launcher on the left side of the desktop.

__ b. Right-click and open the `custom-error.xsl` style sheet in an editor to view it. This file is used to return a custom error message. It returns an HTML response that includes a traceable transaction number, error code, error subcode, and the error message for correlation during problem discovery.

Note

Note the following style sheet code:

```
Transaction ID:
<xsl:value-of      select="dp:variable('var://service/transaction-id')" />
```

This code uses the DataPower extension function `dp:variable` to retrieve the service variable `var://service/transaction-id`, and append it to the HTML text `Transaction ID:`

You find similar code for other service variables.

__ c. Close the editor and the File Browser.

__ 2. Edit the **BookingServiceProxy** service to add the error rule to the service policy.

__ a. Edit the BookingServicePolicy multi-protocol gateway policy by clicking the pencil (edit) icon. The current configuration of the policy is displayed.

__ b. Click **New Rule** to create a rule.

__ c. Enter `BookingServicePolicy_ErrorRule` in the **Rule Name** field. Usually, you use the default rule name, but in this case, you want a specific name.

__ d. From the **Rule Direction** list, select **Error**.

__ e. Double-click the **Match** icon.

__ f. In the “Configure a Match action” window, select the Matching Rule that you created in an earlier step: `GenericErrorcode`

__ g. Click **Done**.

__ h. In the policy editor, drag the **Transform** icon onto the rule configuration path after the matching rule.

This action is used to transform the error message that the DataPower device generates into an HTML file that includes some of the service variable values.

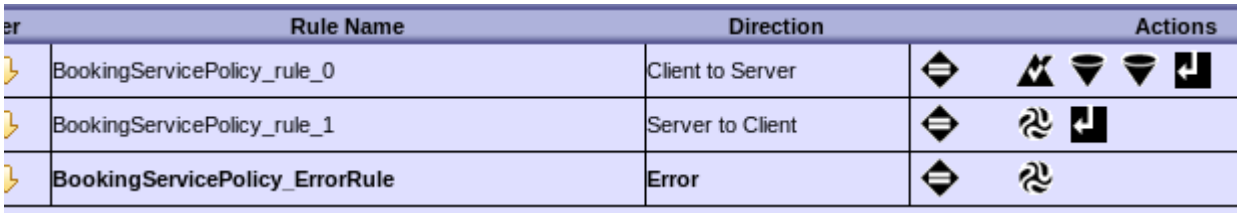
__ i. Double-click the **Transform** action.

__ j. Select **Transform with XSLT style sheet**.

__ k. Click **Upload** > **Browse**, and go to the file: `<lab_files>/errors/custom-error.xsl`

__ l. Click **Open** > **Upload** > **Continue**, and then click **Done**.

__ m. In the policy editor, click **Apply Policy**. You now have three rules that are listed in the Configured Rules section at the bottom of the window.



__ n. Click **Close Window**.

__ o. Click **Apply** to save the changes to the multi-protocol gateway.

Section 4: Test the error rule

__ 1. In the SoapUI project tree, expand the BookingServices project until **01 – Initial Request** is visible.

__ 2. Double-click **01 – Initial Request** to open the request window.

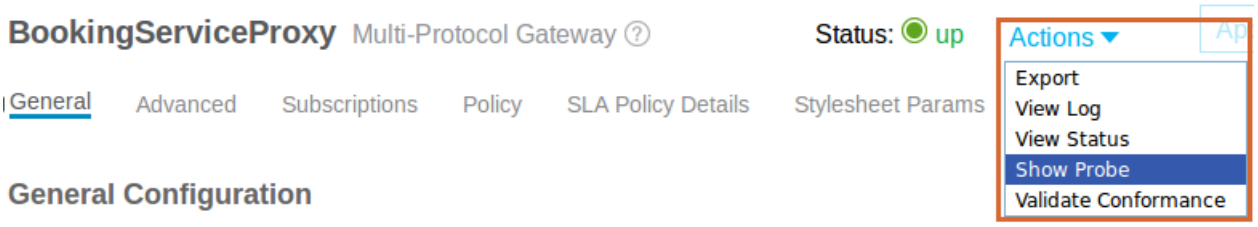
__ 3. Verify that the URL address field is:
http://${dp_public_ip}:*${mpgw_booking_port}*/BookingService

__ 4. Click the green **Submit** arrow to POST the request to BookingServiceProxy.

__ 5. The response should be a <book:BookingResponse> normal response.

__ 6. You now examine a new transaction by using the DataPower probe. Start the multi-step probe for BookingServiceProxy.

__ a. Click **Actions > Show Probe** at the top of the Multi-Protocol Gateway configuration page.



__ b. Click **Enable Probe** at the top of the transaction list page. Then, click **Close** in the Action Completed Successfully window.

__ 7. In SoapUI, double-click **08 – Request Error** to open the request window.

__ 8. If any response tab contents are visible, clear them.

__ 9. Verify that the URL address field is:
http://${dp_public_ip}:*${mpgw_booking_port}*/BookingService

__ 10. Click the green **Submit** arrow.

__ 11. You receive an HTML result that contains the service variables that are generated from `custom-error.xsl`. The error message text in the response is the error message that is generated internally. The error message specifies the reason for the failure, a faulty element.

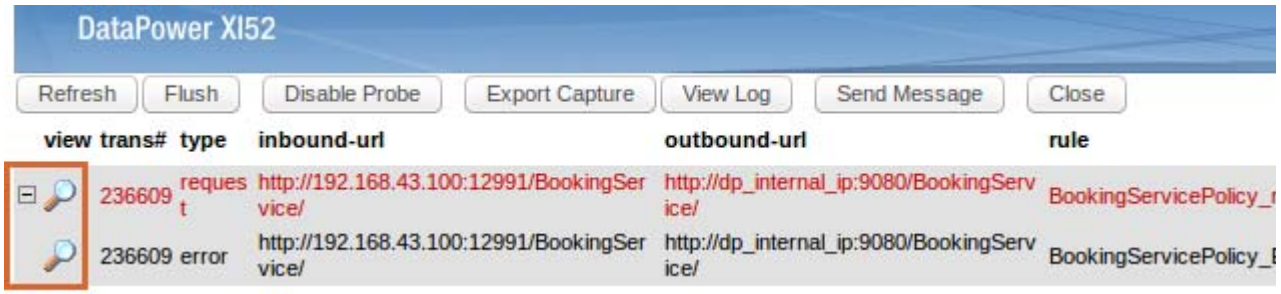
Notice how this error message is different than the error message you received before. The difference is because the new Error Rule taking precedence over the default Error Policy.

This result also shows the dynamic retrieval of transaction-specific values such as transaction ID.

```
ntent="text/html; charset=UTF-8" http-equiv="Content-Type"><title>My Company Application<
     Please print this page and forward to Application Support.
</p><p>Thank you.</p><p><font size="2">
          Transaction ID:
          236609</font></p><p><font size="2">
          Error code:
          0x00230001</font></p><p><font size="2">
          Error-subcode:
          0x01d30003</font></p><p><font size="2">
          Error message:
          http://192.168.43.100:12991/BookingService/: cvc-particle 2.1: in element {http
```

__ 12. Now you examine this transaction in the probe. In the probe's Transaction List window, click **Refresh**.
Your transaction has a plus sign (+) in front of the magnifying glass icon.

__ 13. Expand the entry.



Information

The red color of the `Request` message processing identifies an error in the request rule. When the processing rule experienced an error, control was passed to the error rule, `BookingServiceProxy_ErrorRule`. The error rule that is processed successfully is represented in the black color text.

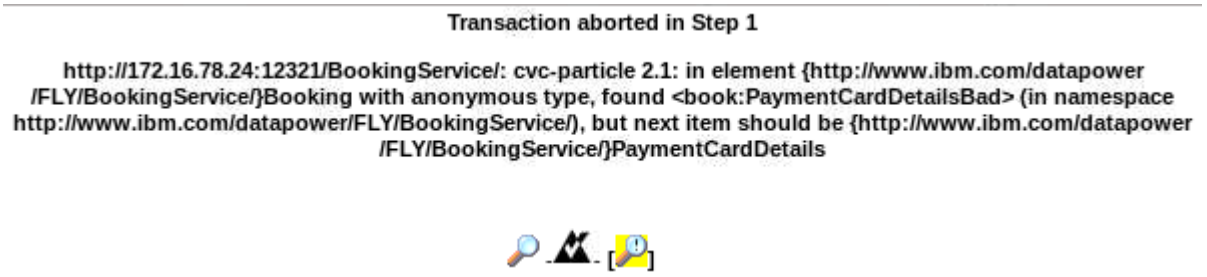
__ 14. Click the magnifying glass for the *request*.

__ 15. In the probe transaction window, the INPUT context is listed in the bottom half. Notice the <book:PaymentCardsDetailsBad> element, which is not part of the schema.

__ 16. At the top of the window, the selected request rule is shown.

__ 17. You can see the Validate action, but what happened to the rest of the actions in the rule? First, notice that the focus of this window is right after the Match action is processed, and just before the first processing action, Validate, is about to be executed. You can tell that this condition is the case because of the square brackets around the magnifying glass on the left (execution goes left to right, similar to the policy editor).

__ 18. Now click the magnifying glass after the Validate is processed.



__ 19. The probe indicates that the Validate faction failed, and displays the error message. Since the request rule terminated processing, the other actions do not display because they never executed.

Important

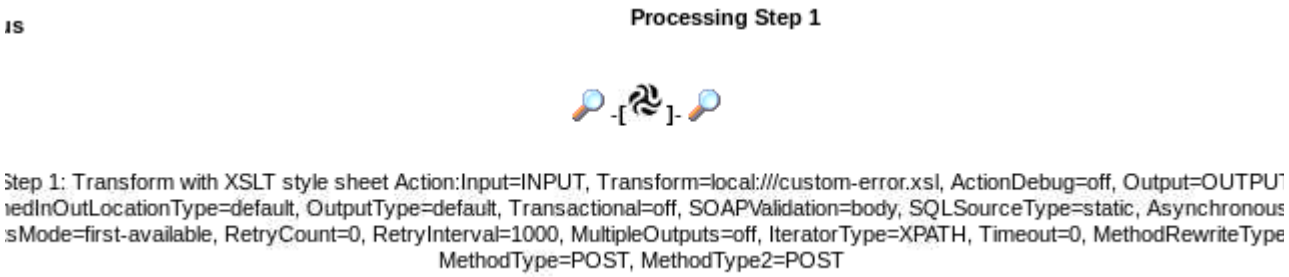
The probe shows only what happened, not what can possibly occur.

__ 20. Close the request rule transaction window.

__ 21. In the transaction list window, click the magnifying glass for the **error**.

__ 22. Recall that the error rule has a Transform action only. The initial focus is just before the Transform action is processed. Notice that the INPUT context is the original message with the invalid element.

__ 23. Click the Transform action icon. The details of the action are listed.



__ 24. Notice that the Transform action refers to the error style sheet: “Transform=local:///custom-error.xsl”

__ 25. Now click the magnifying glass after the Transform action.

__ 26. The content area shows the OUTPUT context. It contains the message that is returned to the client, which is the HTML produced by the style sheet. Notice that the response contains the transaction ID and other specific information. The style sheet retrieved this information when it executed.

__ 27. Make a note of the transaction ID in the HTML.

__ 28. Click the **Service Variables** tab.

__ 29. This tab is a list of some of the DataPower variables that relate to this execution. Scroll down towards the bottom to find your transaction ID. You see the variable that contains it, “var://service/transaction-id”. That name is the same variable that was used in custom-error.xsl to retrieve the ID. Using the probe, you can examine many of the DataPower variables without having to write style sheets to expose them.

Information

Another variable might have the same number as your transaction ID. That variable is for the global transaction ID, which helps with debugging chained services. Because this execution is the first, and only, service in this execution, the transaction ID and global transaction ID are the same value.

__ 30. Close the transaction window.

__ 31. In the transaction list window, click **View Log**.

__ 32. A log window opens. You can search on the transaction ID that you recorded to see the log entries.

__ 33. Close the log window.

__ 34. Keep the transaction list window open.

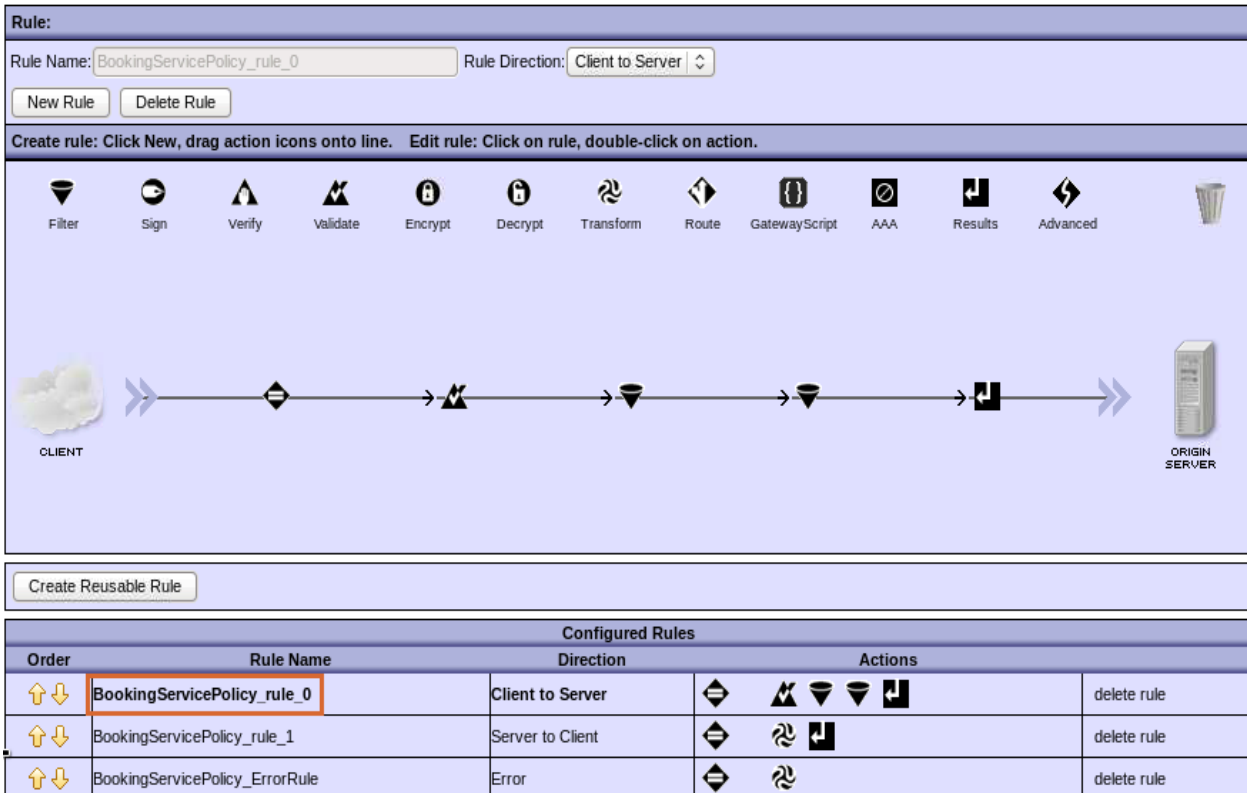
Section 5: Add an On Error action to the policy

In this section, you add an **On Error** action to the request rule. This action sets the error handling for any **subsequent** actions in the rule.

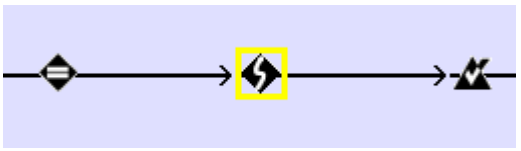
This particular **On Error** action specifies the already-existing error rule as the error handler. You see some differences in the error message that is generated and the probe details.

__ 1. Go back to the BookingServiceProxy policy editor.

__ 2. Be sure that the request rule is displayed on the rule configuration path. If not, select the request rule in the Configured Rules section.



__ 3. Drag the **Advanced** icon to the configuration path before the **Validate** action.



__ 4. Double-click the **Advanced** icon to open the Configure Action page.

__ 5. Scroll down to select **On Error**, and click **Next**.

Note

In firmware V7.5.0.0, you might not be able to scroll the list of advanced actions by using the scroll bar. If so, then click in the list and use the up/down arrow keys to scroll through the list.

__ 6. Set the **Error Mode** to **Cancel** because you want the rule to terminate after the error handling.

__ 7. For the Processing Rule, use the list to select **BookingServicePolicy_ErrorRule**.

__ 8. Leave the **Error Input** field empty. The default behavior is to have the failed action context become the input context for the Processing Rule.

__ 9. Leave the **Error output** field empty. The **Transform** action in the error rule sets the OUPUT context.

__ 10. Click **Done**.

__ 11. Click **Apply Policy**, and then **Close Window** on the policy editor.

__ 12. Click **Apply** on the Multi-Protocol Gateway configuration page.

Section 6: Test the On Error action

__ 1. In SoapUI, verify that the **08 – Request Error** request window is still open. Clear the **Response** tab so that you can see the new response.

__ 2. Click the green **Submit** arrow and submit the error request.

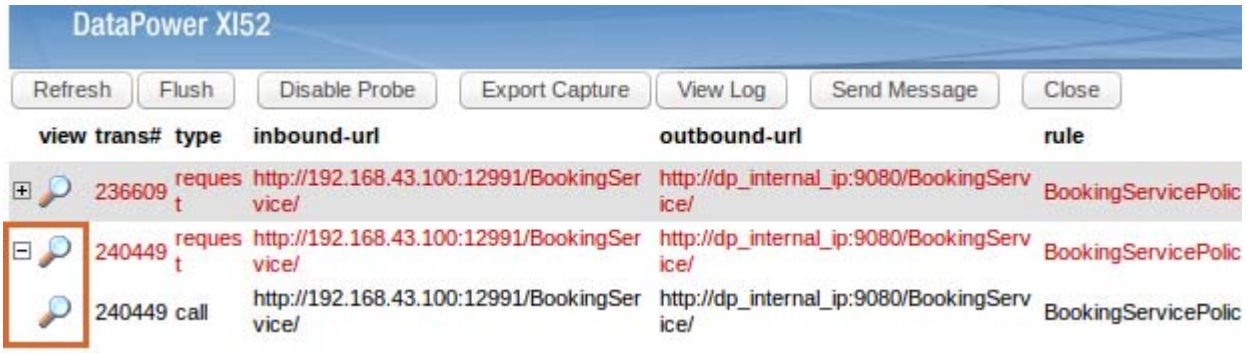
__ 3. You receive an HTML result that contains the service variables that are generated from `custom-error.xsl`, as before.

__ 4. From the **Multi-step Probe Transaction** list, select **Refresh**.

Note

Depending on conditions, saving a service might disable the probe. If that happens, click **Enable Probe** and run the test again.

__ 5. Expand your transaction as before. You see **request** and **call** probe entries.



__ 6. Click the magnifying glass for the **request**. The transaction ends at the validation as before, but in reality the **On Error** action takes control, and calls the error rule.

__ 7. On the Transaction List window, notice that the probe entry is of type **call**. Click the magnifying glass for the **call**. This probe is for running the error rule.

The **Transform** action is still using `custom-error.xsl`.

The output context displays the HTML you received on the SoapUI response tab, which contains the error message for the validation error.

Information

When an error occurred on the `BookingServiceRequest` rule, the processing implemented the configuration of the On Error action. Notice that the processing type of **call** is shown on the multi-step probe for the processing action. Before, when using the Error Rule, the processing type of **`error`** was used.

__ 8. Close the Probe Details window.

__ 9. From the **Multi-step Probe Transaction** list, select **Disable Probe** to stop the probe recording for this service.

Optional

You can open the system log and see the On Error action and error rule processing. You should see:

- Both an **information** and **debug** level message that details the setting of the On Error action
- The Validate action execution, and its failure
- The BookingServicePolicy_ErrorRule executing

Section 7: Add another Error rule and On Error action

__ 1. Open the policy editor again, and click **New Rule**.

__ 2. Name the new **error** rule: `BookingServicePolicy_filter_ErrorRule`

__ 3. Create it the same way as the previous error rule, but in the **Transform** action, upload `filter-custom-error.xsl` instead. For more information, see the previous section Create the error rule and add it to the service policy.

__ 4. Click **Apply Policy** to commit the new error rule.

Configured Rules		
Rule Name	**Direction**	**Actions**
BookingServicePolicy_rule_0	**Client to Server**	
BookingServicePolicy_rule_1	Server to Client	
BookingServicePolicy_ErrorRule	Error	
BookingServicePolicy_filter_ErrorRule	Error	

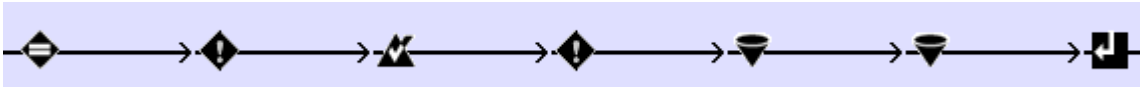
__ 5. Select the request rule with the Filter actions (probably named `BookingServicePolicy_rule_0)` to move it to the rule configuration path.

__ 6. Using the **Advanced** icon, add an **On Error** action just before the first **Filter** action.

__ 7. On the Configure On Error Action page, set the **Error Mode** to **Cancel** and leave the **Error Input** and **Error output** fields at **(none)**. For the Processing Rule, select the newly created **BookingServicePolicy_filter_ErrorRule**.

__ 8. Click **Done**.

__ 9. Confirm that the rule looks like the image:



__ 10. Click **Apply Policy** and close the policy editor.

__ 11. Click **Apply** to save the MPGW.

Section 8: Send a message to test the new error-handling

__ 1. For this test, you send a message that has a valid schema, but has an element with SQL injection. Double-click **05 - SQL Injection** to open the request window.

__ 2. Verify that it still has the correct endpoint URL: **http://${dp_public_ip}:<*mpgw_booking_port*>/BookingService**

__ 3. Click the green **Submit** arrow to send the error request.

__ 4. On the response tab, notice that the HTML contains the text “Illegal operation attempted” and “This error will be reported to Application Security”. You must scroll to the right in the tab to see the “illegal operation” text. This response is different from the response for an invalid schema.

Information

The second **On Error** action *overrides the previous one*. Therefore, this new **On Error** action directed the failing message to the second error rule, which produced a different error message for the client.

__ 5. Click **Save changes** in the message area to commit the MPGW to the domain configuration.

__ 6. Close any SoapUI request windows.

End of exercise

Exercise review and wrap-up

In this exercise, you examined the two ways to manage errors that occur while a policy is running: error rules, and **On Error** actions.

You also saw the effect of adding the **On Error** action to call a different error rule within the same request rule.

Because this service is an MPGW, you also created an error policy at the service level.

Exercise 5. Creating cryptographic objects and configuring SSL

Estimated time

01:00

Overview

This exercise shows you how to create cryptographic objects in DataPower, and how to use them to configure SSL connections. You create the cryptographic objects that you need to support an SSL connection: crypto key, crypto certificate, crypto identification credentials, and crypto validation credentials. These objects are used as part of an SSL client profile, SSL server profile, or SNI SSL server profile that defines one end of an SSL connection. You create and modify multi-protocol gateways (MPGWs) to use an SSL connection between them.

