**Chapter 8: XML Firewall**

Add a note hereNow that you have a good understanding of what a DataPower service is and the different service types available, it is time to create our first DataPower service. This chapter takes you step by step through the creation of an XML Firewall service and discusses many of the service parameters and options available. Using simple yet common use cases, these examples give you a good understanding of the capabilities of the XML Firewall service as well as basic configuration of the Processing Policy and actions contained within it.

**Add a note here****XML Firewall Overview**

Add a note hereAn XML Firewall (XMLFW) is a service type that can be used to process XML and non-XML traffic sent over the HTTP(S) protocol. It is considered the simplest of the three main service types to configure because it can have one single IP:port that it can listen on for traffic and can only communicate with the client and the backend server over HTTP(S).[[1](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=277781658&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ftn.ch08fn01)] Although it is limited to one listening IP address:port and one protocol, it can provide the same level of threat protection and multistep processing capabilities as the Multi-Protocol Gateway service type. The one additional feature that is offered by the XML Firewall is the ability to easily configure it as a loopback proxy.[[2](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=277781658&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ftn.ch08fn02)] As discussed in [Chapter 6](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=755#755), [“Introduction to DataPower Services,”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=755#755) a loopback proxy does not forward requests to a backend server, but generates and returns the response back to the client itself. As a request enters the XMLFW, it is processed within the service, where a response is also generated and sent back to the client. Because of this capability, the XMLFW is often used as a loopback proxy when testing or debugging other services when the backend server is not available. The XMLFW service type is also capable of proxying backend services as it provides threat protection and multistep processing capabilities. We will demonstrate these capabilities in the examples that follow.

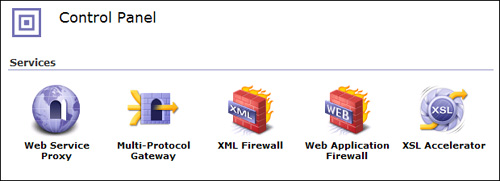
Add a note here[[1](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=277781658&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ch08fn01)]It is possible to configure the XMLFW to communicate with other protocols by using custom programming, which is discussed in [Chapter 23](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4198#4198), [“Programming within the DataPower Environment.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4198#4198)

Add a note here[[2](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=277781658&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ch08fn02)]The Multi-Protocol Gateway and Web Service Proxy services can also act as loopback proxies; however, it is not as easily configured as in the XMLFW service. This is discussed in [Chapter 9](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1276#1276), [“Multi-Protocol Gateway.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1276#1276)

## Creating an XMLFW

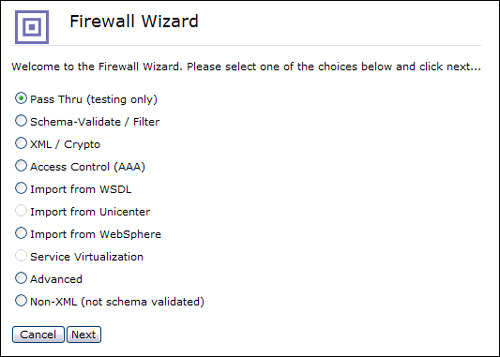
Add a note hereTo begin creating the XMLFW service, you must first log in to the DataPower WebGUI as it is the interface used for the creation of the service. Be sure to navigate to the domain in which you want to configure the XMLFW service. Remember, the default domain is only for device and user administration. Never create services within the default domain!

Add a note hereAfter logging in, you are presented with the home page (Control Panel) that shows three sections: Services, Monitoring and Troubleshooting, and Files and Administration. We are interested in the top section of the main page, labeled Services. To create or modify an XMLFW service, click the XML Firewall icon as shown in Figure 8-1.

[](javascript:PopImage('IMG_125','http://images.books24x7.com/bookimages/id_30903/08fig01_alt.jpg','620','224'))  
Add a note hereFigure 8-1: Selecting the XMLFW service type from the Control Panel.