Objectives

After completing this exercise, you should be able to:

- Generate crypto keys by using the DataPower cryptographic tools
- Create a crypto identification credential by using a crypto key object and a crypto certificate object
- Validate certificates by using a validation credential object
- Create an SSL client profile that initiates an SSL connection request from a DataPower service
- Create an SSL server profile that accepts an SSL connection request from a client
- Create an SNI SSL server profile that accepts an SSL connection request with an SNI extension from a client

Introduction

The end goal of this exercise is to provide an SSL connection between the existing BookingServiceProxy MPGW and a new BookingServiceSSLProxy MPGW.

To define the SSL endpoints in DataPower, several objects need to be configured. In the first half of the exercise, you create the crypto objects that you need for the SSL connection:

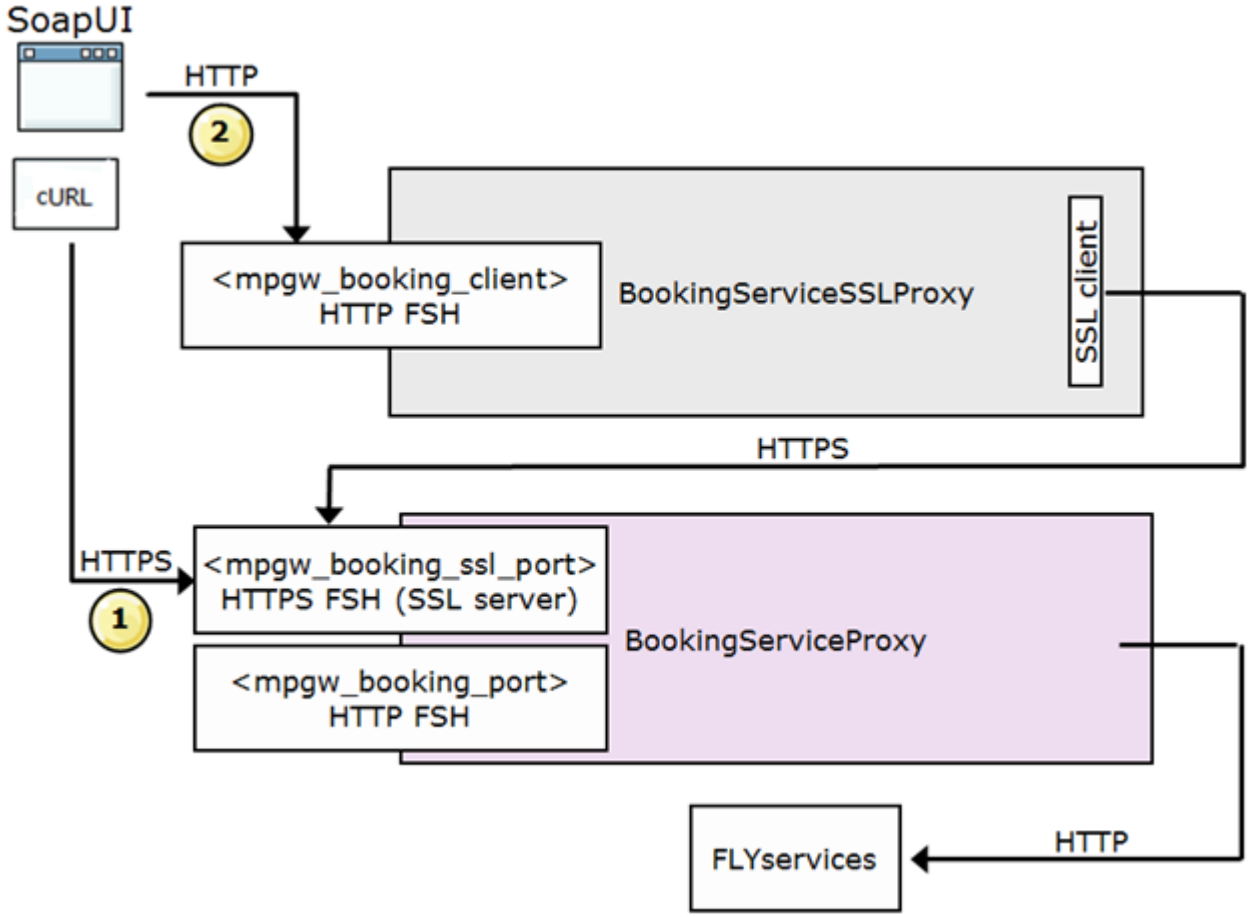
- Generate the asymmetric keys and certificates, and create the related crypto objects
- Create identification credential objects and validation credential objects
- Create the SSL profiles that specify the characteristics of the SSL endpoints

In the second half of the exercise, you modify the BookingServiceProxy MPGW to support an HTTPS front end, which acts as an SSL server. You test that HTTPS handler by sending an HTTPS

request from cURL (Step 1 in the figure). Next, you create a BookingServiceSSLProxy, which acts as an SSL client when calling the HTTPS front side handler of the BookingServiceProxy MPGW. You send an HTTP request from SoapUI to the BookingServiceSSLProxy to test the SSL connection (See Step 2).

You do the following activities:

- Create an HTTPS front side handler, with an associated SSL SNI Server Profile, for the BookingServiceProxy MPGW
- Verify the correct behavior of this HTTPS handler
- Create a BookingServiceSSLProxy MPGW that calls the BookingServiceProxy over SSL
- Test the SSL connection by using SoapUI



Requirements

To complete this exercise, you need:

- Access to the DataPower gateway
- Completion of the previous exercises (see the Preface section in the exercise instructions for details)
- cURL to send SSL SNI requests to the DataPower gateway
- SoapUI, to send requests to the DataPower gateway

- The BookingServiceBackend web service that runs on the DataPower gateway in the `FLYServices` domain
- Access to the `<lab_files>` directory

Exercise instructions

Preface

- This exercise depends on the previous completion of Exercise 3: Enhancing the BookingService gateway for the MPGWs used in this exercise.
- Remember to use the domain and port address that you were assigned in the exercise setup. **Do *not*** use the default domain.
- The references in exercise instructions refer to the following values:
  - <*lab_files*>: Location of the student lab files. Default location is: `/usr/labfiles/dp/`
  - <image_ip>: IP address of the student image (use `/sbin/ifconfig` from a terminal window to obtain value).
  - <*dp_internal_ip*>: IP address of the DataPower gateway development and administrative functions that are used by internal resources such as developers.
  - <*dp_public_ip*>: IP address of the public services on the gateway that is used by customer and clients.
  - <*dp_WebGUI_port*>: Port of the WebGUI. The default port is `9090`.
  - <*nn*>: Assigned student number. If no instructor is present, use “`01`”.
  - <*studentnn*>: Assigned user name and user account. If no instructor is present, use “`student01`”.
  - <*studentnn_password*>: Account password. In most cases, the initial value is the same as the user name. You are prompted to create a password on first use. Write it down.
  - <*studentnn_domain*>: Application domain that the user account is assigned to. If no instructor is present, use “`student01_domain`”.
  - <*FLY_booking_port*>: Port number that the back-end BookingServices web services listen on. The default port is 9080.
  - <*mpgw_booking_port*>: 12*nn*1, where “*nn*” is the two-digit student number. This number is the listener port for the MPGW that proxies the BookingService web service.
  - <*mpgw_booking_ssl_port*>: 12*nn2*, where “*nn*” is the two-digit student number. This port number is the listener port for the BookingServiceProxy web services over HTTPS.
  - <*mpgw_booking_client*>: 12*nn3*, where “*nn*” is the two-digit student number. This port number is the listener port for the BookingServiceSSLProxy services over HTTP.

5.1. Initialize the lab environment

Some setup activities are required to properly configure the lab environment and determine IP addresses and ports.

__ 1. If you have not yet performed the setup activities, you must go to **Appendix B: Lab environment setup**. Complete those activities before proceeding.

These activities need to be performed only once for the course.

5.2. Generate a certificate-key pair on the DataPower gateway

In this section, you create a certificate-key pair on the DataPower gateway. The certificate-key pair can be used during SSL connections. Although the message data exchange is encrypted with a symmetric key, the initial SSL handshake protocol uses the asymmetric certificate-key pair. As an SSL server, the generated certificate that contains your public key is presented to any SSL clients. An SSL client uses the certificate to encrypt the SSL handshake protocol messages, which only your private key can decrypt.

__ 1. Use a web browser to log on to the DataPower WebGUI: `https://<dp_internal_ip>:<dp_WebGUI_port>`

__ 2. Switch to your own student domain.

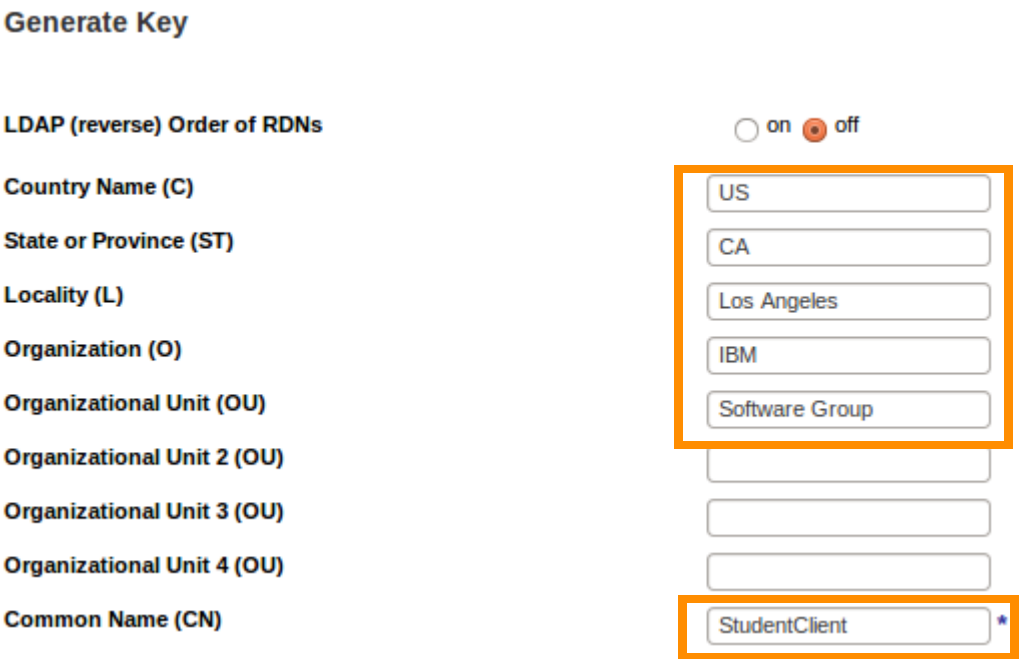
__ 3. Switch to the Blueprint Console.

__ 4. Generate a certificate-key pair on the DataPower gateway.

__ a. From the **Open** menu, start typing `crypto` in the search field. Select **Crypto Tools** from the list. (Another approach is to click **Administration > Miscellaneous > Crypto tools**.)

__ b. The **Generate Key** tab uses the information that is entered on this page to generate a certificate-key pair. The fields from **Country Name** down to **Common Name** are part of the distinguished name. Enter the following information for the distinguished name:

- **Country Name (C)**: `US`
- **State or Province (ST)**: `CA`
- **Locality (L)**: `Los Angeles`
- **Organization (O)**: `IBM`
- **Organizational Unit (OU)**: `Software Group`
- **Common Name (CN)**: `StudentClient`



__ c. The remaining fields are for certificate-key pair information. Enter the following information and leave the remaining fields at their default values:

- **Export Private Key**: **on**
- **Object Name**: `StudentClientKeyObj`

Note

If you do not select **on** for export private key or export self-signed certificate, then you cannot download the keys. They are placed in the `cert:///` directory, which does not allow downloading of the key-related files.

__ d. Click **Generate Key**.

__ e. Click **Confirm** to proceed with generating the private key and self-signed certificate. A pop-up window indicated a successful completion.

Information

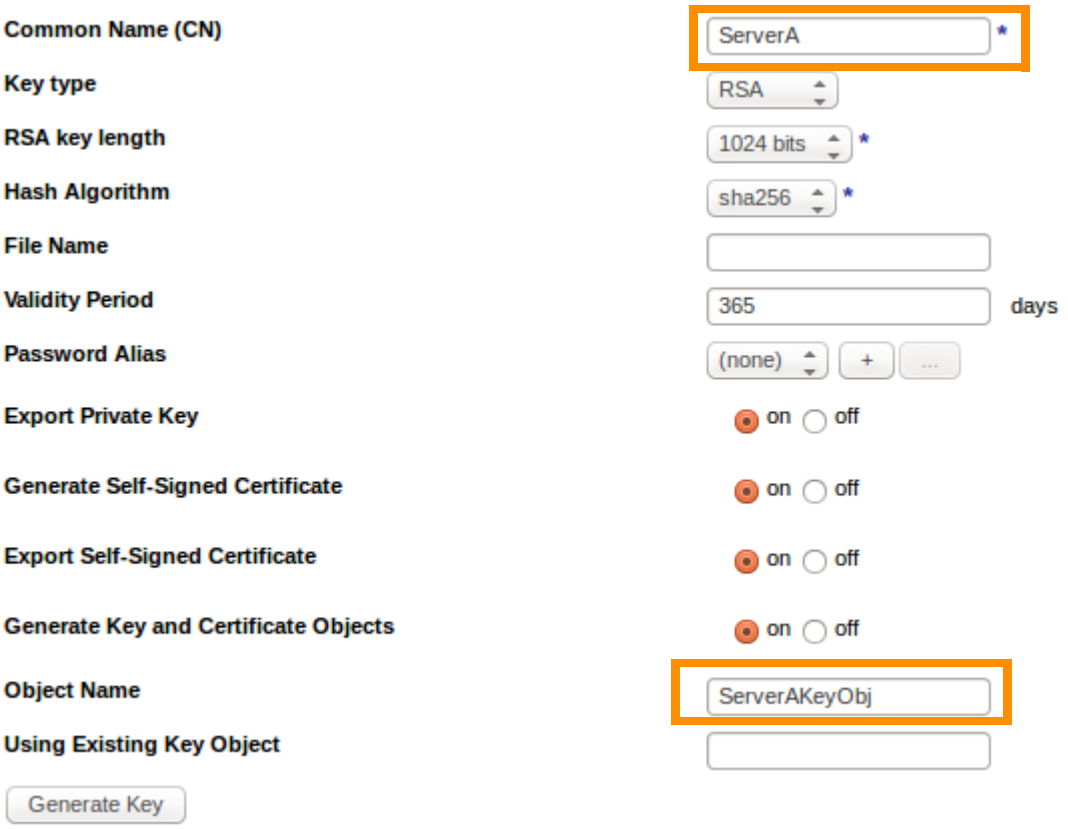
This action generates a private key and self-signed certificate and places them in the DataPower gateway `temporary:///` directory. Two objects, each with the name `StudentClientKeyObj`, are created for both the private key and self-signed certificate.

In addition to generating the private key and self-signed certificate, a *certificate signing request* (CSR) is also generated. A CSR is a request message sent to a certificate authority (CA) to create a digital certificate. A CSR consists of identifying information (common name, for example) and your public key. The request is signed with your private key, but the actual private key is not included in the request. The CA issues a signed digital certificate that replaces your self-signed certificate.

__ 5. Generate a certificate-key pair for use by an SSL server.

__ a. Change the following values in the Crypto Tools form. All other entries remain the same.

- **Common Name:** `ServerA`
- **Object Name**: `ServerAKeyObj`



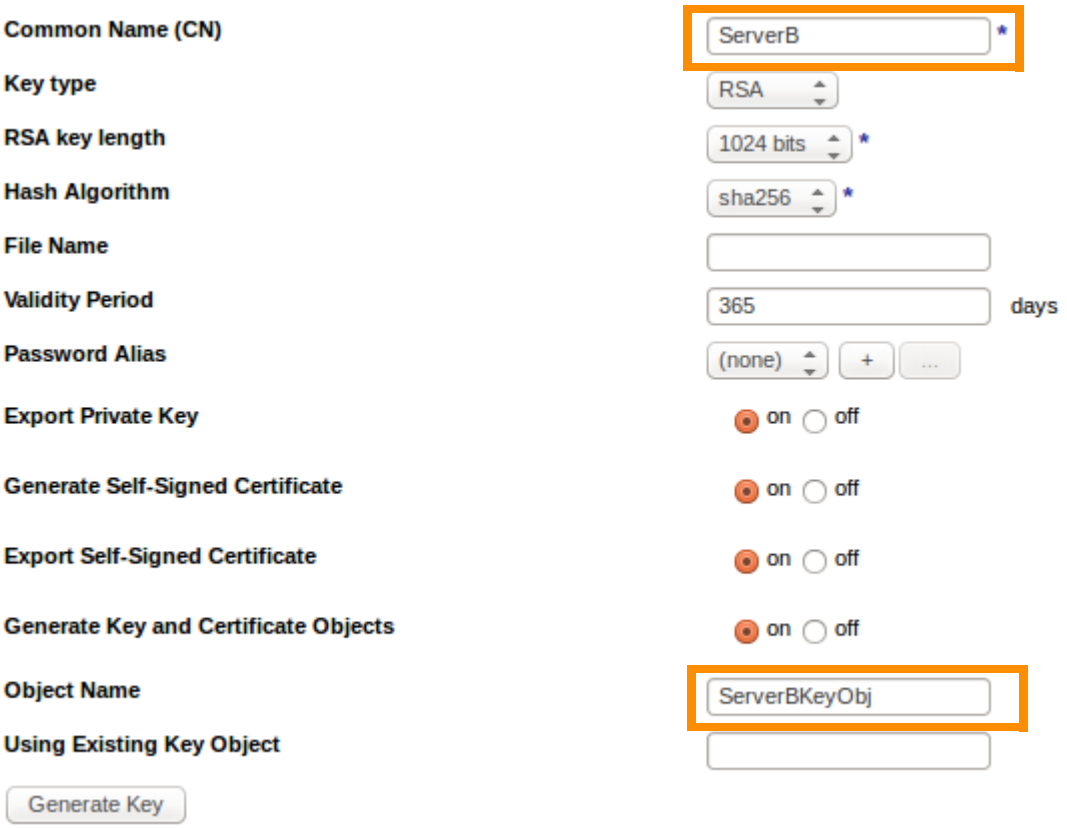
__ b. Click **Generate Key**.

__ c. Click **Confirm** to proceed with generating the private key and self-signed certificate.

__ 6. Generate a certificate-key pair for use by an SSL SNI server.

__ a. Change the following values in the Crypto Tools form. All other entries remain the same.

- **Common Name:** `ServerB`
- **Object Name**: `ServerBKeyObj`



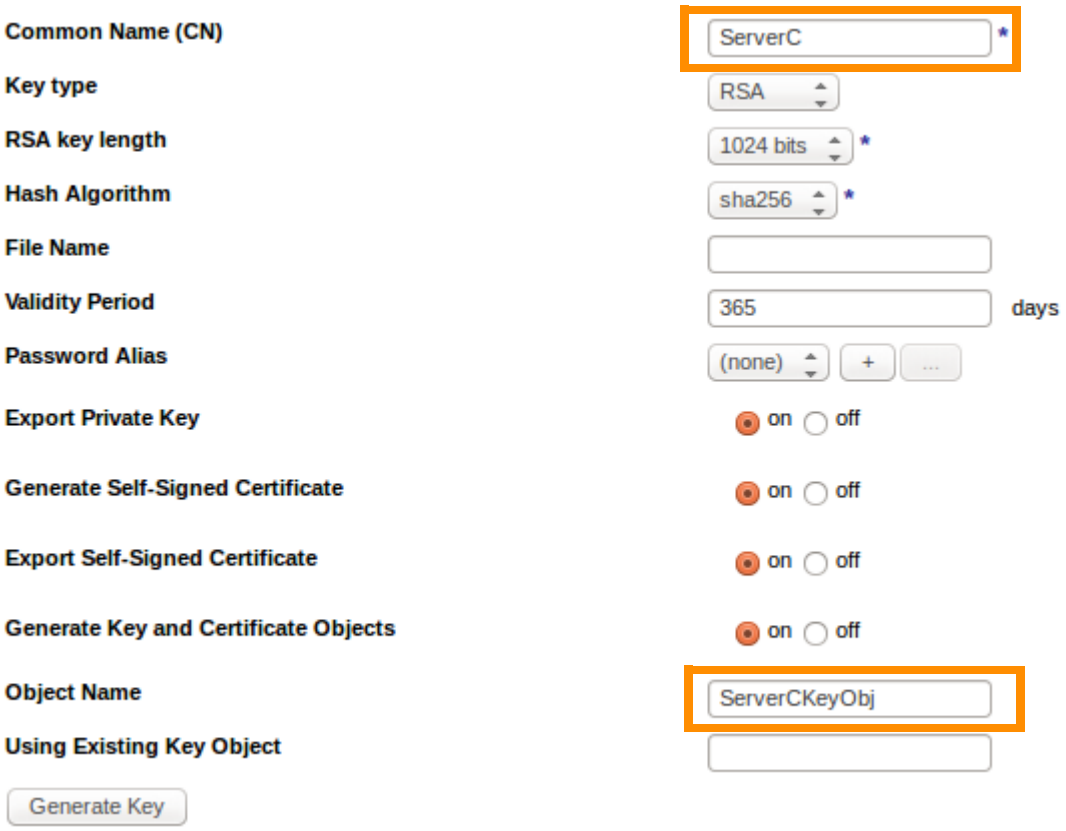
__ b. Click **Generate Key**.

__ c. Click **Confirm** to proceed with generating the private key and self-signed certificate.

__ 7. Generate a certificate-key pair for use by an SSL server.

__ a. Change the following values in the Crypto Tools form. All other entries remain the same.

- **Common Name:** `ServerC`
- **Object Name**: `ServerCKeyObj`



__ b. Click **Generate Key**.

__ c. Click **Confirm** to proceed with generating the private key and self-signed certificate.

Example

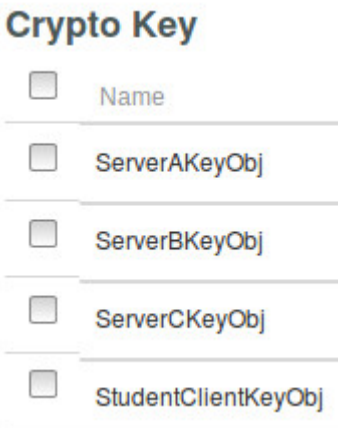
In a typical production environment, The ServerA certificate can represent the website of Company A. Similarly, the ServerB certificate can represent the website of Company B, and the ServerC certificate can represent the website of Company C.

The clients that access those sites with HTTPS expect the appropriate certificate to be returned during the SSL handshake.

__ 8. Verify the generation of the private key and certificate objects you created.

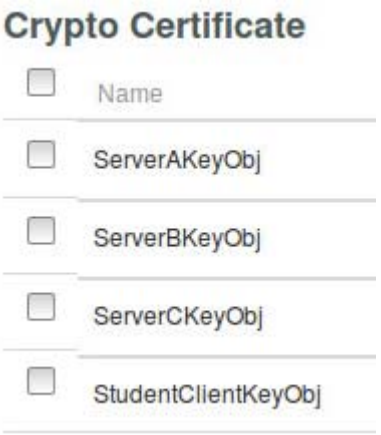
__ a. From the **Open** menu, type key in the search field. Select **Crypto Key** from the list. (Another approach is to click **Objects > Crypto Configuration > Crypto Key**.)