Add a note hereWhen clicking the XML Firewall icon from the Control Panel, a screen is presented showing all XML Firewall services configured within the current domain. Also, notice there is an Add Wizard button on this screen. This option is used to configure a new XMLFW service with the guidance of a wizard. For our first XMLFW service example, we will take advantage of this wizard-driven approach as it guides us through the process of producing a base XMLFW that can be modified after it is created.

Add a note hereThe first wizard screen presents a list of options with radio buttons as illustrated in Figure 8-2. This provides a basic starting point for the wizard by indicating the primary function that the service will perform. For example, you can indicate on this screen that the service should only accept XML documents in the request and should schema validate them. Or you can indicate that the request will not be schema validated and might not be XML. This merely gives the wizard some idea of the intent for the service, so it can guide you through its creation. Table 8-1 provides a brief description of each of these options.

[](javascript:PopImage('IMG_126','http://images.books24x7.com/bookimages/id_30903/08fig02_alt.jpg','533','381'))  
Add a note hereFigure 8-2: Initial XMLFW Wizard screen.

| Add a note hereTable 8-1: Initial XMLFW Wizard Screen Options  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=863242058&objectid=ch08table01&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=863242058&objectid=ch08table01&objecttype=spreadsheet) | |
| --- | --- |
| **Add a note hereOption** | **Add a note hereDescription** |
| Add a note herePass Thru (testing only) | Add a note hereCreates an XMLFW with a request type of SOAP and a default request (Client to Server) rule consisting of a Match and a Result action. This does not create a rule with a request type of Pass Thru as you might expect. |
| Add a note hereSchema-Validate/Filter | Add a note hereCreates an XMLFW that contains a request and response processing rule with processing actions to schema validate the request and response messages using the schema file selected in subsequent steps. Optionally, this choice allows the inclusion of a Filter action within the request/response rules. |
| Add a note hereXML / Crypto | Add a note hereCreates an XMLFW that will verify and/or decrypt request and/or response messages using the certificate specified in the creation of the service |
| Add a note hereAccess Control (AAA) | Add a note hereCreates an XMLFW with a AAA policy that will be used to authenticate, authorize, and audit all requests based on the information provided during the creation of the service |
| Add a note hereImport from WSDL | Add a note hereCreates an XMLFW using the information provided in a WSDL. This is not a recommended practice as a WSP service type should be used instead. |
| Add a note hereImport from WebSphere | Add a note hereCreates an XMLFW for a Web service deployed on a WebSphere Application Server (WAS). Three files from WAS are required to be uploaded to the DataPower device to provide the required information—Binding Information file, Extension Information file, and the WSDL. |
| Add a note hereAdvanced | Add a note hereCreates an empty XMLFW, bypassing the wizard. The service created must be configured manually. |
| Add a note hereNon-XML (not schema validated) | Add a note hereCreates an XMLFW where the request document is not required to be XML, therefore is not schema validated. |

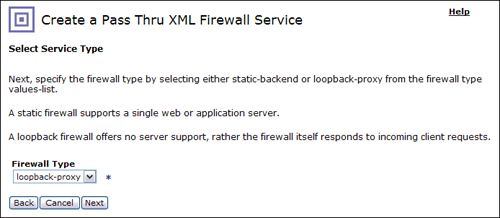
Add a note hereA simple example of an XMLFW service might be used to create a service that echoes the request document back to the client in the response. Consider this the Hello World of DataPower services. In this case, because our service simply echoes the request message back to the client in the response, we select the Pass Thru (testing only) option on this screen as shown in Figure 8-2. The second screen presented by the wizard is an easy one. This screen requires a name for the new service we are creating. This should be a descriptive name that has some relevance to what the service does. Because our example simply echoes the request back to the client, we will name this service Echo as shown in Figure 8-3.

[](javascript:PopImage('IMG_127','http://images.books24x7.com/bookimages/id_30903/08fig03_alt.jpg','674','232'))  
Add a note hereFigure 8-3: Provide a name for the new service.

### Tip: Pass Thru Request Type

Add a note hereChoosing the Pass Thru (testing only) option when configuring an XMLFW via the wizard does not produce a service with a Request Type of Pass Thru. This must be done in the service configuration screen itself. When the Request Type is set to Pass Thru, a Processing Policy is not invoked for requests and no threat protection is provided. Because of this, careful consideration should be given to using an XMLFW service with a request type of Pass Thru.

Add a note hereThe next screen asks for the firewall type. This indicates whether the firewall has a static backend server or will act as a loopback proxy. A static backend requires a backend server address that all requests are forwarded to; a loopback-proxy does not require a backend server, as it generates the response within the service itself. Although the XMLFW can have a dynamic backend, this is not an option on this screen. It can be selected only after the service is created; this will be demonstrated in an example later in this chapter. In our example, we create a loopback proxy, so we select the loopback-proxy option from the drop-down for the firewall type as shown in Figure 8-4.

[](javascript:PopImage('IMG_128','http://images.books24x7.com/bookimages/id_30903/08fig04_alt.jpg','694','302'))  
Add a note hereFigure 8-4: Selecting the loopback-proxy option.

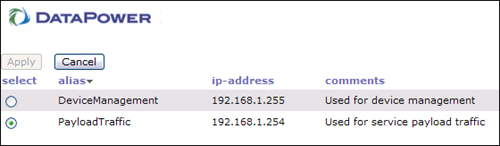
Add a note hereThe next, and final, step in creating the service is to assign the IP address and the port that the service listens on. This step requires some careful consideration as the IP:port combination must be unique within the entire device (not just the domain). On this screen, there is a field for the Device Address and a field for the Device Port.

Add a note hereThe Device Address represents the IP address of an actual Ethernet interface on the device. This can be represented in one of three ways. The first way is the default value that is populated when this screen is presented. Instead of an IP address, there is a series of four zeros separated by periods (0.0.0.0). This indicates that the service can receive requests on any of the configured Ethernet interfaces on the device. This is a highly discouraged practice as the payload traffic should always be segregated from the device management traffic, which should be dedicated to one of the four interfaces.

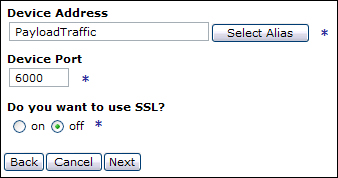
Add a note hereThe second method of entering the IP address that the service listens on is by simply entering the actual IP address of the Ethernet interface. Although this is more acceptable than listening on all interfaces, it hard codes the IP address in the service configuration itself, which presents migration issues when migrating the service from one environment to another (e.g. development environment to test environment). This topic is addressed in detail in [Chapter 15](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2720#2720), [“Build and Deploy Techniques.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2720#2720)

Add a note hereThe last, and recommended, method for providing the IP address is to use a host alias. A host alias is an abstraction of the IP address of an Ethernet interface and is configured within the default domain. This method provides a more seamless migration from environment to environment as the IP address is not embedded in the service configuration; however it will need to be defined on each device within the default domain. The benefits and configuration of a host alias are discussed in detail in [Chapter 2](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=145#145), [“DataPower Quick Tour and Setup,”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=145#145) and [Chapter 15](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2720#2720).

Add a note hereFor this example we use a host alias that we have configured named PayloadTraffic that resolves to a specific IP address on the device. Clicking the Host Alias button next to the Device Address field presents a list of configured host aliases where we select PayloadTraffic as shown in Figure 8-5.

[](javascript:PopImage('IMG_129','http://images.books24x7.com/bookimages/id_30903/08fig05_alt.jpg','560','163'))  
Add a note hereFigure 8-5: Select a host alias.

Add a note hereNow that we have configured the IP address that the device accepts traffic on, it is also necessary to enter the port that the service listens on. The default port listed is the next available port on the device starting from 2048 and counting up. You can choose to leave the default port here or enter a new port that is not already in use on your device for the same interface. It is important to assign and keep track of all ports being used on the device to avoid port collisions. For our example, we entered port 6000 in this field. Figure 8-6 shows the final Device Address and Device Port configuration for our Echo service. Notice that you can indicate on this screen whether the request will be sent over SSL.