__ b. Verify that you see the four objects referencing the generated private key files.



__ c. Now type `cert` in the search field. Select **Crypto Certificate** from the list. (Another approach is to click **Objects > Crypto Configuration > Crypto Certificate**.)

__ d. Verify that you see the objects that reference the generated self-signed certificate files.



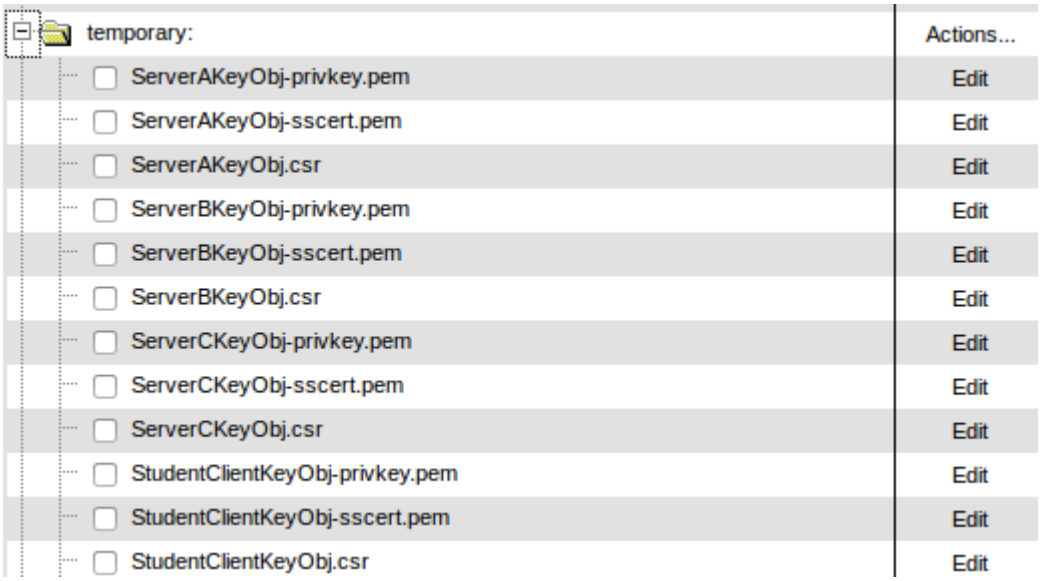
Note

The objects that DataPower generates with the same name are separate objects. They point to different files in the `cert:` directory. You can edit the key or certificate to see the specific file name.

__ 9. View the private keys and self-signed certificates that were exported to the `temporary` directory.

__ a. From the **Open** menu, click **Files**.

__ b. Expand the `temporary:` directory. You see the exported private keys, the self-signed certificates, and the CSR files:
For example, the `ServerAKeyObj-privkey.pem` is the private key file. The `ServerAKeyObj-sscert.pem` is the self-signed certificate file. The `ServerAKeyObj.csr` is the certificate signing request.



5.3. Create cryptographic objects

__ 1. Create the client identification credential object.

Information

An identification credential object defines the relationship between the private key and its associated certificate, and is used to reference a certificate-key pair during an SSL connection. You create an identification credential that references the certificate-key objects that were created in the previous steps. The identification credential object is used to identify yourself during an SSL connection, and to participate in the SSL handshake.

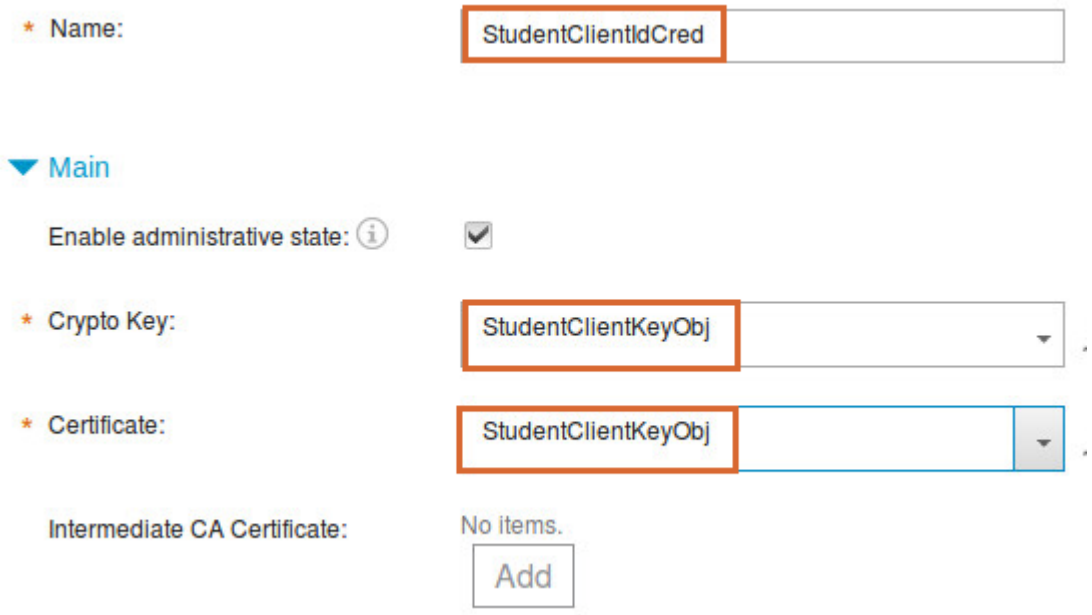
__ a. From the **Open** menu, type `cred` in the search field.

__ b. Click the **Crypto Identification Credentials** link.

__ c. On the Configure Crypto Identification Credentials list page, click **New**.

__ d. On the next page, enter the following information:

- **Name**: `StudentClientIdCred`
- **Crypto Key**: `StudentClientKeyObj`
- **Certificate**: `StudentClientKeyObj`



__ e. Click **Apply**.

This action creates an identification credential object. If a third-party CA signed the certificate, you can specify CA certificates in the **Intermediate CA Certificate** field.

2. Create the Server A identification credential object.

__ a. On the Crypto Identification Credentials list page, click **New**.

__ b. On the next page, enter the following information:

- **Name**: `ServerAIdCred`
- **Crypto Key**: `ServerAKeyObj`
- **Certificate**: `ServerAKeyObj`

__ c. Click **Apply**.

This action creates an identification credential object. As before, if a third-party CA signed the certificate, you can specify CA certificates in the **Intermediate CA Certificate** field.

3. Create the Server B identification credential object.

__ a. On the Crypto Identification Credentials list page, click **New**.

__ b. On the next page, enter the following information:

- **Name**: `ServerBIdCred`
- **Crypto Key**: `ServerBKeyObj`
- **Certificate**: `ServerBKeyObj`

__ c. Click **Apply**.

4. Create the Server C identification credential object.

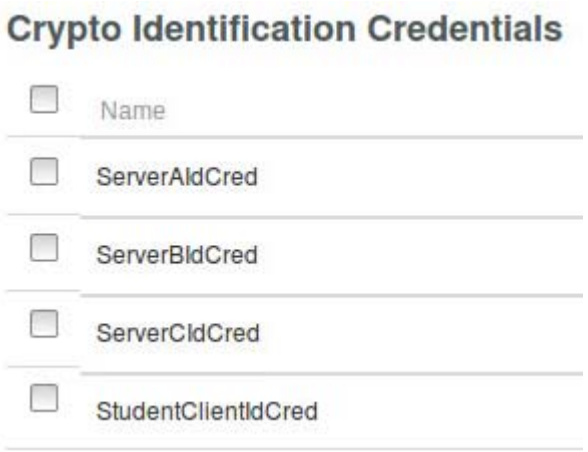
__ a. On the Configure Crypto Identification Credentials list page, click **New**.

__ b. On the next page, enter the following information:

- **Name**: `ServerCIdCred`
- **Crypto Key**: `ServerCKeyObj`
- **Certificate**: `ServerCKeyObj`

__ c. Click **Apply**.

5. The Crypto Identification Credentials list page should show the four credentials:



6. Create a Validation Credential object to verify the servers

__ a. From the **Open** menu, type `valida` in the search field. Select **Crypto Validation Credentials** from the list.

Information

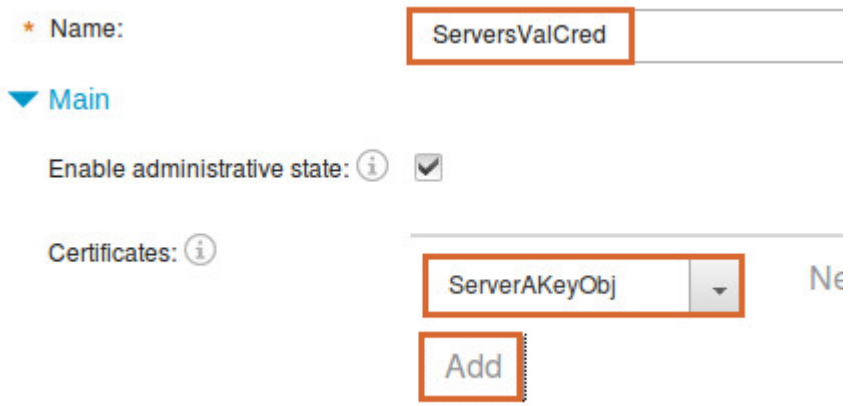
A ***validation credential object*** is used to validate the authenticity of certificates and digital signatures that are presented to an SSL endpoint.

__ b. On the Crypto Validation Credentials list page, click **New**.

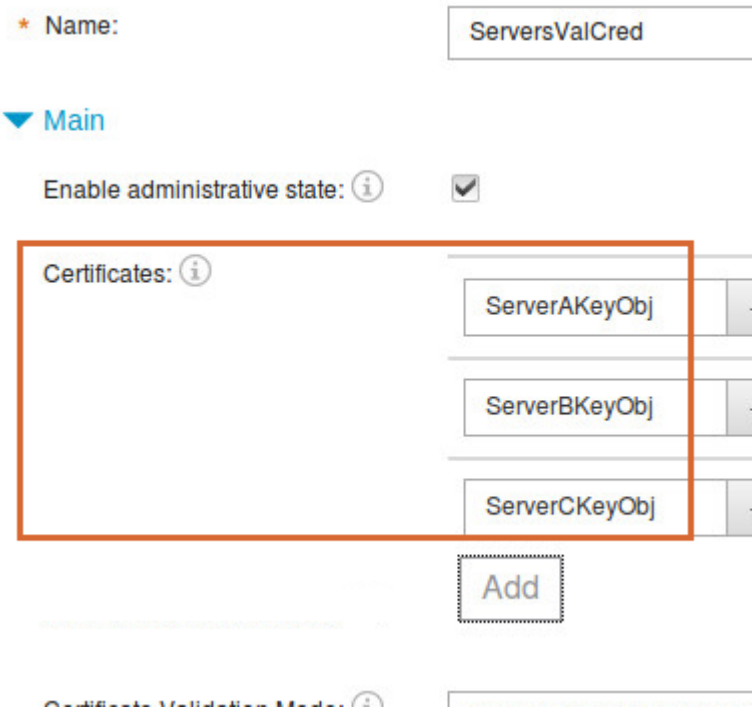
__ c. Enter the following information:

- **Name**: `ServersValCred`
- **Certificates**: `ServerAKeyObj` (select it from the list)

__ d. Click **Add** to add the `ServerAKeyObj` certificate.



__ e. Use the **Add** button to add `ServerBKeyObj` and `ServerCKeyObj` to the list of certificates. The list then contains the three server certificates.



__ f. Leave the remaining fields at their default values. Click **Apply** to save the Crypto Validation credential. This validation credential validates *any* of the three server certificates, if one is presented.

Example

If the client uses this validation credential, it accepts the certificates for Server A (Company A), Server B (Company B), and Server C (Company C).

---

7. Create a Validation Credential object to verify the client.
   __ a. On the Configure Crypto Validation Credentials list page, click **New**.
   __ b. Enter the following information:
      - **Name**: `ClientValCred`
      - **Certificates**: `StudentClientKeyObj` (Use **Add** to select it from the list)
   __ c. Leave the remaining fields at their default values. Click **Apply** to save the Crypto Validation credential.
8. Click **Save changes** in the banner.

5.4. Create SSL/TLS objects

Now that you created the necessary cryptographic objects to support SSL/TLS connections, you create the SSL/TLS objects that services, such as a multi-protocol gateway, can use.

1. Create an SSL Server profile for ServerA.
   __ a. From the **Open** menu, type `ssl` in the search field. Select **SSL Server Profile** from the list.
   __ b. On the SSL Server Profile list page, click **New**.
   __ c. Enter `ServerA` in the **Name** field.
   __ d. Leave the Protocols at the default settings of enabled TLS V1.0, TLS V1.1, and TLS V1.2.
   __ e. Leave the **Ciphers** list at its default.
   __ f. Select `ServerAIdCred` from the **Identification credentials** list.
   __ g. Click **Apply**.

Example

In the example of a typical production environment, the ServerA SSL server profile represents the website for Company A. Its identification credential refers to the ServerA certificate that is sent on an SSL handshake.

---

2. Create an SSL Server profile for ServerB.
   __ a. On the SSL Server Profile list page, click **New**.
   __ b. Enter `ServerB` in the **Name** field.
   __ c. Clear the **Enable TLS version 1.0** check box under **Protocols**.
   __ d. Leave the **Ciphers** list at its default.
   __ e. Select `ServerBIdCred` from the **Identification credentials** list.
   __ f. Click **Apply**.
3. Create an SSL Server profile for ServerC.
   __ a. On the Configure SSL Server Profile list page, click **New**.
   __ b. Enter `ServerC` in the **Name** field.
   __ c. Clear the **Enable TLS version 1.0** check box under **Protocols**.
   __ d. Clear the **Enable TLS version 1.1** check box under **Protocols**.
   __ e. Leave the **Ciphers** list at its default.
   __ f. Select `ServerCIdCred` from the **Identification credentials** list.
   __ g. Set **Request client authentication** to **enabled**. The page repaints.

__ h. Leave **Require client authentication** and **Validate client certificate** set to **enabled**.

__ i. Set **Send client authentication CA list** to **disabled**.

__ j. Select `ClientValCred` from the **Validation credentials** list.

__ k. Click **Apply**.

Information

ServerA supports TLS V1.0, V1.1, and V1.2, and does not request client authentication.

ServerB supports TLS V1.1 and V1.2, and does not request client authentication.

ServerC supports TLS V1.2 only, and requires client authentication.

---

4. Click **Save changes** in the banner.
5. Create an SSL Host Name Mapping object.

__ a. From the **Open** menu, type `ssl` in the search field. Select **SSL Host Name Mapping** from the list.

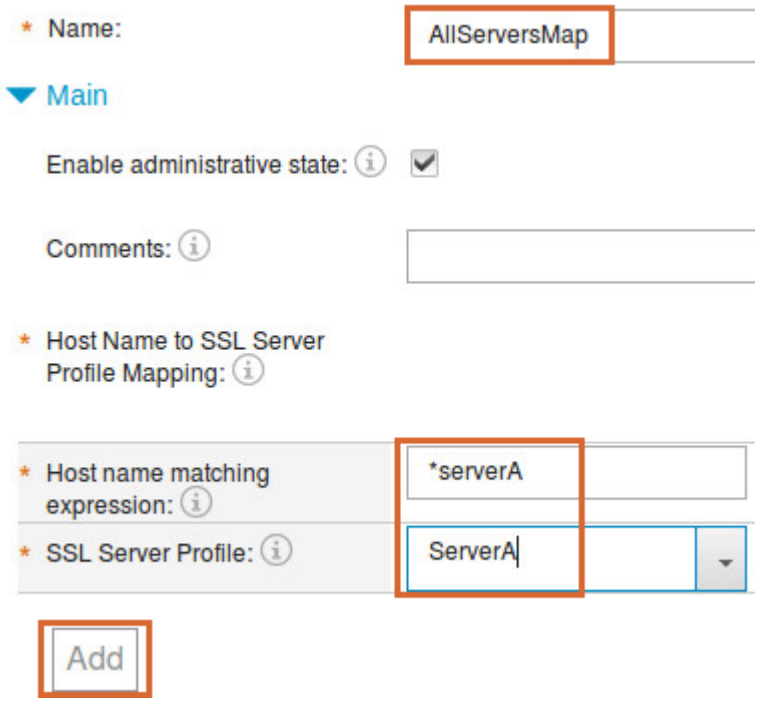
__ b. On the SSL Host Name Mapping list page, click **New**.

__ c. Enter `AllServersMap` in the **Name** field.

__ d. Under **Host Name to SSL Server Profile Mapping**, click **Add**.

__ e. Enter `*serverA` in the **Host name matching expression** field.

__ f. Select `ServerA` from the **SSL Server Profile** list.



__ g. Click **Add** to create another map.

__ h. Use the following values for the new map:

- **Host name matching expression**: `*serverB`
- **SSL Server Profile**: `ServerB`

__ i. Click **Add** to create another map.

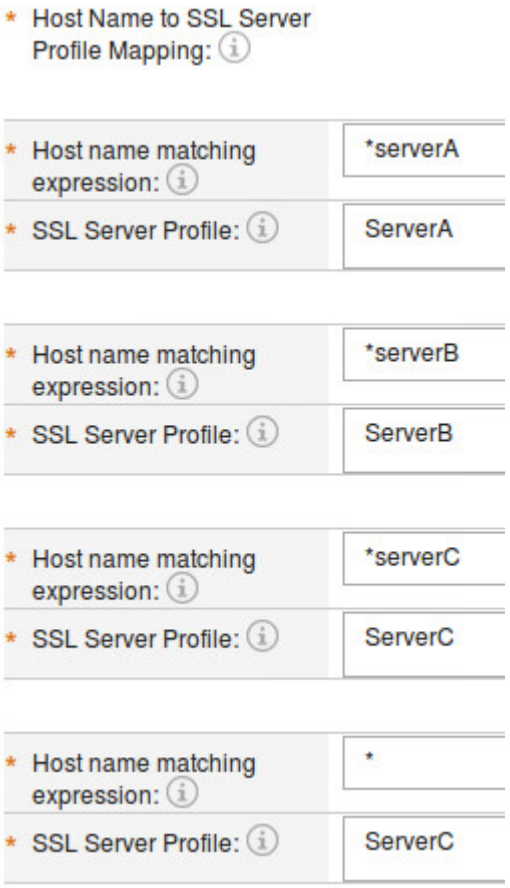
__ j. Use the following values for this new map:

- **Host name matching expression**: `*serverC`
- **SSL Server Profile**: `ServerC`

__ k. Click **Add** to create another map. This map captures all SNI server names that don't match any of the other three.

__ l. Use the following values:

- **Host name matching expression**: *
- **SSL Server Profile**: `ServerC`



__ m. Click **Apply** to complete configuration of the SSL Host Name Mapping object.

6. Click **Save changes** in the banner.

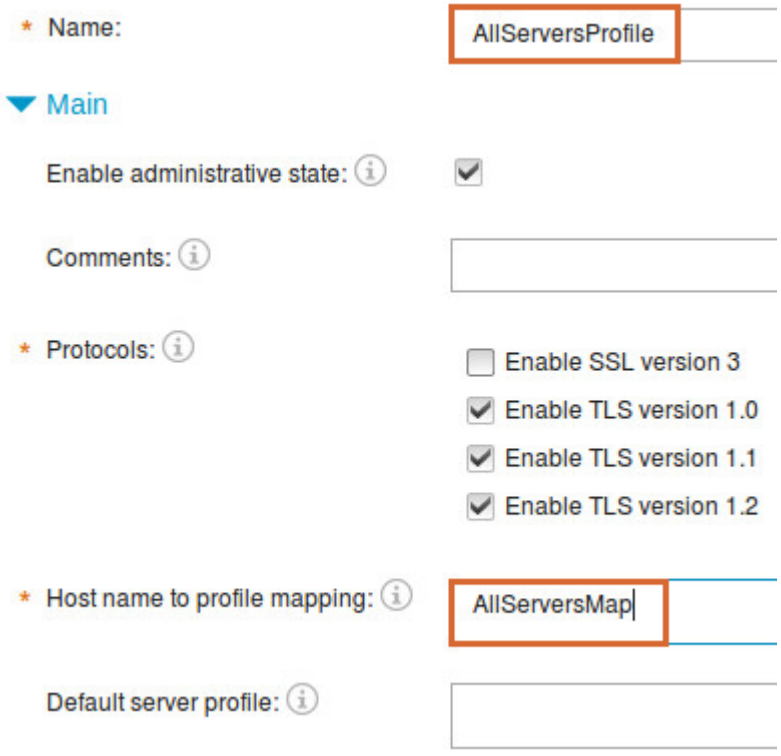
7. Create an SSL SNI Server Profile.

__ a. From the **Open** menu, start typing `sni` in the search field. Select **SSL SNI Server Profile** from the list.

__ b. On the SSL SNI Server Profile list page, click **New**.

__ c. Enter `AllServersProfile` in the **Name** field.

__ d. Select `AllServersMap` from the **Host name to profile mapping** list.



__ e. Click **Apply**.

__ f. Click **Save changes** in the banner.

__ 8. Create an SSL Client profile.

__ a. From the **Open** menu, start typing `ssl` in the search field. Select **SSL Client Profile** from the list.

__ b. On the SSL Client Profile list page, click **New**.

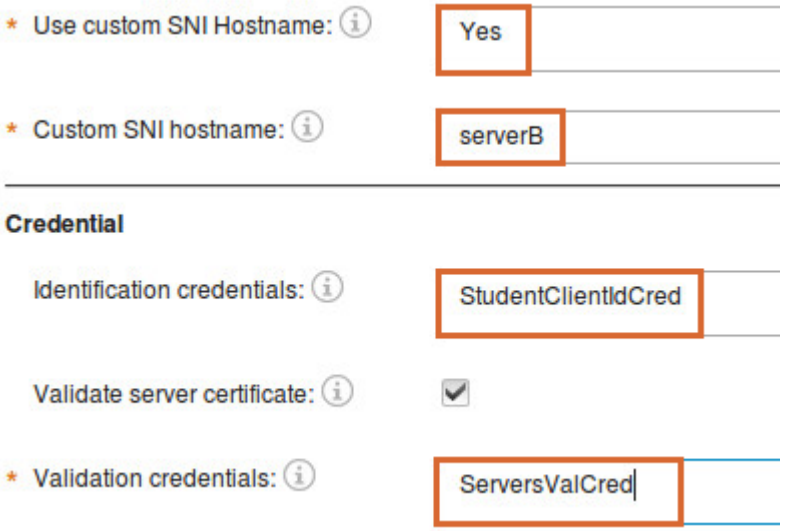
__ c. Enter `StudentClientProfile` in the **Name** field.

__ d. Select Yes from the **Use custom SNI Hostname** list.

__ e. Enter `serverB` in the **Custom SNI hostname** field.

__ f. Select `StudentClientIdCred` from the **Identification credentials** list.

__ g. Select `ServersValCred` from the **Validation credentials** list.



__ h. Click **Apply**.

__ i. Save the configuration.

5.5. Verify web service behavior

Before configuring the SSL support, verify that the web service is operating correctly. Also, attempt to use HTTPS to access the same web service.

__ 1. If required, open the SoapUI tool by clicking the **SoapUI** icon on the desktop.

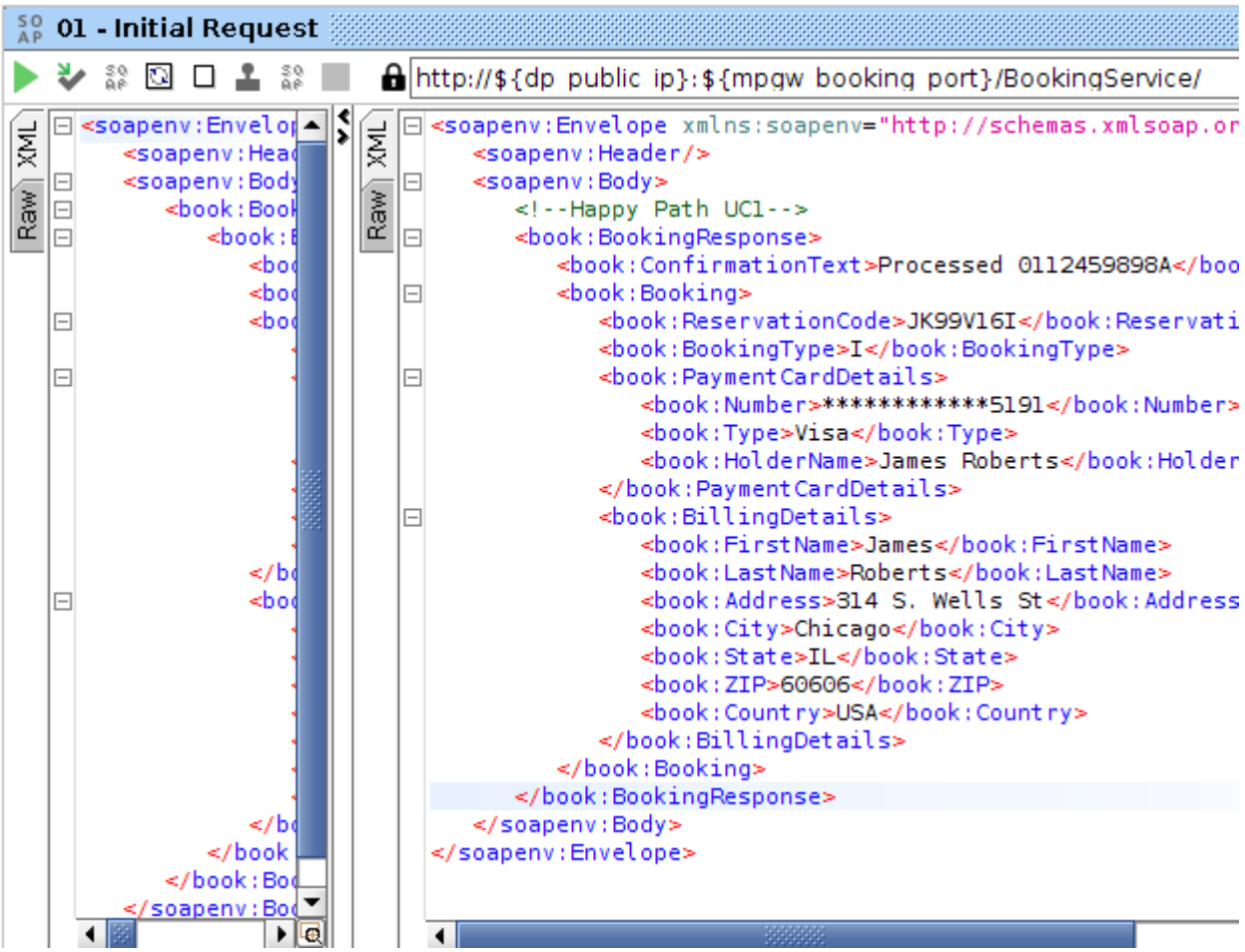
__ 2. In the project tree, expand the **BookingService** project until **01 – Initial Request** is visible.

__ 3. Double-click **01 – Initial Request** to open the request window.

__ 4. In the URL address field, verify that it still contains:
http://${dp_public_ip}:${mpgw_booking_port}/BookingService

__ 5. Click the green **Submit** arrow to POST the request to BookingServiceProxy.

__ 6. You should see the <book:BookingResponse> on the response tab.



5.6. Add an HTTPS handler to the BookingServiceProxy service

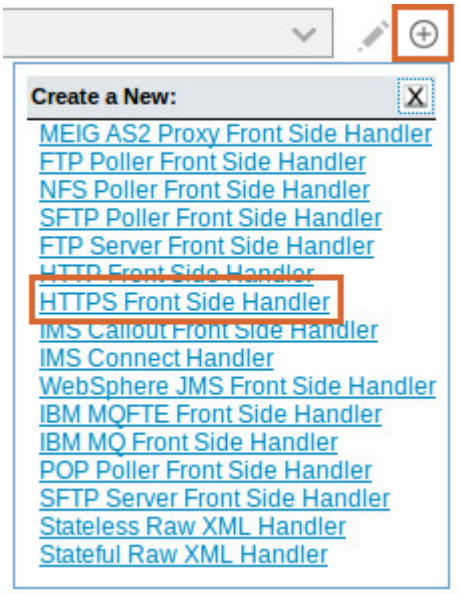
In this section, you add an HTTPS handler to the BookingServiceProxy service so that it can handle HTTPS requests. Because the BookingServiceProxy still has the HTTP handler, it can still receive HTTP requests. Because the clients are initiating the SSL request, the BookingServiceProxy service is configured as the SSL server.

__ 1. Open the **BookingServiceProxy** multi-protocol gateway configuration page.

__ 2. In the Front Side Protocol section of the page, create a handler that uses HTTPS.

__ a. Under **Front Side Protocol**, click the new icon. A list of Handler types appears.

__ b. Select **HTTPS Front Side Handler**.



__ c. The handler configuration dialog box opens. Enter: `HTTPS_12nn2`

__ d. For the Local IP address, use *<dp_public_ip>*, or its alias.

__ e. Use the Port Number: *<mpgw_booking_ssl_port>*

__ f. Near the bottom of the page, select `SNI Server Profile` from the **SSL server type** list.

__ g. Select `AllServersProfile` from the **SSL SNI server profile** list.

__ h. Click **Apply**.

__ 3. The new HTTPS handler now shows as one of the gateway front side handlers.

__ 4. Click **Apply** for the service.

__ 5. Save the configuration.

5.7. Test the HTTPS handler

In this section, you use the command line tool **cURL** to send the same booking request as with SoapUI, but you change the protocol to HTTPS, and change the port in the URL. You need to use **cURL** because it offers greater flexibility for testing against SNI-compliant servers, and the installed version of SoapUI does not send the SNI extension.

__ 1. Open a terminal window.

__ 2. Change to the `Documents` directory with the following command: `cd Documents`

__ 3. Copy the `BookingRequest.xml` file to your current directory. Execute the following command (do not miss the ending period):

```
cp /usr/labfiles/dp/BookingService/BookingRequest.xml .
```

__ 4. Issue the `ls` command to verify the `BookingRequest.xml` file copied successfully.

__ 5. Issue the `curl` command to send the request to the HTTPS Front Side Handler of the gateway. This command takes the following form. Note the double dash characters before `resolve` and `data-binary`.

```
curl --resolve serverA:<mpgw_proxy_ssl_port>:<dp_public_ip> --data-binary
@BookingRequest.xml https://serverA:<mpgw_proxy_ssl_port>/BookingService
-k -v
```

For example:

```
curl --resolve serverA:12972:172.16.78.14 --data-binary
@BookingRequest.xml https://serverA:12972/BookingService -k -v
```

```
localuser@ubuntu-base:~$ cd Documents
localuser@ubuntu-base:~/Documents$ cp /usr/labfiles/dp/BookingService/BookingReq
uest.xml .
localuser@ubuntu-base:~/Documents$ ls
BookingRequest.xml
localuser@ubuntu-base:~/Documents$ curl --resolve serverA:12972:172.16.78.14 --d
ata-binary @BookingRequest.xml https://serverA:12972/BookingService -k -v
* Added serverA:12972:172.16.78.14 to DNS cache
* Hostname was found in DNS cache
*   Trying 172.16.78.14...
```

Information

The “-k” parameter tells cURL that is does not have to validate the server certificate. The “-v” parameter sets the response are “verbose”.

Notice that the URL uses the hostname of “serverA”. The “--resolve” tells cURL to create a dynamic DNS cache, and resolve the hostname into an IP address.

Further down in the verbose response, you can see the “Host” HTTP request header that contains the “serverA” hostname from the URL.

Just before the HTTP request headers, you can see the server certificate that was presented to cURL; it is from ServerA. The SSL server code in DataPower checks the Host HTTP request parameter and attempts to resolve that hostname to the correct SSL server profile.

__ 6. Examine the response to verify that it is a booking request response.

```
< Content-Type: text/xml
< Date: Tue, 10 May 2016 17:20:54 GMT
< X-Client-IP: 172.16.80.60
< X-Global-Transaction-ID: 8544
<
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmln
s:book="http://www.ibm.com/datapower/FLY/BookingService/">
   <soapenv:Header/>
   <soapenv:Body>
      <!--Happy Path UC1--><book:BookingResponse><book:ConfirmationText>Processe
d 0112459898A</book:ConfirmationText>
         <book:Booking>
            <book:ReservationCode>JK99V16I</book:ReservationCode>
            <book:BookingType>I</book:BookingType>
            <book:PaymentCardDetails>
               <book:Number>************5191</book:Number>


               <book:Type>Visa</book:Type>
               <book:HolderName>James Roberts</book:HolderName>
            </book:PaymentCardDetails>
```

__ 7. Run another request without using the SNI support of curl. These commands should all be on one line.

```
curl --data-binary @BookingRequest.xml
https://<dp_external_ip>:<mpgw_proxy_ssl_port>/BookingService -k -v
```

For example:

```
curl --data-binary @BookingRequest.xml
https://172.16.78.14:12972/BookingService -k -v
```

The SSL server attempted to resolve the hostname of “172.16.78.14” to an SSL server profile, and could not resolve it.

```
localuser@ubuntu-base:~/Documents$ curl  --data-binary @BookingRequest.
://172.16.78.14:12972/BookingService -k -v
* Hostname was NOT found in DNS cache
*   Trying 172.16.78.14...
* Connected to 172.16.78.14 (172.16.78.14) port 12972 (#0)
* successfully set certificate verify locations:
*   CAfile: none
  CApath: /etc/ssl/certs
* SSLv3, TLS handshake, Client hello (1):
* SSLv3, TLS alert, Server hello (2):
* error:14077458:SSL routines:SSL23_GET_SERVER_HELLO:tlsv1 unrecognized
* Closing connection 0
curl: (35) error:14077458:SSL routines:SSL23_GET_SERVER_HELLO:tlsv1 unr
 name
```

__ 8. Return to the DataPower Blueprint Console and examine the configuration of the SSL SNI Server.

__ a. From the **Open** menu, start typing `sni` in the search field. Select **SSL SNI Server Profile** from the list.

__ b. On the SSL SNI Server Profile list page, click `AllServersProfile`.

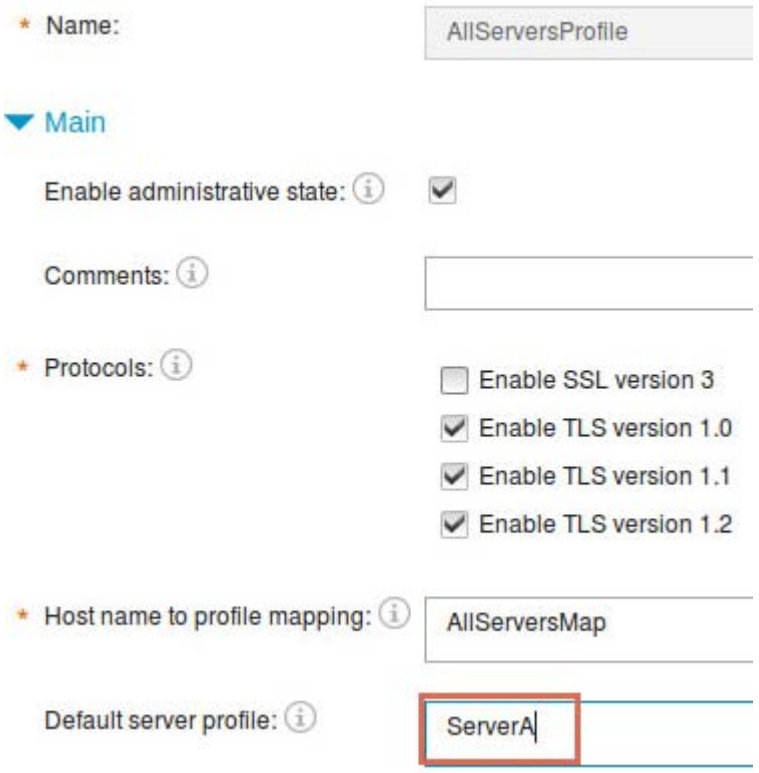
__ c. On the AllServersProfile configuration page, click **Actions > View Log**.

__ d. Notice that the error indicates a problem with parsing the TLS extension from the client.

time ▾	category	level	tid	direction	client	msgid	message	Show last 50 100
Thursday, May 12, 2016								
10:04:55 PM	ssl	error	52961		172.16.80.64	0x8120002f	ssl-sni-server (AllServersProfile): SSL library error: error:1408A0E3:SSL routines:ssl3_get_client_hello:parse tlsext	
10:04:55 PM	ssl	error	52961		172.16.80.64	0x8120002f	ssl-sni-server (AllServersProfile): SSL library error: error:1412E0E2:SSL routines:ssl_parse_clienthello_tlsext:clienthello tlsext	

__ e. Close the log window.

__ f. Set the **Default server profile** to ServerA.



__ g. Click **Apply**.

__ h. Save the configuration.

__ i. Reissue the curl command in the terminal window. It succeeds.

```
curl: (7) Failed to connect to 172.16.78.13 port 12972: Connection refused
localuser@ubuntu-base:~/Documents$ curl --data-binary @BookingRequest.xml https:
//172.16.78.14:12972/BookingService -k -v
* Hostname was NOT found in DNS cache
*   Trying 172.16.78.14...
* Connected to 172.16.78.14 (172.16.78.14) port 12972 (#0)
* successfully set certificate verify locations:
*   CAfile: none
  CApath: /etc/ssl/certs
* SSLv3, TLS handshake, Client hello (1):
```

Although the SSL client did not send an SNI extension, the SNI SSL server profile now has a default SSL server profile that it can use.

__ j. Close the terminal window.

5.8. Configure an SSL Proxy Booking MPGW

In this section, you add another MPGW to your domain. This MPGW receives an HTTP request from SoapUI, and acts as an SSL client to send the request on to the BookingService Proxy HTTPS handler. This MPGW is added strictly to provide an opportunity to configure an SSL client in a DataPower service; it acts as a simple proxy only.

__ 1. Click the **Services** icon on the Blueprint Console.

__ 2. Click **New Service > Multi-Protocol Gateway**.

__ 3. Enter `BookingServiceSSLProxy` as the Multi-Protocol Gateway name.

__ 4. Create a service policy.

__ a. Click the new icon to create a Multi-Protocol Gateway Policy.

__ b. In the policy editor, enter **BookingSSLpolicy** as the policy name and click **Apply Policy**.

__ c. Click **New Rule**.

__ d. Configure the **Match** action to use the **MatchAnyURI** matching rule.

__ e. Set the Rule Direction to **Client to Server**.

__ f. Click **New Rule** again.

__ g. Configure the **Match** action to use the **MatchAnyURI** matching rule.

__ h. Set the Rule Direction to **Server to Client**.

__ i. Click **Apply Policy**. The policy editor automatically adds a Results action to each rule.

Configured Rules		
Rule Name	**Direction**	**Actions**
BookingSSLpolicy_rule_0	Client to Server	
BookingSSLpolicy_rule_1	**Server to Client**	

__ j. Close the policy editor window.

__ 5. Create a front side handler for SoapUI to call.

__ a. Click the new icon to create a front side handler.

__ b. Select **HTTP Front Side Handler**.

__ c. Enter `HTTP2_12nn3` as the front side handler name.

__ d. Enter `dp_public_ip` as the alias for the Local IP address.

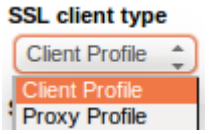
__ e. Enter a port number of <*`mpgw_booking_client`*>.

__ f. Click **Apply** to save the front side handler configuration.

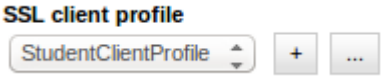
__ 6. Enter the Default Backend URI as `https://dp_public_ip:<`*`mpgw_booking_ssl_port`*`>/BookingService/`

__ 7. Now that the backend is using SSL, you need to configure the SSL Client Profile.

__ a. Select **Client Profile** from the SSL client type list.



__ b. Select the `StudentClientProfile` from the **SSL client profile** list.



__ 8. Scroll down the MPGW and set **Propagate URI** to **off**.

__ 9. Click **Apply**.

__ 10. Save the configuration.

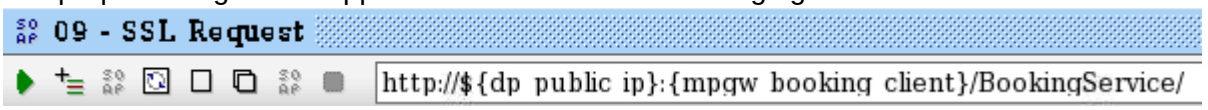
5.9. Test the SSL connection between the BookingServiceSSLProxy and the BookingServiceProxy

In this section, you test the SSL connection between the two MPGWs by invoking the BookingServiceSSLProxy over an HTTP connection. The BookingServiceSSLProxy connects to the BookingServiceProxy over an SSL connection.

__ 1. If required, Open the SoapUI tool by clicking the **SoapUI** icon on the desktop.

__ 2. In SoapUI, open **09 – SSL Request**.

__ 3. Ensure that the POST URL includes the `${mpgw_booking_client}` in the URL. A proper configuration appears as shown in the following figure:



__ 4. Click the green **Submit** arrow to POST the message to BookingServiceSSLProxy. The request should return a successful booking reservation.

Information

The SoapUI request sent an HTTP Post SOAP request to the HTTP front side handler of the BookingServiceSSLProxy. The BookingServiceSSLProxy passed the request by using the SSL Client configuration out of the back end, which called the HTTPS front side handler of the BookingServiceProxy. The communication between the BookingServiceSSLProxy MPGW and the BookingServiceProxy is SSL. The reply is returned to the originating client.

__ 5. You investigate the chained MPGWs further. Open each MPGW and enable the probes. Click **Flush** to remove any previous entries.

__ 6. Use SoapUI to submit the request again.

__ 7. Open the probe for the first service, BookingServiceSSLProxy.

__ 8. Refresh the list.

__ 9. Expand the transaction entry, and select the magnifying glass for the request.

__ 10. Click the magnifying glass at the end of the request, following the Results action.

__ 11. Click the **Service Variables** tab.

__ 12. Scroll to the bottom until you find the variable **var://service/transaction-id**. The following images show some sample entries.

__ 13. Just a few entries further down, you see the entries for the incoming request **var://service/URL-in**, and the outgoing request, **var://service/URL-out**. Notice that the incoming request is over HTTP, and the outgoing request is over HTTPS.

__ 14. Scroll back up the list to find the variable **var://service/global-transaction-id**.

__ 15. Notice that the transaction-id and global-transaction-id have the same value. Because this transaction is the originating transaction in a possible chain of services, the global transaction ID is the originating transaction ID.

__ 16. Click the **Headers** tab.

__ 17. Notice the X-Global-Transaction-ID HTTP header, and that it has the same value as the global transaction ID. DataPower added this header automatically. This behavior is how the following services get the global transaction ID.

__ 18. Close this request entry.

__ 19. Switch to the probe for the BookingServiceProxy.

__ 20. Refresh the list.

__ 21. Expand the transaction.

__ 22. Click the request entry.

__ 23. Click the **Headers** tab.

__ 24. Notice the X-Global-Transaction-ID header, and that its value is the same as from the previous probe.

__ 25. Click the **Service Variables** tab.

__ 26. Scroll down until you see the variable **var://service/global-transaction-id**. Notice that it has the same value as the HTTP header, and the same value as the global transaction ID specified in the previous transaction.

__ 27. Scroll towards the bottom until you see the variable **var://service/transaction-id**. Notice that it has its own unique transaction ID.

The global transaction ID can be used to correlate different DataPower transactions that are involved in the same client request.

__ 28. Scroll a bit further down to the **URL-in** and **URL-out** service variables. Notice that it is HTTPS coming in, and HTTP going out.

var://service/URL-in	string	'https://172.16.78.24:12322 /BookingService'
var://service/URL-out	string	'http://dp_internal_ip:9080/BookingService'

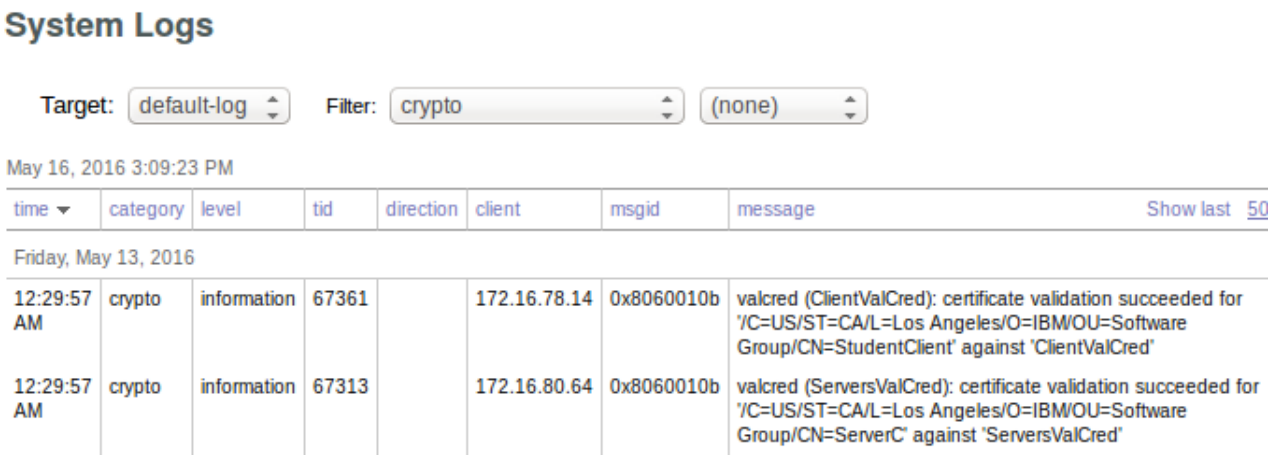
__ 29. Disable the probes, close the probe transaction list windows, and close any log windows.

__ 30. If it is visible in the banner, click **Save changes**.

Optional

If you want to try mutual authentication, recall that the ServerC SSL server profile requires mutual authentication. You can change the **Custom SNI hostname** in the **StudentClientProfile** SSL client profile to `serverC` and send the SoapUI request again.

You can set the log filter to **crypto**, and the server and client authentication should be listed.