  
Add a note hereFigure 8-6: Device Address and Port configuration.

Add a note hereNow that all the required information has been entered for our new service, the next, and final screen, presents a summary of the service that will be created. Here we can confirm and commit all our configuration choices. Congratulations! We just created our first DataPower service. You can now see the objects created, as well as their current status as shown in Figure 8-7. All these objects should have a status of up, indicating they are active.

[](javascript:PopImage('IMG_131','http://images.books24x7.com/bookimages/id_30903/08fig07.jpg','485','373'))  
Add a note hereFigure 8-7: Service objects and status.

Add a note hereThis section described how to create an XMLFW using the built-in wizard. This was demonstrated by providing a simple example of creating a service that echoes the request document back to the client in the response. Although this simple example was chosen for demonstration purposes, it can be useful when testing and debugging other services by providing this Echo service URL as the backend server where a response is required.

### Add a note hereTesting the Service

Add a note hereAfter creating a DataPower service, it is likely that you will want to test it to be sure that you get the expected results. This test involves sending a request document to the IP address:port combination on which the service is listening. There are several methods and utilities that can be used to accomplish this, which are discussed in [Chapter 27](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4747#4747), [“Problem Determination Using Built-In Tools”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4747#4747) and [Chapter 28](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4971#4971), [“Problem Determination Using External Tools.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4971#4971) One simple way to do this is to use the built-in testing tool that DataPower provides to send a request document to a service and view the results.

Add a note hereThis utility is accessed from the left-side menu of the WebGUI under Administration→Debug→Send a Test Message. Clicking this link brings up a screen that allows you to enter a test message, as well as request headers that you want to send to your service.

Add a note hereLet’s utilize this feature to test our Echo XMLFW created in the previous example. At the test message screen, enter the URL of the service to which you will be sending a test message. If we are testing the Echo service, we enter http://<ip address>:<port>, where the IP address is the address that the service is listening on and the port is the port that the service is listening on. In our example, the IP address would be the address mapped to the host alias specified in the service configuration and port 6000.

Add a note hereAfter entering the URL that the request will be sent to, we can enter the actual request document in the Request Body text box. For our example, we will send the SOAP document shown in Listing 8-1. Because the Echo service does not expect any specific request header, we will leave this section blank.

Add a note hereListing 8-1: Sample XML Request Document

Add a note here<?xml version='1.0' ?>

<SOAP-ENV:Envelope

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body>

<book>

<name>Moby Dick</name>

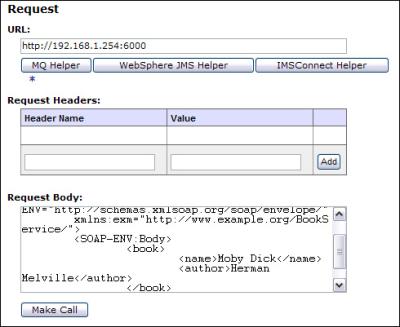
<author>Herman Melville</author>

</book>

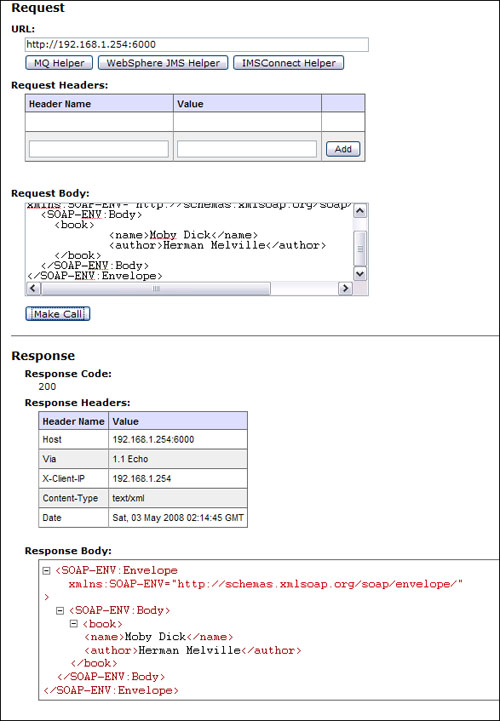
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