End of exercise

Exercise review and wrap-up

In this exercise, you generated key and certificate objects on the DataPower gateway. The key and certificate objects are used to create an identification credential objects and validation credential objects. These objects are referenced by the various SSL profile objects that are needed to create an SSL connection.

You used these objects to configure both server-side and client-side SSL on two separate multi-protocol gateways.

Exercise 6. Implementing a service level monitor in a multi-protocol gateway

Estimated time

00:30

Overview

In this exercise, you specify SLM criteria to a multi-protocol gateway. You then send a series of requests, and observe the responses and log entries. To receive SLM-only log messages, you create a custom log target.

Objectives

After completing this exercise, you should be able to:

- Specify service level monitoring criteria for a multi-protocol gateway
- Inspect and edit an SLM policy object
- Create an SLM Resource Class object
- Create a custom log target for SLM events

Introduction

In an earlier exercise, you created a `BookingServiceProxy` multi-protocol gateway. A test suite is configured in SoapUI to send a `BookingRequest` SOAP message to the `BookTravel` operation multiple times. In the first section, you run the script to see the non-monitored responses. Because the SLM-related log messages might be difficult to find in the plethora of log messages, you then create a custom log target that captures only the SLM-related log messages.

In another section, you create an SLM Resource Class object that identifies a specific resource request. The MPGW service policy is edited to add an SLM action to the request rule. As part of specifying the SLM action, you define an SLM Policy object. The SLM resource class is used in the configuration of the SLM statement within the SLM policy. Again, you run the load test to observe the SLM behavior. The SLM behavior is then changed from throttling to shaping. Another load test is run to observe the new behavior.

Requirements

To complete this exercise, you need:

- Access to the DataPower gateway

- Completion of the previous exercises (see the Preface section in the exercise instructions for details)
- SoapUI, to send requests to the DataPower gateway
- The BookingServiceBackend web service that runs on the DataPower gateway in the `FLYServices` domain
- Access to the `<lab_files>` directory

Exercise instructions

Preface

- This exercise also depends on the previous completion of Exercise 4: Adding error handling to a service policy.
- Remember to use the domain and port address that you were assigned in the exercise setup. **Do *not*** use the default domain.
- The references in exercise instructions refer to the following values:
  - <*lab_files*>: Location of the student lab files. Default location is: `/usr/labfiles/dp/`
  - <image_ip>: IP address of the student image (use `/sbin/ifconfig` from a terminal window to obtain value).
  - <*dp_internal_ip*>: IP address of the DataPower gateway development and administrative functions that are used by internal resources such as developers.
  - <*dp_public_ip*>: IP address of the public services on the gateway that is used by customer and clients.
  - <*dp_WebGUI_port*>: Port of the WebGUI. The default port is `9090`.
  - <*nn*>: Assigned student number. If no instructor is present, use “`01`”.
  - <*studentnn*>: Assigned user name and user account. If no instructor is present, use “`student01`”.
  - <*studentnn_password*>: Account password. In most cases, the initial value is the same as the user name. You are prompted to create a password on first use. Write it down.
  - <*studentnn_domain*>: Application domain that the user account is assigned to. If no instructor is present, use “`student01_domain`”.
  - <*FLY_booking_port*>: Port number that the back-end BookingServices web services listen on. The default port is 9080.
  - <*wsp_proxy_port*>: 12*nn*5, where “*nn*” is the two-digit student number. This port number is the listener port for the `BookingServiceWSProxy` service.

6.1. Initialize the lab environment

Some setup activities are required to properly configure the lab environment and determine IP addresses and ports.

__ 1. If you have not yet performed the setup activities, you must go to **Appendix B: Lab environment setup**. Complete those activities before proceeding.

These activities need to be performed only once for the course.

6.2. Test the existing MPGW with SoapUI

Send a SOAP message to the `BookingServiceProxy` MPGW over HTTP.

__ 1. In SoapUI, expand the **BookingServices** project until **BookTravel - 01 - Initial Request** is visible.

__ 2. Double-click **BookTravel - 01 - Initial Request** to open the request.

__ 3. Ensure that the endpoint URL appears as:
`http://${dp_public_ip}:${mpgw_booking_port}/BookingService/`

__ 4. Click the green **Submit** arrow to POST the message to `BookingServiceProxy`.

The response tab shows a valid booking response.

You confirmed that the SOAP message is working properly. You load test the system by using this message in the following sections.

6.3. Test the existing BookingServiceProxy by using the load test

A SoapUI load test is provided to simulate numerous invocations of the **BookTravel** operation. The test sends a <BookingRequest> message ten times.

In this section, you use the load test to invoke the web service without any SLM statements in effect.

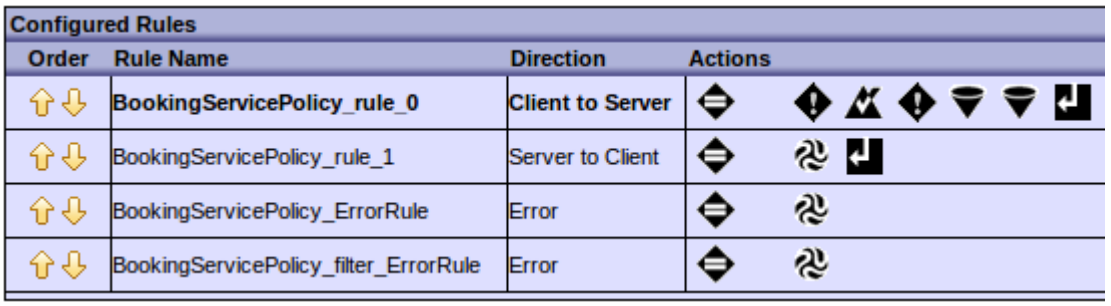
__ 1. Log in to your domain, and switch to the Blueprint Console.

__ 2. Use **Administration > Debug > Troubleshooting** to set the Log Level to **debug**. This level gives you the most detailed log entries.

__ 3. Open **BookingServiceProxy** from the All Services page in the Blueprint Console.

__ 4. Edit the **BookingServicePolicy** service policy.

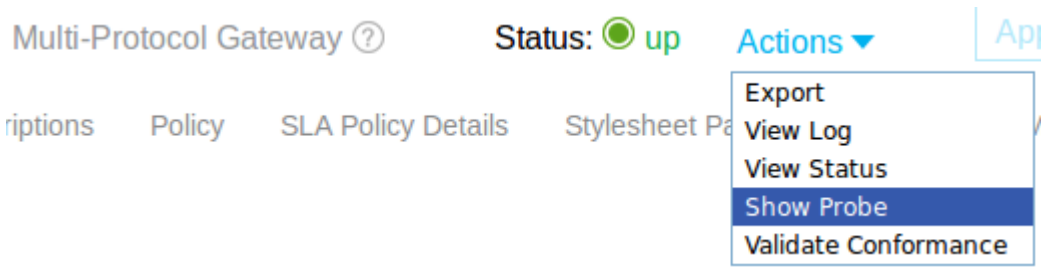
__ 5. Scroll down to the **Configured Rules** section. Verify that no SLM actions exist in any of the request rules. You can hover the mouse over each action to get its specifics.



__ 6. Close the policy editor for now.

__ 7. Enable the Multi-Step Probe so you can easily review the messages sent to the web service proxy.

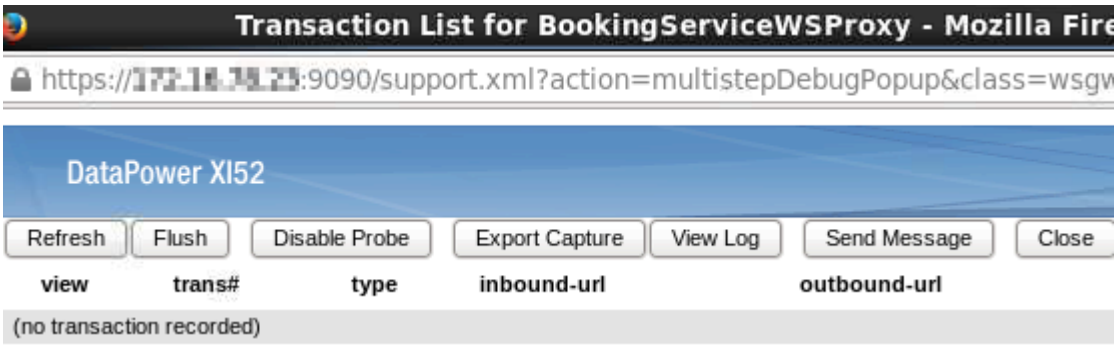
__ a. Click **Actions > Show Probe** link that is at the upper right of the MPGW configuration page.



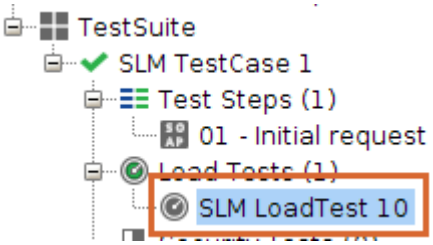
__ b. Click **Enable Probe**.

__ c. Click **Close** to close the enable probe confirmation window.

__ d. Leave the probe transaction list window for the `BookingServiceProxy` open. You return here often for the remainder of the exercise.



__ 8. In SoapUI, expand the **TestSuite** folder until **SLM LoadTest 10** is visible.



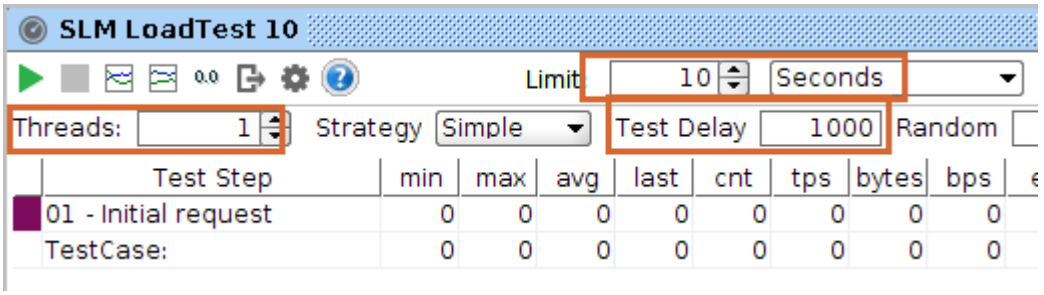
__ 9. Double-click **SLM LoadTest 10** to open the load test window. The load test sends 10 requests to the `BookingServiceProxy` in 10 seconds.

__ 10. Review the details of the Load Test.

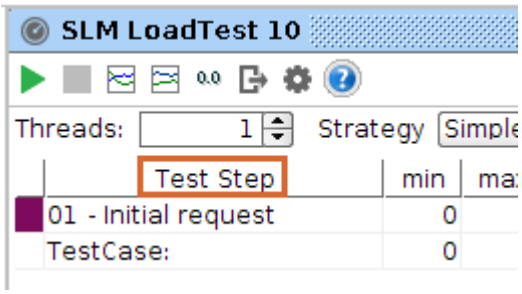
__ a. The **Limit** of **10** in *seconds* defines how long the load test runs. This test runs for 10 seconds.

__ b. **Threads** is the number of *concurrent users*. Only one is required for this test.

__ c. **Test delay** is the delay in milliseconds between message transmissions. The value is set to **1000**, so the load test sends one message every second.



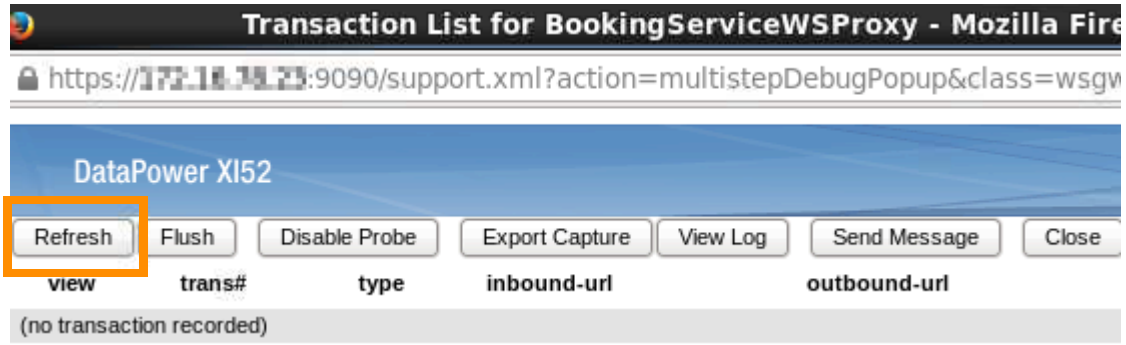
__ d. The **Test Step** of `01 - Initial Request` is the literal SOAP message that is getting sent to the URL location that is defined in the step. This request is message that you sent in the previous section:
`http://${dp_public_ip}:${mpgw_booking_port}/BookingService/`



__ 11. Click the green **Submit** arrow to begin the load test. The values in the columns in the load test change as the messages are sent. The process runs for 10 seconds.

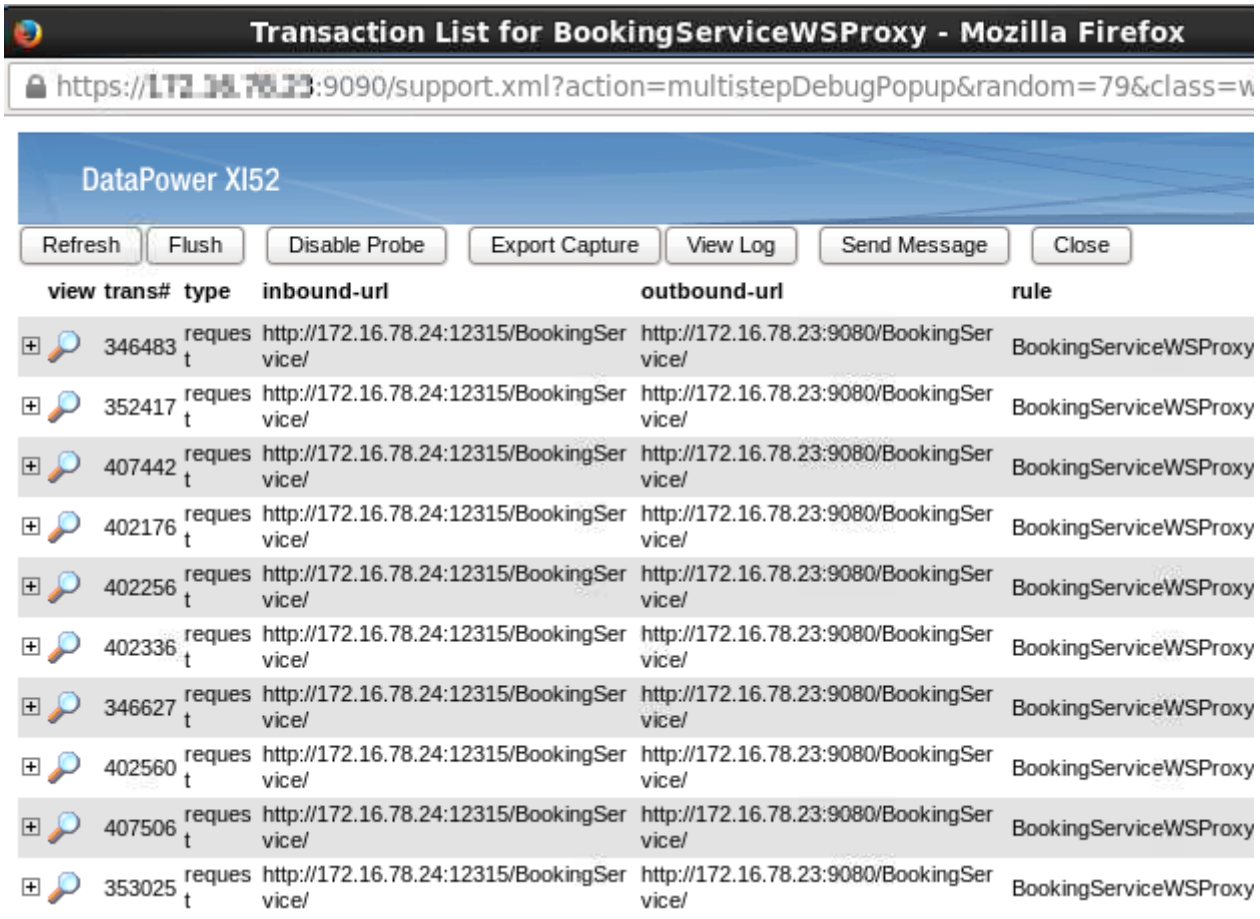
__ 12. Wait 10 seconds for the SLM Load Test to complete processing before moving onto the next step.

__ 13. Return to the probe transaction list window for the `BookingServiceProxy` and click **Refresh**.

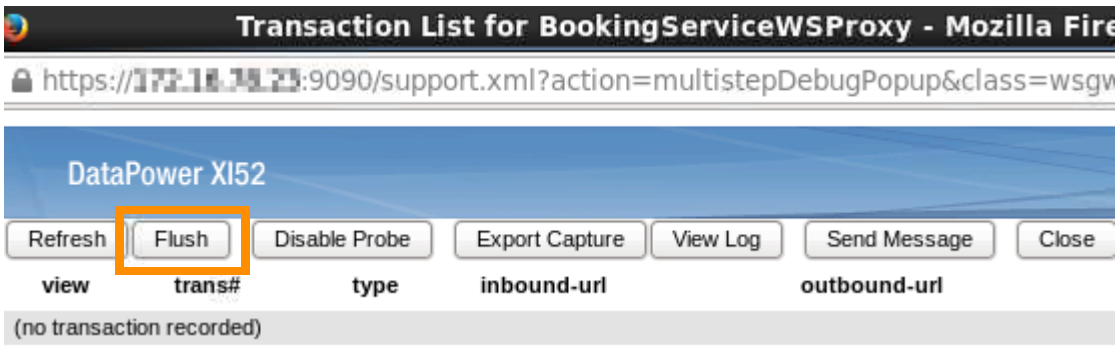


The `BookingServiceProxy` successfully processed the 10 messages. The entries are displayed in the probe window. When errors occur in the probe, the transactions are listed in red. All black transactions indicate successful execution.

The IP address for your transactions can be different than the one displayed here.



__ 14. Click **Flush** to remove all the transactions that are listed in the multi-step probe.



6.4. Create a log target for SLM log messages

When many requests are sent to the web service, SLM-related messages can be hard to find in the system log. In this section, you create a log target that collects only SLM log messages.

__ 1. In the Blueprint Console, enter `log` in the **Open** menu search field.

__ 2. Select **Log Target** in the choices. The list of defined Log Targets is displayed. You should see the default-log in the list.

__ 3. Click **New**.

__ 4. Set the following values on the **Main** twistie:

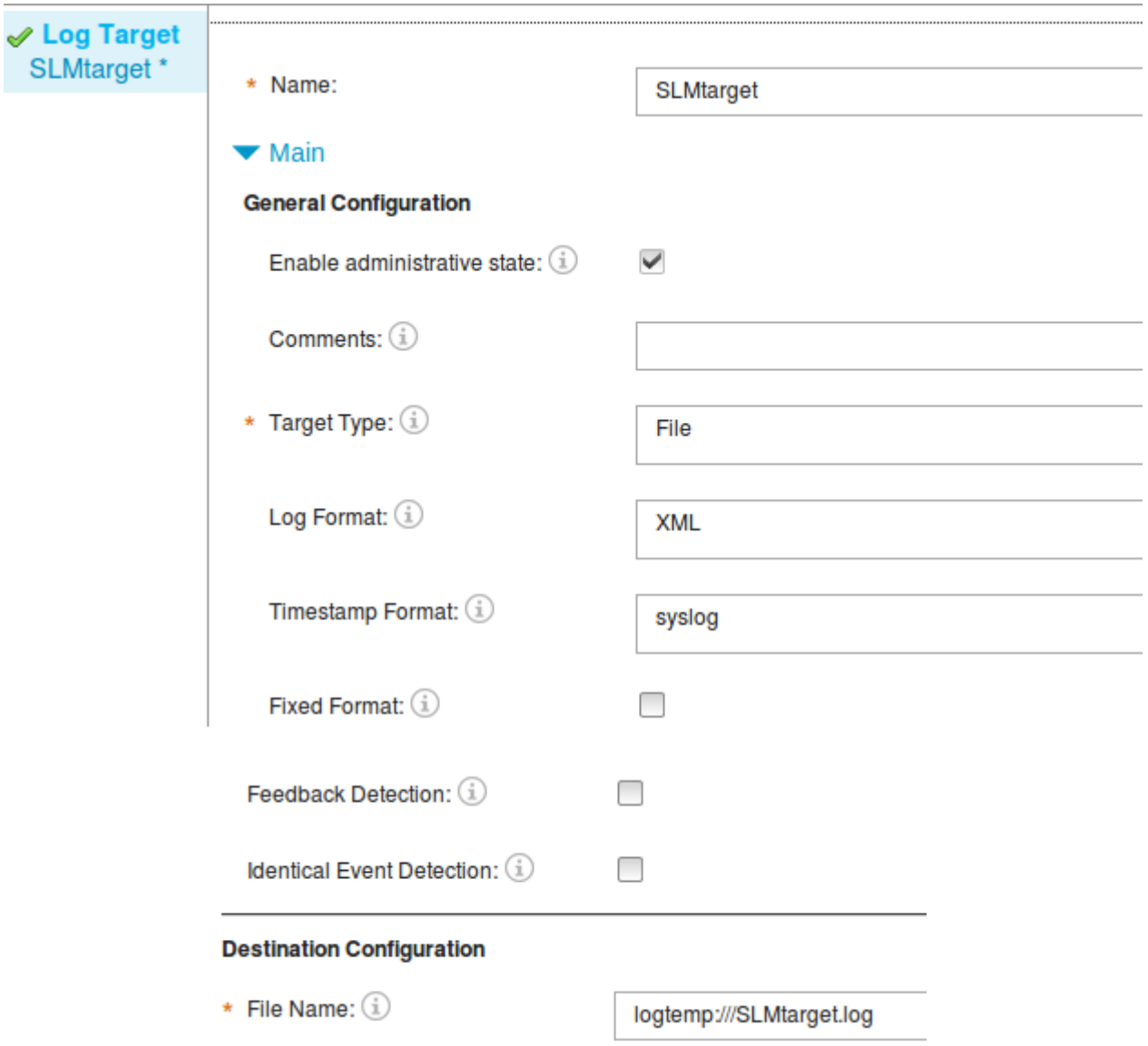
__ a. **Name**: `SLMtarget`

__ b. **Target Type**: `File`

__ c. **Log Format**: `XML`

__ d. **Fixed Format, Feedback Detection,** and **Identical Event Detection**: **off**

__ e. **File Name**: `logtemp:///SLMtarget.log`



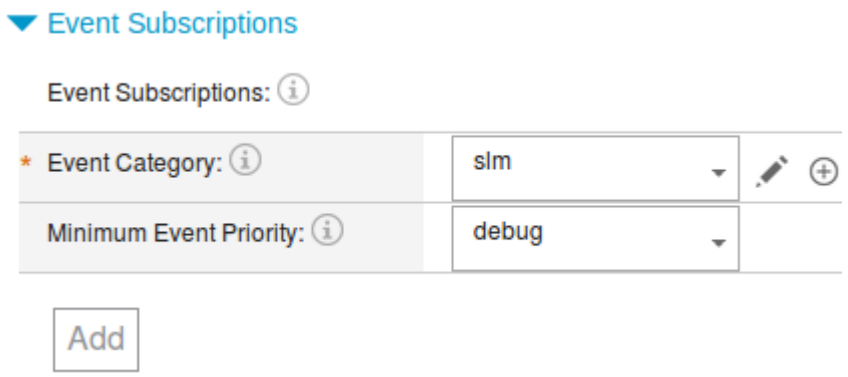
__ 5. Expand the **Event Subscriptions** twistie and add an event:

__ a. Click **Add** under the Events Subscriptions list.

__ b. Specify the following values:

- **Event Category**: `slm`

- **Minimum Event Priority**: debug



__ c. Click **Apply** for the log target.

__ 6. Click **Apply** again.

__ 7. Observe the new transaction that the `BookingServiceProxy` processes.

__ a. In the Blueprint Console, click **Status > View Logs > System Logs**. The default system log is displayed.

__ b. In SoapUI, click the green **Submit** arrow to send the message ten times again.

__ c. On the System Log page, click the “refresh log” icon - the circling arrows. The entries update.

__ d. For the **Target** field, change it from **default-log** to **SLMtarget**.

Note

If a custom log target is defined with a Log Format of **XML**, the log is included in the Target list, and can be viewed.

__ e. The SLMtarget log is still empty because no SLM criteria exists yet. Switch the log back to **default-log**.

6.5. Add SLM criteria to the MPGW

No **SLM action** currently exists in the service policy. If you want to manage the message traffic, you must add an SLM action.

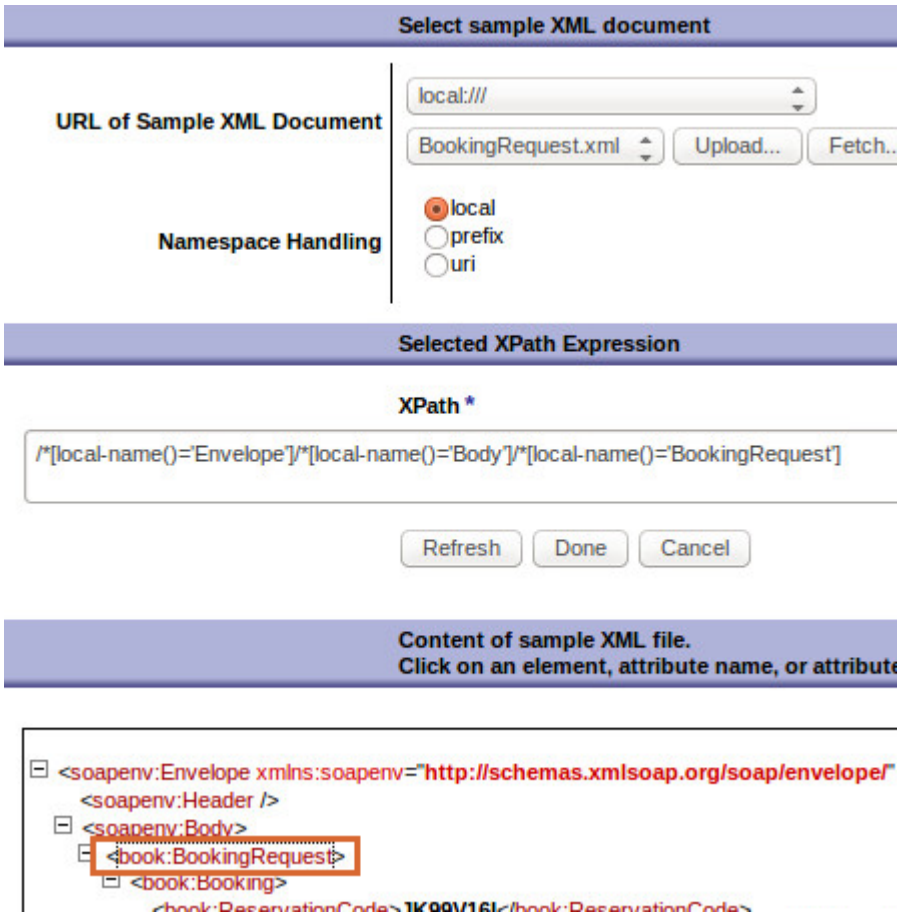
When travel booking requests exceed five requests per minute, you want to reject the new requests.

The first step is to define the resource that you want to monitor: the requests to book travel. That information is specified in the SLM Resource object. This object is used in the SLM action that is added to the MPGW.

__ 1. Create an SLM Resource Class object to identify the target traffic.

- __ a. In the Blueprint Console, enter `SLM resource` in the **Open** menu search field.
- __ b. Select **SLM Resource Class**.
- __ c. From the **SLM Resource Class** list, click **New**.
- __ d. On the SLM Resource Class page, enter a name of `BookingRequestResource`.
- __ e. Set the **Resource Type** as `XPath Expression`.
- __ f. An **XPath Filter** field appears. It expects the XPath expression that identifies the resource. You use the **XPath Tool** to create it. Click it.
- __ g. You need a sample XML document that represents a typical request. Use the **Upload** button to place `<lab_files>/BookingService/BookingRequest.xml` in the **local:///** directory.
- __ h. Set **NameSpace Handling** to `local`.
- __ i. In the lower part of the page, the sample file appears. Click in the **<book:BookingRequest>** element.

__ j. The derived XPath expression appears.



__ k. Click **Done**.

__ l. The XPath expression is placed in the **XPath Filter** field. Click **Apply**.

__ 2. Edit the **BookingServicePolicy** of the BookingServiceProxy to add an SLM action.

__ a. Use the **Services** icon to list the services.

__ b. Click **BookingServiceProxy** to open its configuration page.

__ c. Edit the **BookingServicePolicy** service policy.

__ d. In the policy editor, verify that the **Client to Server** rule is selected.

__ e. Drag the **Advanced** action to the rule and place it right after the Match action.

__ f. Scroll the list of available actions until you see the **SLM** action.

__ g. Select the SLM action and click **Next**.

__ h. A Configure SLM Action page is opens. Click the new (“+”) button for the SLM Policy.

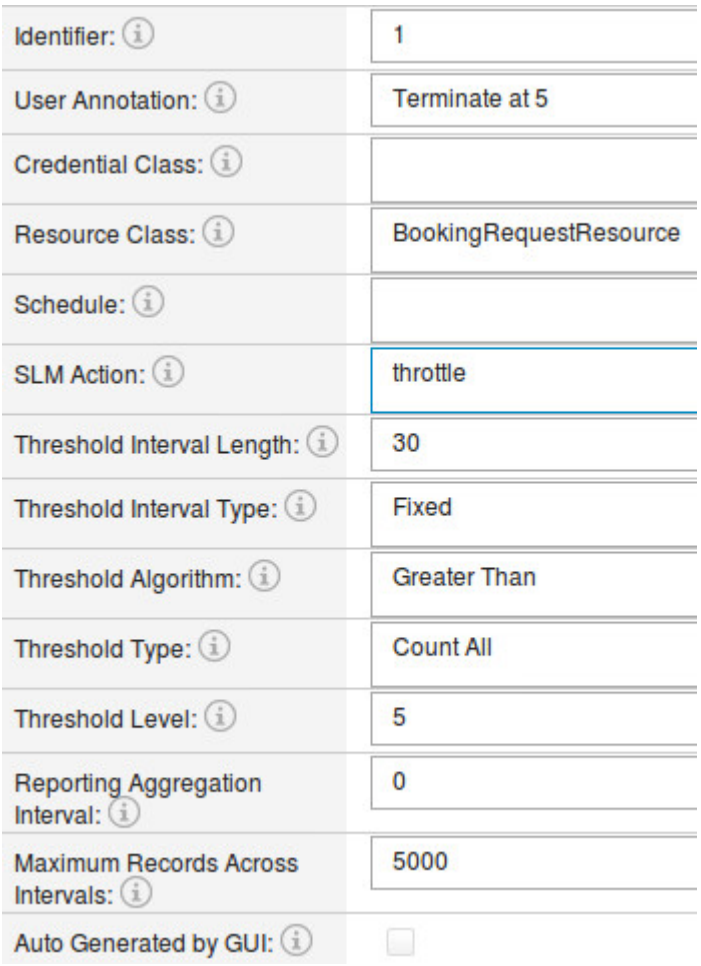
__ i. An SLM Policy page opens. Give it a name of **LimitBookingRequests**, and leave the evaluation method at **Execute All Statements**.

__ j. Click the **Statement** twistie, and click **Add**.

__ k. Enter the following values for the statement:

- **Identifier**: 1

- **User Annotation**: `Terminate at 5`
- **Resource Class**: `BookingRequestResource`
- **SLM Action**: `throttle`
- **Threshold Interval Length**: `30 seconds`
- **Threshold Level**: `5`



__ l. Click **Apply**.

__ m. Click **Done** to save the SLM action.

__ n. Click **Apply Policy** to save the updated rule.

__ o. Close the policy editor window.

__ 3. Click **Save changes**.

6.6. Test the SLM action

Now that you specified some SLM criteria, run the load test to see the results.

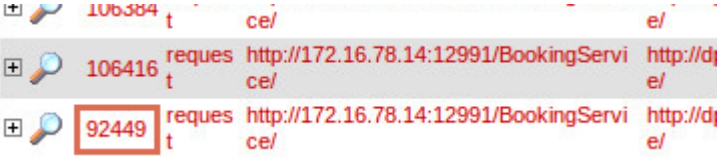
__ 1. On the probe transaction list window, click **Flush**.

__ 2. In SoapUI, click the green **Submit** arrow to send the 10 requests.

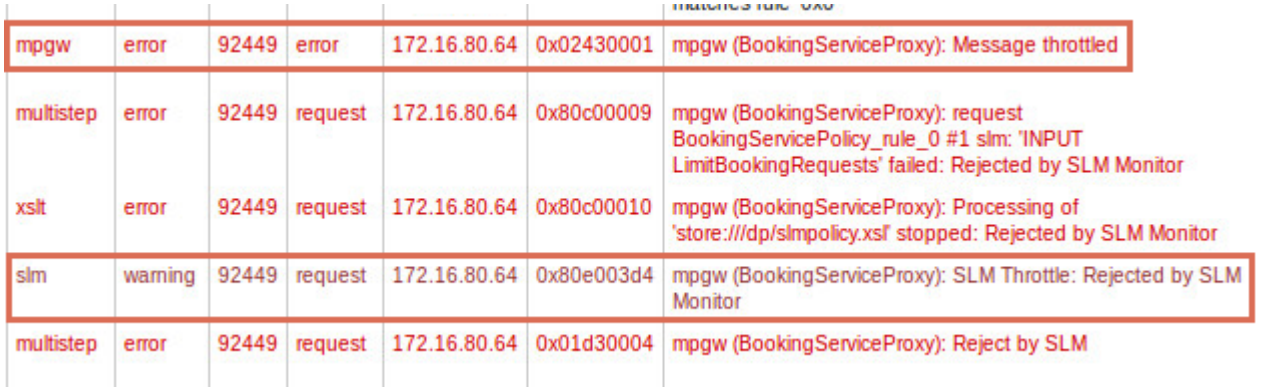
__ 3. On the probe transaction list window, click **Refresh**.

__ 4. You should see the first five entries in black. The remaining entries are in red. The red entries indicate the transactions that were “throttled” because they exceeded the 5 transactions in 30 seconds.

__ 5. To verify that the transaction failed because of the throttling, click the transaction number in the transaction list of the last transaction.



__ 6. The log for this transaction opens. Scroll down the log until you find the red entries for the transaction errors.



mpgw	error	92449	error	172.16.80.64	0x02430001	mpgw (BookingServiceProxy): Message throttled
multistep	error	92449	request	172.16.80.64	0x80c00009	mpgw (BookingServiceProxy): request BookingServicePolicy_rule_0 #1 slm: 'INPUT LimitBookingRequests' failed: Rejected by SLM Monitor
xslt	error	92449	request	172.16.80.64	0x80c00010	mpgw (BookingServiceProxy): Processing of 'store:///dp/slmpolicy.xsl' stopped: Rejected by SLM Monitor
slm	warning	92449	request	172.16.80.64	0x80e003d4	mpgw (BookingServiceProxy): SLM Throttle: Rejected by SLM Monitor
multistep	error	92449	request	172.16.80.64	0x01d30004	mpgw (BookingServiceProxy): Reject by SLM

The transaction was terminated because it exceeded the limits set by the SLM statement within the SLM policy.

__ 7. Close the log window.

__ 8. Click **Flush** to clear the transactions in the list.

6.7. Change the SLM statement to “shape”

In this section, you change the SLM statement within the SLM policy to “shape” the traffic, rather than outright “throttle” it. The “shape” action buffers the traffic to stay within the limit.

__ 1. From the Blueprint Console **Open** menu, enter `slm` in the search field.

__ 2. Select **SLM Policy**. For this section, you directly edit the SLM policy object rather than go through the MPGE service policy.

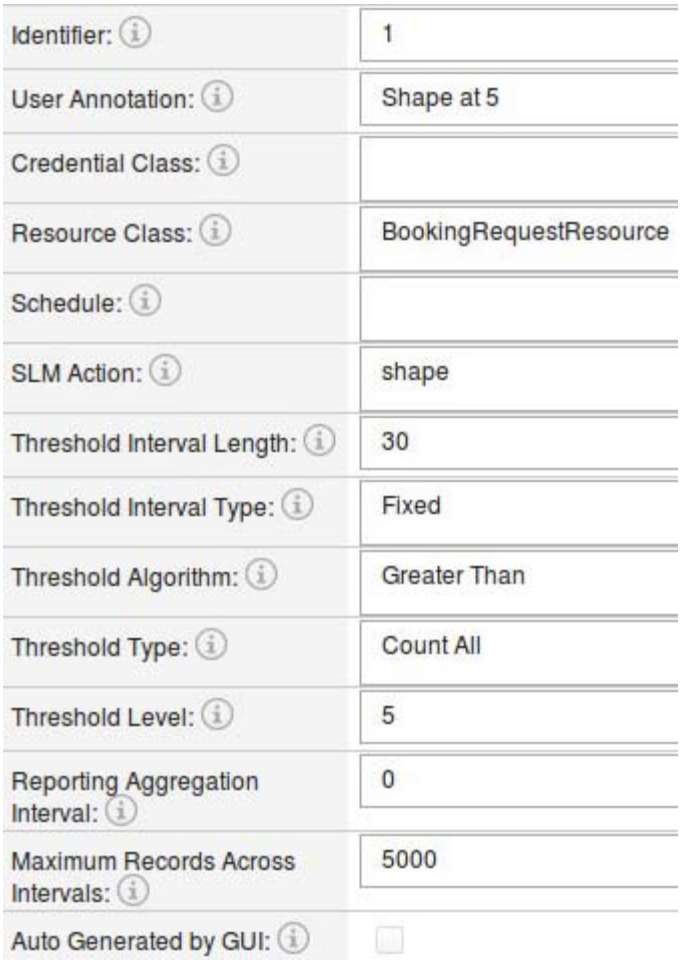
__ 3. On the SLM Policy list page, click **LimitBookingRequests**.

__ 4. Expand the **Statements** twistie.

__ 5. Change the user annotation to `Shape at 5`.

__ 6. Change the SLM action to `shape`.

__ 7. Leave the other settings as-is. Click **Apply**.



__ 8. Click **Save changes**.

6.8. Test the SLM action with “shape”

You send the same messages to the service, but with a different SLM action.

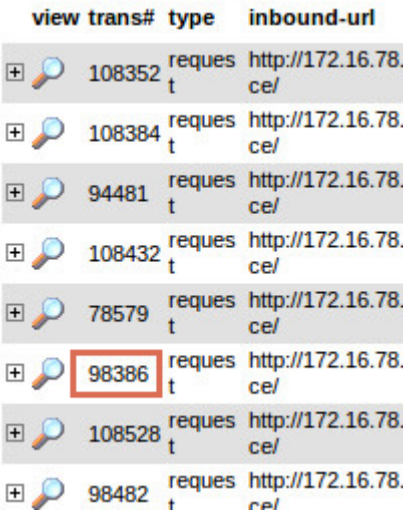
__ 1. Return to SoapUI and click the green **Submit** arrow in the `SLM LoadTest 10` window to begin the load test. The process runs for 10 seconds.

__ 2. *Wait at least 60 seconds* for the transactions to complete processing before moving onto the next step.

__ 3. Return to the probe transaction list window and click **Refresh**.

__ 4. This time, all of the transactions are in black; none failed due to the SLM action.

__ 5. You can see the SLM shaping behavior in the log. Because the shaping behavior is configured to start at the sixth transaction within the 30-second window, click the transaction number of the sixth transaction.



__ 6. Scroll to almost the bottom of the list, and search for the start of the SLM entries (The tid, client, and msgid columns were contracted in the screen capture to make the image fit.).

2:26:02 PM	multistep	information	$	request	17d2	mpgw (BookingServiceProxy): rule (BookingServicePolicy_rule_0): #2 on-error
2:26:02 PM	memory-report	debug	$	request	178d	mpgw (BookingServiceProxy): Processing [Rule (BookingServicePolicy_rule_0 Output(NULL)] finished: memory used 1357128
2:26:02 PM	multistep	information	$	request	17d2	mpgw (BookingServiceProxy): rule (BookingServicePolicy_rule_0): #1 slm: 'INF
2:26:02 PM	memory-report	debug	$	request	178d	mpgw (BookingServiceProxy): Processing [Rule (BookingServicePolicy_rule_0 Input(INPUT), Output(NULL)] finished: memory used 840256
2:26:02 PM	xmlfilter	information	$	request	1736	mpgw (BookingServiceProxy): Accept set.
2:26:02 PM	slm	debug	$	request	17e7	mpgw (BookingServiceProxy): Identifier 1 Shape at 5 Matched resource and c
2:25:44 PM	slm	debug	$		1793	slm-policy (LimitBookingRequests): SLM statement 1 triggered shape action
2:25:44 PM	slm	debug	$	request	17e9	mpgw (BookingServiceProxy): Identifier 1 resource type(xpath-filter) resource v Roberts 314 S. Wells St Chicago IL 60606 USA) using match type() matches
2:25:44 PM	xslt	debug	$		17ac	xmlmgr (default): xslt Compilation Request: Found in cache store:///dp/slmpoli

Notice that the **shape** action was triggered at 2:25:44 in the test.

The transaction is buffered until the SLM restriction is released. That occurred at 2:26:02.

At the top, the next action in the rule, the On Error action, is executed.

__ 7. Close the log window.

__ 8. In the transaction list window, disable the probe.

__ 9. Close the transaction list window.

End of exercise

Exercise review and wrap-up

In this exercise, you modified the `BookingServiceProxy` MPGW to enforce SLM criteria. You created an SLM policy object, and configured an SLM statement. You tested the behavior of the throttle and shape SLM actions.

Exercise 7. Using a DataPower pattern to deploy a service

Estimated time

00:30

Overview

In this exercise, you play the role of a pattern deployer. You specify the values for the exposed points of variability (POV) in a specific pattern, and deploy the pattern into a new generated service.

Objectives

After completing this exercise, you should be able to:

- Import a pattern
- Specify the values for the points of variability in the pattern
- Deploy the pattern into a generated service

Introduction

A pattern creator created a pattern that generates a multi-protocol gateway (MPGW) that secures access to the Booking Service web service. The security is provided by a AAA policy that uses LDAP to store authentication and authorization information. In this exercise, you generate a service to implement this behavior. You import the pattern into your student domain. You then use the Blueprint Console to “deploy” the pattern into a generated service, specifying the values that are exposed by the pattern creator. These values “customize” the pattern into a generated service for your specific situation. You test the deployment by using SoapUI to invoke the web service.

Requirements

To complete this exercise, you need:

- Access to the DataPower gateway
- Completion of the previous exercises (see the Preface section in the exercise instructions for details)
- SoapUI, to send requests to the DataPower gateway
- The FLYServices domain that contains the back-end web service
- Access to the `<lab_files>` directory
- An LDAP server that is pre-loaded with the authentication entries that are used by the service

Exercise instructions

Preface

- Remember to use the domain and port address that you were assigned in the exercise setup. **Do *not*** use the default domain.
- The references in exercise instructions refer to the following values:
  - <*lab_files*>: Location of the student lab files. Default location is: `/usr/labfiles/dp/`
  - <image_ip>: IP address of the student image (use `/sbin/ifconfig` from a terminal window to obtain value).
  - <*dp_internal_ip*>: IP address of the DataPower gateway development and administrative functions that are used by internal resources such as developers.
  - <*dp_public_ip*>: IP address of the public services on the gateway that is used by customer and clients.
  - <*dp_WebGUI_port*>: Port of the WebGUI. The default port is `9090`.
  - <*nn*>: Assigned student number. If no instructor is present, use “`01`”.
  - <*studentnn*>: Assigned user name and user account. If no instructor is present, use “`student01`”.
  - <*studentnn_password*>: Account password. In most cases, the initial value is the same as the user name. You are prompted to create a password on first use. Write it down.
  - <*studentnn_domain*>: Application domain that the user account is assigned to. If no instructor is present, use “`student01_domain`”.
  - <*FLY_booking_port*>: Port number that the back-end BookingServices web services listen on. The default port is 9080.
  - <*mpgw_patterns_port*>: 12*nn8*, where “*nn*” is the two-digit student number. This port number is the listener port for the BookingService MPGW that is generated from a pattern.

7.1. Initialize the lab environment

Some setup activities are required to properly configure the lab environment and determine IP addresses and ports.

__ 1. If you have not yet performed the setup activities, you must go to **Appendix B: Lab environment setup**. Complete those activities before proceeding.

These activities need to be performed only once for the course.

7.2. Import a pattern into your application domain

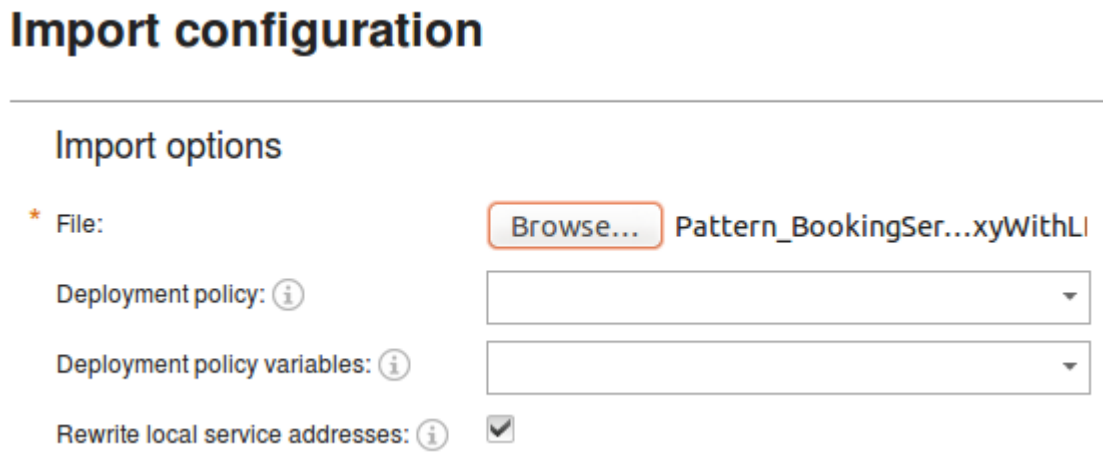
Before deploying the pattern, you must first import it.

__ 1. Log in to your student domain **student*nn*_domain** on the gateway.

__ 2. Switch to the Blueprint Console.

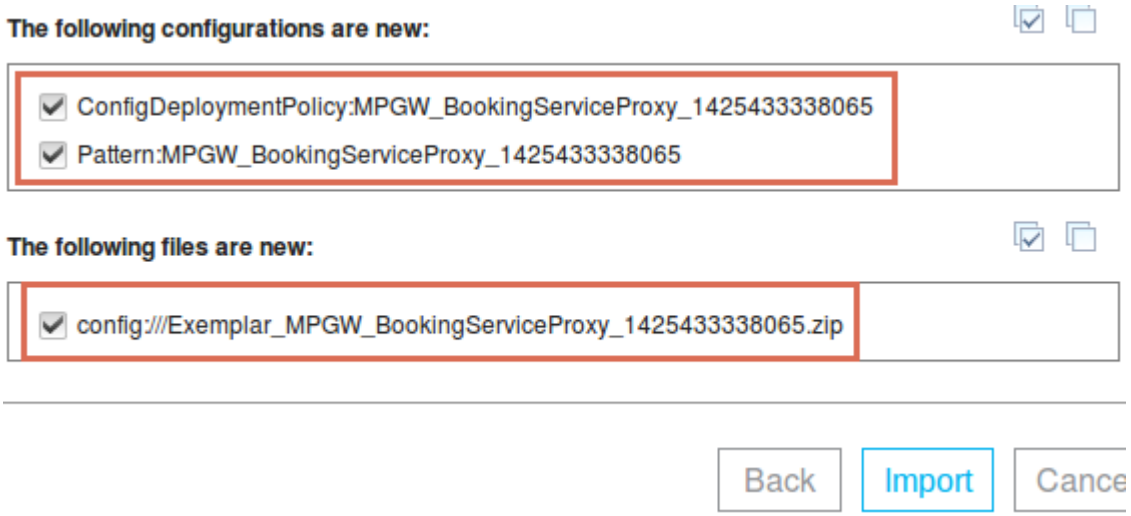
__ 3. Use the **Open** menu to click **Import Configuration**.

__ 4. For the **File** field, browse to ***<lab_files>*/Patterns**, and select the file **Pattern_BookingServiceProxyWithLDAP_AAA.zip**.



__ 5. Click **Next**.

__ 6. The Import Configuration page shows the files that are imported from the `.zip` file.



- The “Exemplar” file is placed in the **config:** directory. It is an exported version of the original source service.
- The “Pattern” configuration contains the specifics of how this pattern must be deployed, and what values are exposed.

- The “Deployment Policy” configuration is the standard DataPower deployment policy object that manages the actual value substitutions in the source configuration to make it the generated service.

__ 7. Click **Import**.

__ 8. The page should show a successful import. Click **Close**.

Information

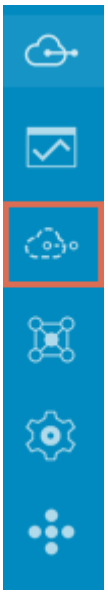
Normally, a pattern is imported by using the Blueprint Console. You use this technique to import the pattern so that you can see the constituents of the imported `.zip` file. The underlying files are not shown when importing a pattern from within the Blueprint Console.

__ 9. Save the changes.

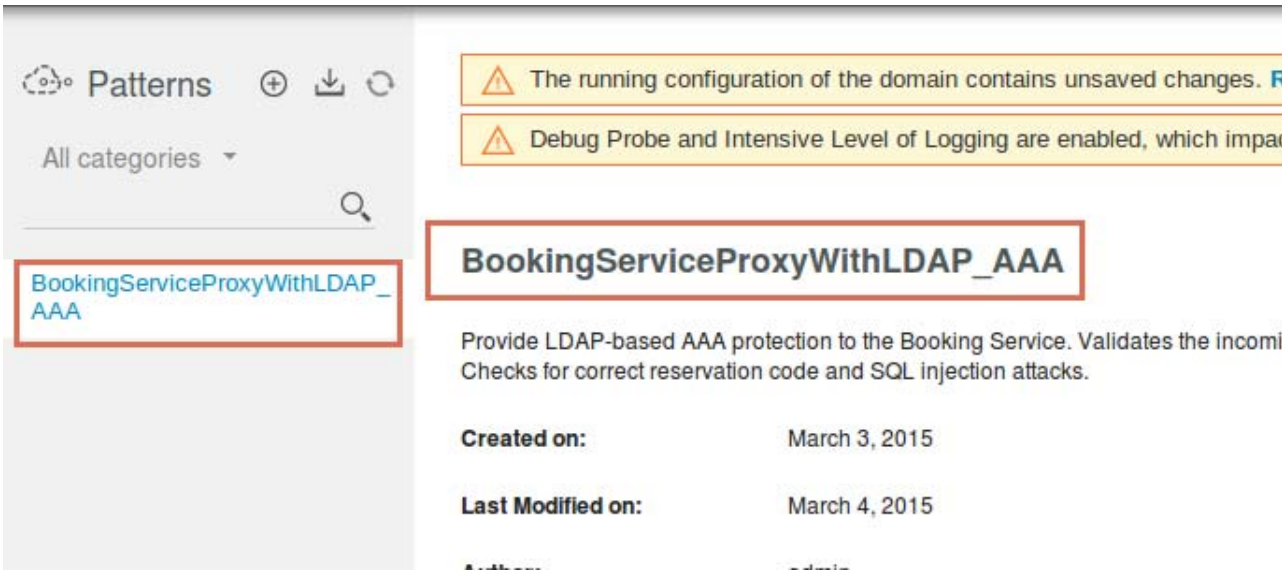
The pattern should now be available in the Blueprint Console.

7.3. Deploy a pattern

__ 1. In the navigation area, click the **Patterns** icon.



__ 2. The page repaints to list any patterns in this domain. You should see the new pattern that is listed in the navigation area, and the pattern details shown in the display area.



Troubleshooting

If the new pattern does not display initially, switch to the WebGUI tab, and reinitiate the Blueprint Console. Then, click the **Patterns** icon in the navigation area again.

__ 3. Read the documentation on the pattern in the display area.

__ 4. Click **Deploy**.

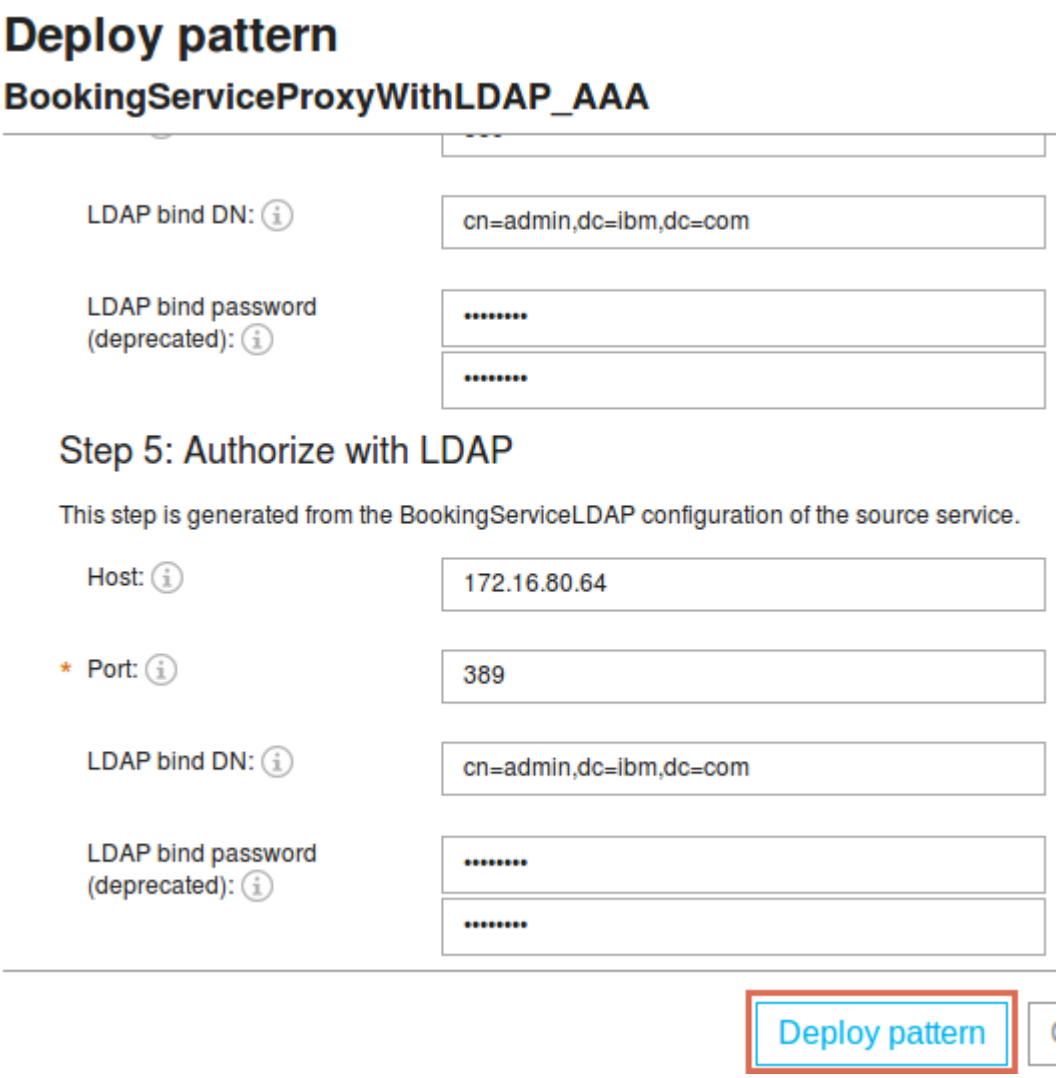
__ 5. In the presented Deploy pattern window (be sure to use the scroll bar on the right), enter:

- **Service name**: `BookingServiceProxyFromPattern`
- **Description**: `My service generated from a pattern`
- **Destination URL**: `http://<dp_internal_ip>:9080/BookingService`
- **HTTP Connection – IP address**: *<dp_public_ip>*
- **HTTP Connection – Port**: *<mpgw_patterns_port>*
- **Authenticate with LDAP – Host**: *<image_ip>*
- **Authenticate with LDAP – Port**: 389
- **Authenticate with LDAP – LDAP bind DN**: `cn=admin,dc=ibm,dc=com`
- **Authenticate with LDAP – LDAP bind password**: `passw0rd`
- **Authorize with LDAP – Host**: *<image_ip>*
- **Authorize with LDAP – Port**: 389
- **Authorize with LDAP – LDAP bind DN**: `cn=admin,dc=ibm,dc=com`
- **Authorize with LDAP – LDAP bind password**: `passw0rd`

Information

The fields that are listed in the deployment wizard are the “points of variability” that the pattern creator decided to expose to a pattern deployer.

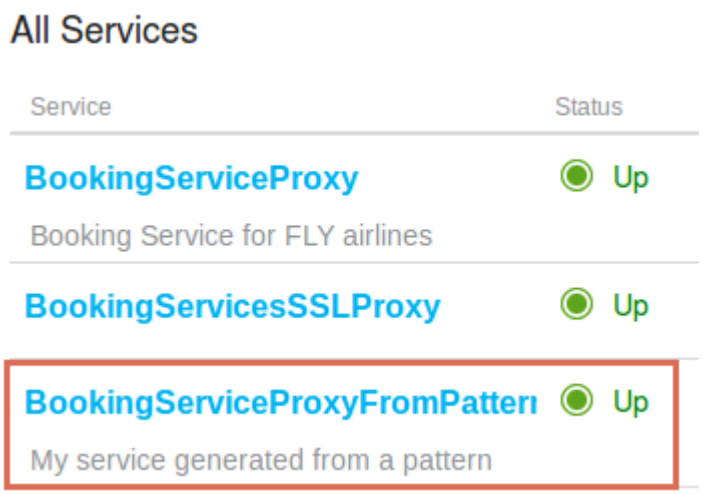
__ 6. Click **Deploy pattern**.



__ 7. The service is generated. The Deploy Pattern page closes. A temporary window states that the service was generated successfully.

__ 8. In the navigation area, click the **Services** icon.

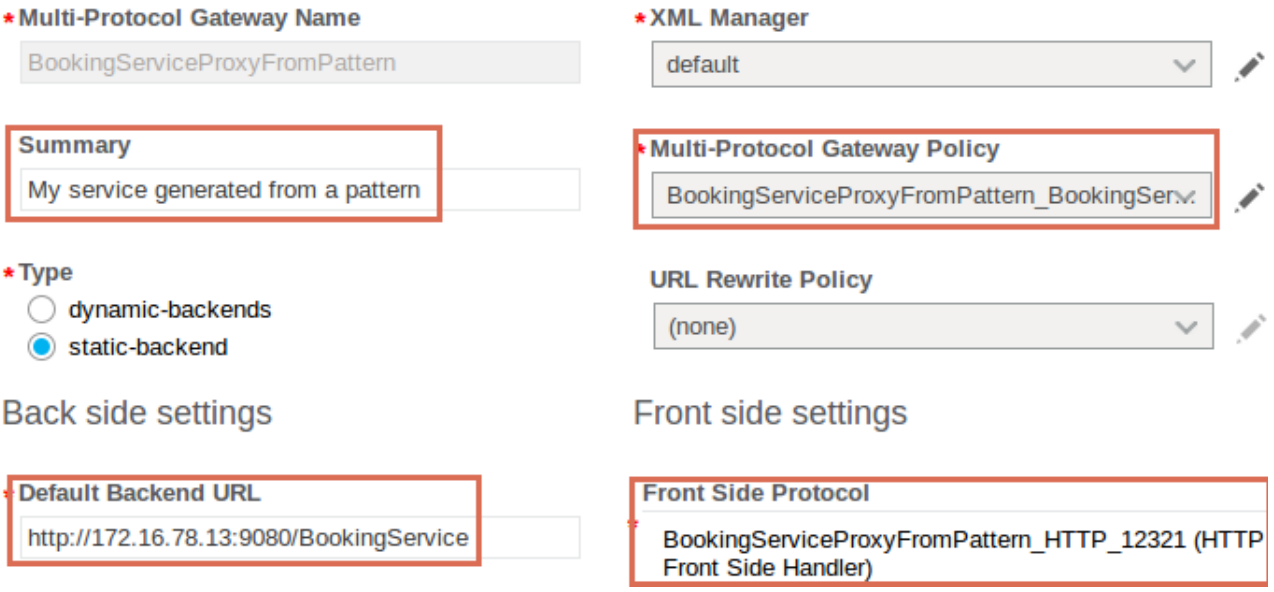
__ 9. The newly generated service should appear in the list.



__ 10. Select the service.

__ 11. On the Multi-Protocol Gateway configuration page, notice the following points:

- The **Summary** field contains the text that you entered in the **Description** field in the Deploy Pattern wizard.
- The name of the **Multi-Protocol Gateway Policy** is the name of the policy object from the source service (BookingServiceProxy), prefixed by the generated MPGW name.
- The **Default Backend URL** field contains the URL string that you entered in the wizard. This field does allow the use of host aliases.
- The **Front Side Protocol** handler object gets a unique name that uses the same naming pattern that was used for the service policy.

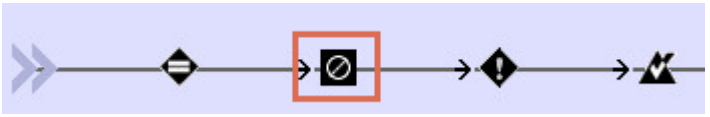


__ 12. Open the front side handler object and notice that it uses the HTTP Connection - Port value from the wizard.

__ 13. Edit the **BookingServiceProxyFromPattern_BookingServicePolicy** service policy.

__ 14. Select the Client to Server rule.

__ 15. In the rule, double-click the AAA action.



__ 16. To deploy this service from the pattern, you did not need to know the intricacies of configuring a AAA policy. The creator of the pattern needed the skill. Edit the AAA policy within the AAA action.

__ 17. The first page of the policy defines how to extract the user's identity. In this case, the AAA policy uses the HTTP authentication header. Scroll down to click **Next**.

Troubleshooting

If the scroll bar on the right does not work correctly, click inside the page and use the arrow keys to scroll.

__ 18. This section of the AAA policy defines how to authenticate the user's identity that it extracts. In this case, it uses an LDAP server. You should see the LDAP parameters that you entered in the pattern deployment wizard. It also defines how to map the authenticated identity to a different credential ("None"). Scroll down and click **Next**.

__ 19. The next page defines how to extract the requested resource ("URL sent by client"), and how to map the resource ("None"). Click **Next**.

__ 20. The next section specifies how to authorize the extracted user for the requested resource. Again, you should see the LDAP parameters that you entered during pattern deployment. Click **Next**.

__ 21. The last page of the AAA policy wizard shows some post-processing options, of which none are of importance for now. Click **Cancel**.

__ 22. Cancel the AAA action.

__ 23. Close the policy editor.

__ 24. Save the changes.

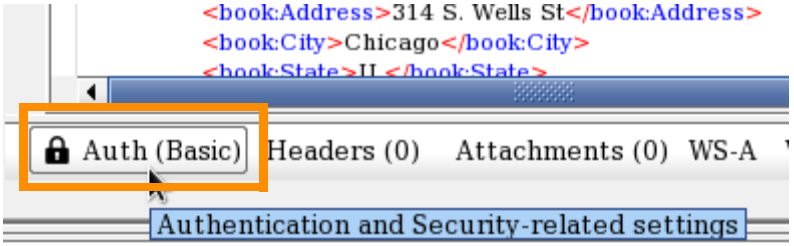
7.4. Test the generated service

In this section, you test the new MPGW that uses an LDAP server for authentication and authorization.

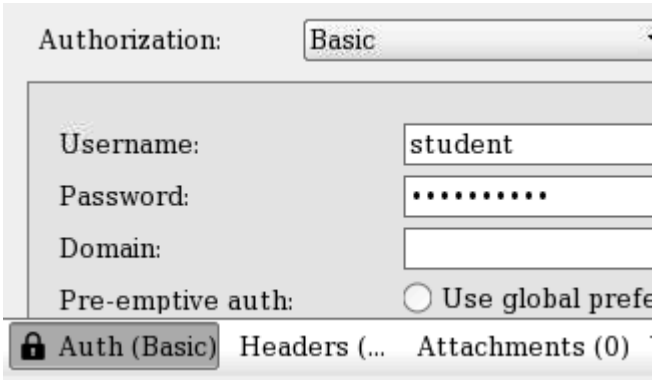
__ 1. In SoapUI, open the Booking Service requests until you see the **16- AAA LDAP Request**.

__ 2. Open the request.

__ 3. Click the **Auth** tab that is in the bottom portion of the 16 - AAA LDAP Request.



__ 4. Enter `student` as the **Username**, and `passw0rd` as the **Password**.



__ 5. Ensure that the **Outgoing WSS** and **Incoming WSS** are blank.

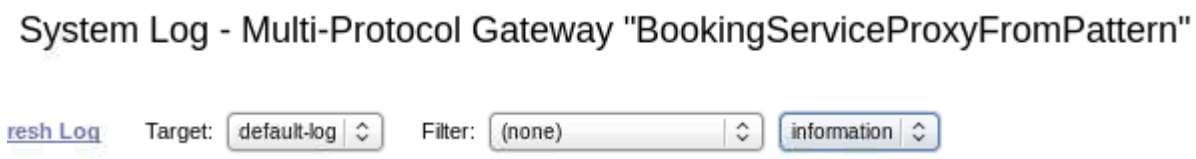
__ 6. Ensure that the endpoint URL appears as:
`http://${dp_public_ip}:${mpgw_patterns_port}/BookingService/`

__ 7. Click the green **Submit** arrow to POST the message to BookingServiceProxyFromPattern.

__ 8. Confirm that the <book:BookingResponse> is received.

__ 9. On the Multi-Protocol Gateway configuration page, click **Actions > View log**.

__ 10. At the top of the log page, set the Filter to **information**. This setting makes the log show only the entries that are at “information” level or more severe.



__ 11. Scroll down a page or two to the “request” set of entries. Look for the entries that indicate LDAP activity.

request	172.16.80.5	0x838000f1	mpgw (BookingServiceProxyFromPattern): ldap authorization succeeded with credential 'cn=stude '/BookingService/'
request	172.16.80.5	0x83800016	mpgw (BookingServiceProxyFromPattern): ldap authentication succeeded with (http-basic-auth, us password="*********"configured-realm='login')
request	172.16.80.5	0x83800095	mpgw (BookingServiceProxyFromPattern): get-user-dn-result: cn=student,dc=ibm,dc=com

__ 12. If you want, you can change the Filter in the log to **debug** to see more details.

__ 13. In SoapUI, change the password to an invalid value. The response displays a SOAP fault of “rejected by policy”.

__ 14. Close the log window.

End of exercise

Exercise review and wrap-up

In this exercise, you deployed a pattern into a new MPGW service. After successful deployment, you tested the service for successful execution.

You were able to deploy a service that used a capability (LDAP) that you did not need expertise on.

Appendix A. Exercise solutions

This appendix describes:

- The dependencies between the exercises and other tools.
- How to load the sample solution configurations for the various exercises. The solutions were exported from the appliance into a `.zip` file. You can import a sample solution into your domain.

Part 1: Dependencies

Certain exercises depend on previous exercises, and on other resources, such as the need for the back-end application server to support service calls. The back-end application server that is on each student image uses the variable `<image_ip>`. The `image_ip` variable is the literal IP address of the student image. Students can discover their IP address by opening a terminal window, and entering: `/sbin/ifconfig`

Table 1. Dependencies

Exercise	Depends on exercise	Uses cURL	Uses SoapUI	Uses Baggage web service	Uses Booking web service
1: First exposure to the DataPower developer environment		Yes			
2: Creating a BookingService gateway			Yes		Yes
3: Enhancing the BookingService gateway	2		Yes		Yes
4: Adding error handling to a service policy	3		Yes		Yes
5: Creating cryptographic objects and configuring SSL	3	Yes	Yes		Yes
6: Implementing a service level monitor in a multi-protocol gateway	4		Yes		Yes
7: Using a DataPower pattern to deploy a service			Yes		Yes

If the class is using the standard images and setup, the LDAP server is running on the student image. The Baggage and Booking services are running as services on the DataPower gateway. Therefore, each student is using a different IP address for the student image. Assuming that each student has their own DataPower virtual gateway, each student also has different IP addresses for the gateway.

If the exercises are run in the IBM remote lab environment, like Skytap, the IP addresses might be the same for each student because each student has a unique entry point into the virtualized environment.

Part 2: Importing solutions

Note

The solution files use port numbers that might already be in use. You must change the port numbers of the imported service. You might also find it necessary to update the location of the back-end application server that provides the web services.

__ 1. Determine the `.zip` file to import from the following table:

Table 2. Exercise solution files

Exercise	Compressed solution file name
1: First exposure to the DataPower developer environment	N/A
2: Creating a BookingService gateway	dev_Ex02_SimpleMPGW.zip
3: Enhancing the BookingService gateway	dev_Ex03_AdvMPG.zip
4: Adding error handling to a service policy	dev_Ex04_Errors.zip
5: Creating cryptographic objects and configuring SSL	dev_Ex05_SSL.zip
6: Implementing a service level monitor in a multi-protocol gateway	dev_Ex06_SLM.zip
7: Using a DataPower pattern to deploy a service	N/A
Booking and Baggage FLY Services	dev_FLYservices_domain.zip

__ a. The `.zip` file names begin with the naming convention `dev_Ex`*`NN`*, where *`NN`* represents the two-digit exercise number.

__ b. To import a solution to begin a new exercise, import the solution for the previous exercise. For example, if you are ready to start Exercise 4, you would import `<`*`lab_files`*`>/Solutions/dev_Ex04_Errors.zip`.

__ 2. Import the `.zip` solution file into your application domain.

__ a. From the **Control Panel**, in the vertical navigation bar, click **Administration > Configuration > Import Configuration**.

__ b. Make sure that the selection for **From** is **ZIP Bundle** and the selection for **Where** is **File**.

__ c. Click **Browse** and navigate to your respective `.zip` solution file.

__ d. Click **Next**.

__ e. In the next page, leave the files selected. Scroll down and click **Import**.

__ f. Make sure that the import is successful. Click **Done**.

__ 3. *Be sure to update the port numbers and application server location* to your local values. Because private keys (key files) are not exported, *you also must create keys and certificates*. In some exercise solutions, the key files are exported in the `local:` directory. After import, you move those files into the `cert:` directory.

__ 4. The lab exercises call two web services, **Booking Service** and **Baggage Service**. Both of the web services are in the FLY service domain. To perform the labs on another DataPower appliance, be sure to import the dev_FLYservices_domain.zip file in the **FLYServices** domain.

Appendix B. Lab environment setup

The appendix instructs how to set up the lab environment, including:

- Defining the literal variable values in SoapUI
- Testing the Booking and Baggage web service back ends
- Identifying the IP address of your student image
- Populating a convenient table with all the required variables that are used by this course

Part 1: Configure the SoapUI variables for use

The SoapUI tool supports specification of properties to reduce the redundant entry of the same value for testing. For these exercises, the client testing usually accesses the public interface of the student-created DataPower services. Rather than requiring the students to constantly enter the same value, the public IP address of the appliance is configured as a SoapUI property.

__ 1. Obtain the required variables for this course. The variable information can be found in at least on one of the following locations, based on the type of course you are taking:

- On the image desktop as a background
- On the image background from the SPVC you logged in to
- In an email you received with instructions for this course
- From your instructor if you are in a classroom (virtual or literal) environment

  The variables are:

  - The DataPower appliance's public IP address **<dp_public_ip>**
  - The DataPower appliance's internal IP address **<dp_internal_ip>**
  - The (your) student image IP address **<image_ip>**
  - Your student number (it is a two-digit number) **<nn>**

__ 2. Open SoapUI by using the icon on the desktop.

Attention

If you get a "new version available" message, close the message window and do not download or install any upgrades.

__ 3. Click **File > Preferences**.

__ 4. Click the **Global Properties** choice.

__ 5. Update the values for the following variables.

__ a. Double-click the **Value** cell for the **dp_internal_ip** property.

__ b. Replace the value (`x.x.x.x` or `1.2.3.4`) with the literal value of the ***<dp_internal_ip>*** address in the cell. That is, replace 1.2.3.4 with the IP address of the DataPower appliance that is being used for your class.

__ c. Press Enter while the cursor is in the cell. This action registers the new value.

__ d. Double-click the **Value** cell for the **dp_public_ip** property.

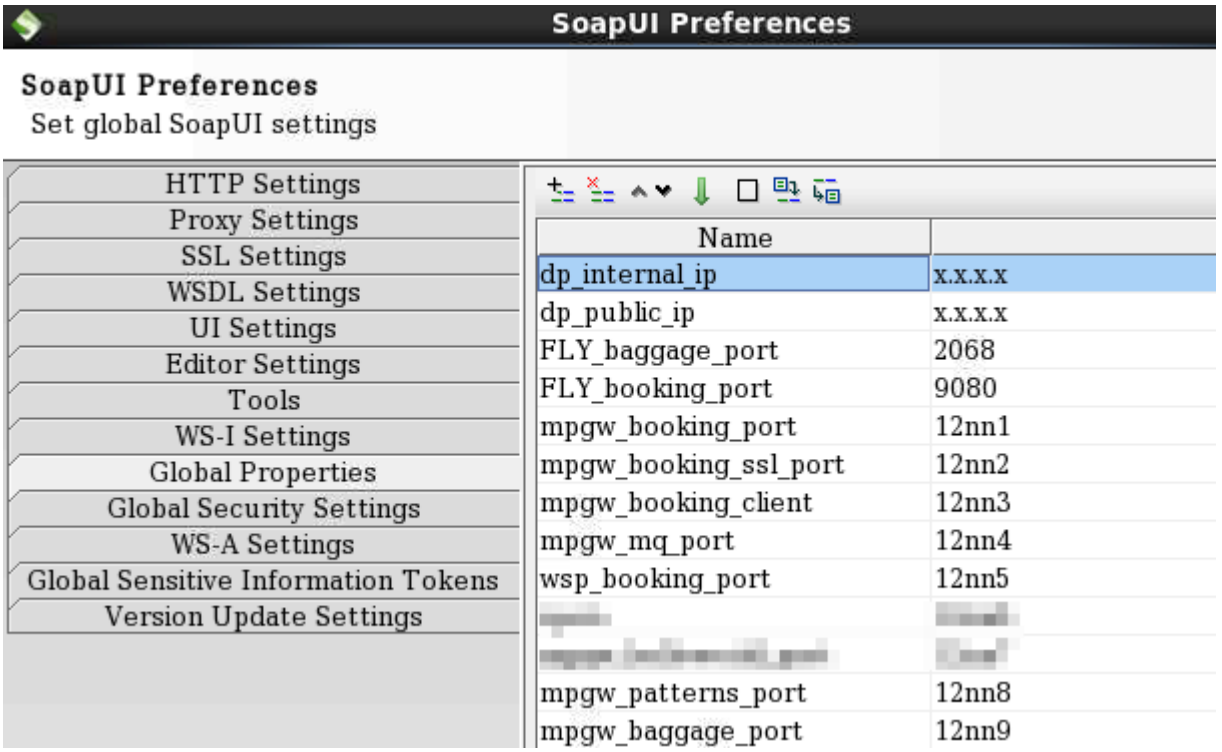
__ e. Replace the value (`x.x.x.x` or `1.2.3.4`) with the literal value of the ***<dp_public_ip>*** address in the cell. That is, replace 1.2.3.4 with the IP address of the DataPower appliance that is being used for your class.

__ f. Press Enter while the cursor is in the cell. This action registers the new value.

__ g. Double-click the **Value** cell for the **mpgw_booking_port** property.

__ h. Replace "`nn`" with your appropriate student number. For example, if you are `student 01`, the value for `mpgw_booking_port` of `12nn1` is updated to `12011`.

__ i. Press Enter while the cursor is in the **Value** cell. This action registers the new value.



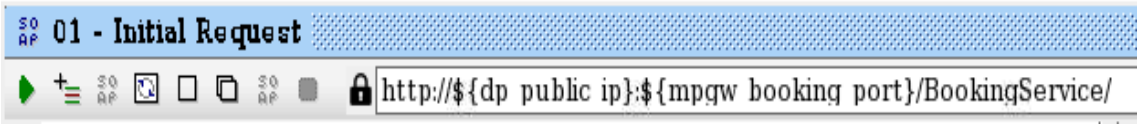
__ j. Repeat the previous steps g - i for the remaining values.

__ k. Click **OK**.

__ l. Click **File > Save Preferences**.

SoapUI is now configured for all exercises in this course. The SOAP message that is sent to DataPower when using SoapUI references these variables. No further SoapUI configuration is required, unless stated in the specific exercise.

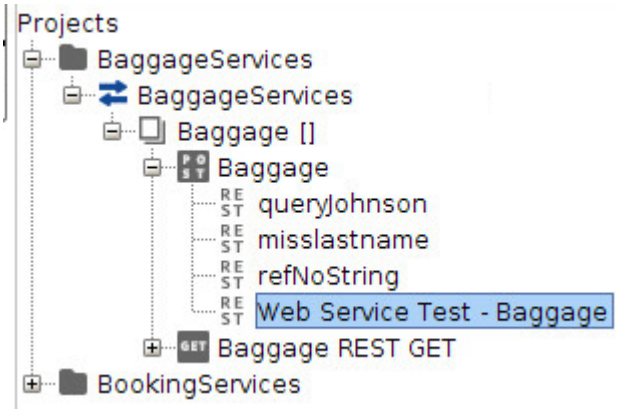
When SoapUI recognizes the `dp_public_ip` reference in a request (“`${dp_public_ip}` “), it substitutes the correct IP address into the URL.



Part 2: Confirm that the Booking and Baggage web services are up

Test the Booking web service and the Baggage web service. The following steps ensure that the back-end web services are operational. In addition to testing the availability of the web service, it is also a useful troubleshooting technique to verify network connectivity to the back-end web service.

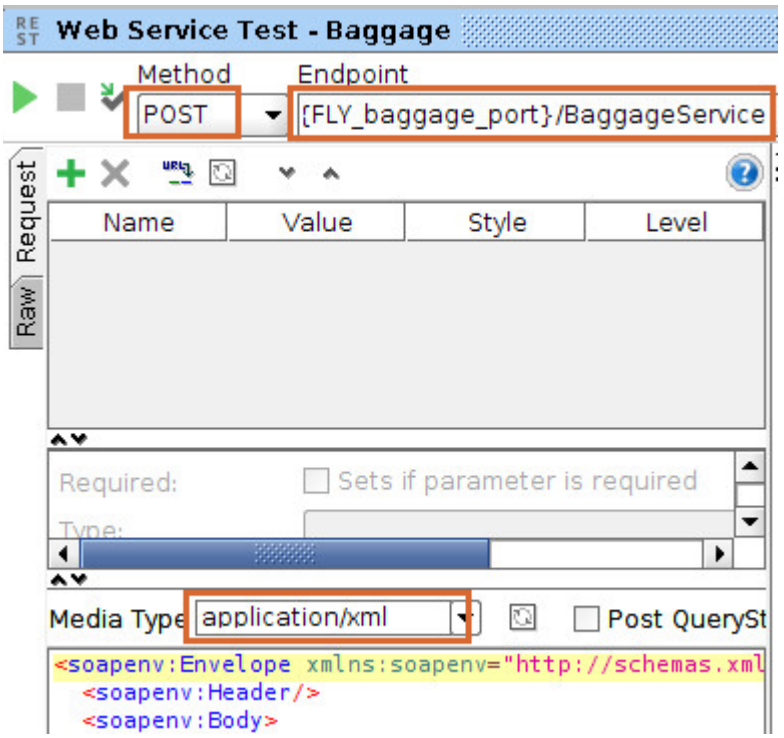
__ 1. In the project tree, expand the **BaggageServices** project until **Web Service Test - Baggage** is visible.



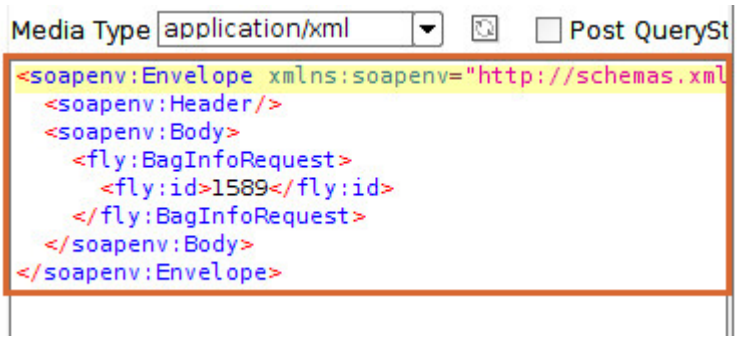
__ 2. Double-click **Web Service Test - Baggage** to open the request window. If a double-click does not work, right-click the request and click **Show Request Editor**.

__ 3. Ensure that the following information is preconfigured in the request message.

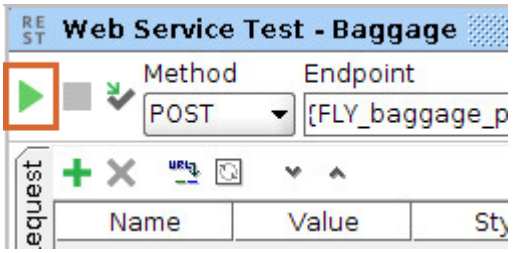
- Method: **POST**
- Endpoint: **http://${dp_internal_ip}:${FLY_baggage_port}/BaggageService**
- Media Type: **application/xml**



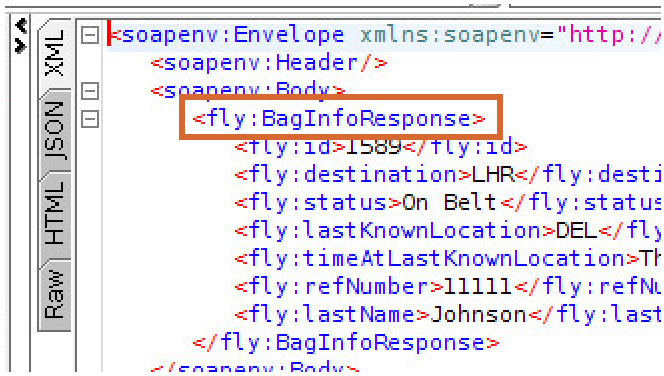
- Soap message:



__ 4. Click the green **Submit** arrow to send the request message directly to the **BaggageService** web service for FLY airlines.



__ 5. Confirm that a successful **BagInfoResponse** response is returned in the response tab.



Important

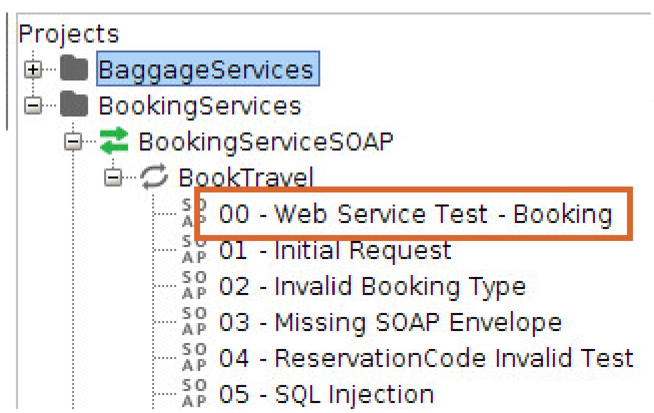
If you do not get the correct response, there can be several reasons for the failure:

- The variables that are entered in SoapUI General Preferences are not installed on the DataPower appliance.
- The DataPower appliance is unreachable from your student image due to some network connectivity issue.

Verify that you entered the correct values for the SoapUI variables. If the values are correct, escalate for assistance.

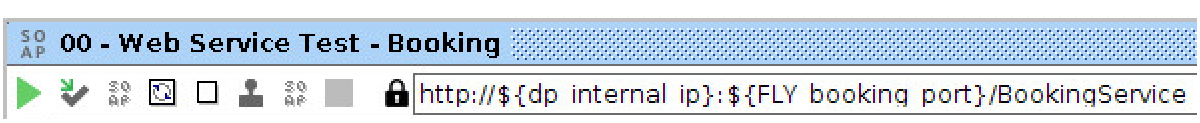
__ 6. Close the **Web Service Test - Baggage** window.

__ 7. In the project tree, expand the **BookingServices** project until **00 – Web Service Test - Booking** is visible.



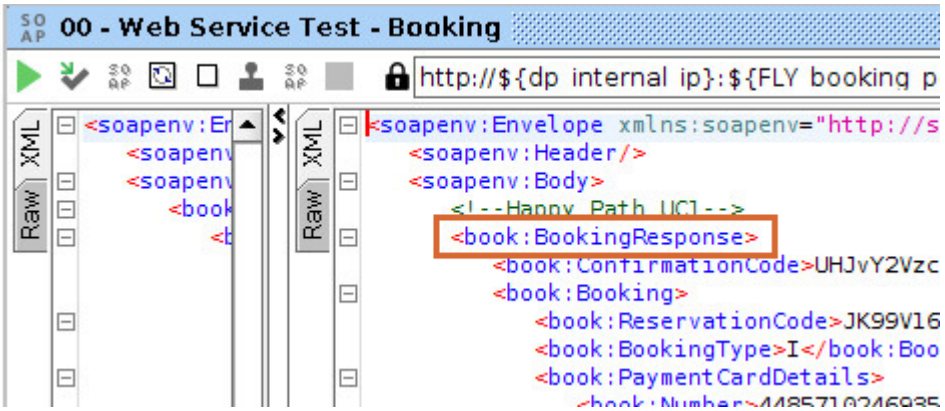
__ 8. Double-click **00 – Web Service Test - Booking** to open the request window. If a double-click does not work, right-click the request and click **Show Request Editor**.

__ 9. Confirm that the URL address field contains:
http://${dp_internal_ip}:${FLY_booking_port}>/BookingService



__ 10. Click the green **Submit** arrow that is to the left of the URL address field to send the SOAP request test message directly to the FLY Airlines Booking web service.

__ 11. If everything worked properly, you should see <book:BookingResponse> xml tree in the Response tab.



Important

If you do not get the correct response, there can be several causes:

- The variables that are entered in SoapUI General Preferences are wrong.
- The FLYService domain that contains the web services is not installed on the DataPower appliance.
- The DataPower appliance is unreachable from your student image due to a network connectivity issue.

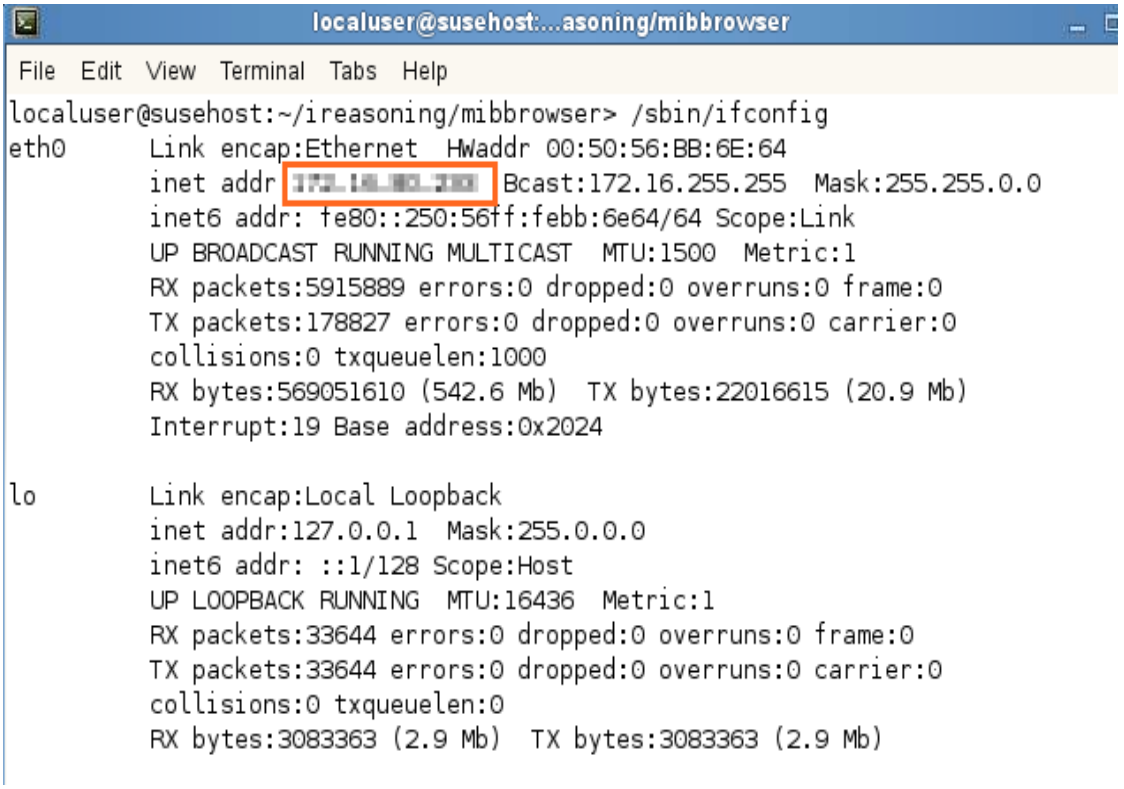
Verify that the values that you entered for the SoapUI variables are correct. If you still have problems, you must contact whatever support you have for the class.

__ 12. Close the **00 - Web Service Test - Booking** window.

Part 3: Identify the student image IP address

On Linux, you can discover the IP address by running the `ifconfig` command. Open a terminal window. The terminal window is available from the icon on the desktop. From within a terminal window, run the `ifconfig` command that includes the full path, as follows:

```
> /sbin/ifconfig
```



When the IP address of the local student image is obtained, update the information in table B1 for the variable *<image_ip>*.

Close the terminal window.

Part 4: Port and variable Table values

If you want to have a single reference for all variables that are used in this course, the following table is supplied. You might want to tear these two pages out of your book, or print both pages if you have a PDF, as a quick reference point.

__ 1. Complete the following table with the values supplied by the instructor.

Table B-1. Developers course variable and port assignments table

Object	**Value (default)**
Lab files location	
<lab_files> Location of student lab files for this course	/usr/labfiles/dp
Student information	
<nn>	
<student*nn*>	
<student*nn*_domain>	
<student*nn*_password>	student*<nn>*
<student*nn*_updated_password>	

<image_ip> IP address of the student image	
Logins that are not DataPower	
<linux_user>	localuser
<linux_user_password>	passw0rd
<linux_root_user>	root
<linux_root_password>	passw0rd
DataPower information	
<dp_public_ip> IP address of the public services on the appliance	
<dp_internal_ip> IP address of the DataPower appliance development and administrative functions	
<dp_WebGUI_port> Port number for the WebGUI	9090
<dp_its_http_port> Port for the HTTP interface to the Interoperability Test Service	9990
<dp_FLY_baggage_port>	2068
<dp_FLY_booking_port>	9080
Server information	
<SoapUI_keystores>	/usr/labfiles/dp/WSSecurity/Client.jks
<SoapUI_keystores_password>	myjkspw
<ldap_password>	passw0rd
<ldap_server_port> Port number for the LDAP administration console	9080
<ldap_server_root_dir>	/opt/ibm/ldap/V6.3/bin
<ldap_user_name>	cn=root
<http_server_port>	80
<http_server_root_dir>	/opt/IBM/HTTPServer/
<logger_app_port>	1112
Student-built DataPower services	

<mpgw_booking_port>	12nn1
<mpgw_booking_ssl_port>	12nn2
<mpgw_ssl_booking_port>	12nn3
<mpgw_mq_port>	12nn4
<wsp_booking_port>	12nn5
<mpgw_helloworld_port>	12nn7
<mpgw_patterns_port>	12nn8
<mpgw_baggage_port>	12nn9

End of appendix

IBM Training