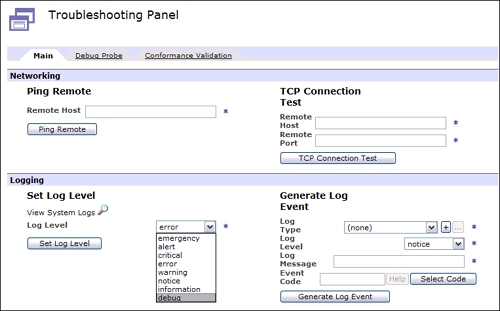
Add a note hereAfter this information is entered as shown in Figure 8-8, clicking the Make Call button sends the request to the Echo service.

[](javascript:PopImage('IMG_132','http://images.books24x7.com/bookimages/id_30903/08fig08.jpg','490','401'))  
Add a note hereFigure 8-8: Send the test request.

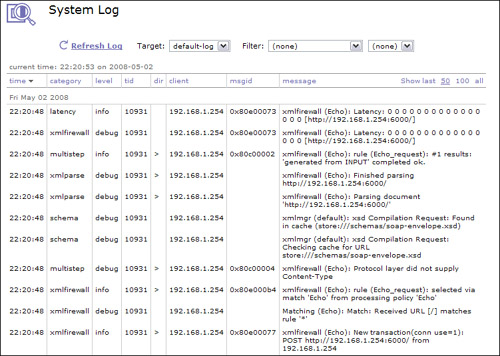
Add a note hereAfter the request is sent, the response from the Echo service is displayed in the Response section of the screen. Because the Echo service simply echoes the request document back to the client, the request body is identical to the response body as shown in Figure 8-9. In addition to the response body, you will notice that all response headers and the return code are also displayed.

[](javascript:PopImage('IMG_133','http://images.books24x7.com/bookimages/id_30903/08fig09_alt.jpg','587','847'))  
Add a note hereFigure 8-9: Response from Echo service.

Add a note hereTo show that the request actually hit the service and the service generated a response, we can take a look at the DataPower logs. If you were to look at the logs after sending the previous test request, you would most likely not see an entry for this request. This is because by default, DataPower is set only to log messages with a priority/severity of “error” or greater. To see our service generate messages that are written to the DataPower log, we can set the log level to debug. This setting is found at Control Panel→Troubleshooting→Set Log Level. Here we can set the log level to log all messages by selecting debug from the drop-down and clicking the Set Log Level button as shown in Figure 8-10. Note that this level of logging is recommended only in development environments because it logs a significant amount of data that would not be desirable in a production environment.

[](javascript:PopImage('IMG_134','http://images.books24x7.com/bookimages/id_30903/08fig10_alt.jpg','763','474'))  
Add a note hereFigure 8-10: Set log level.

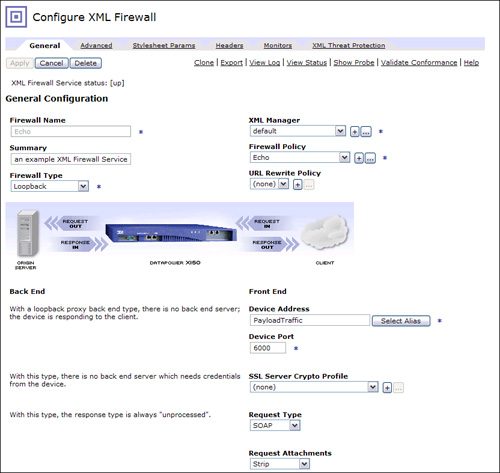
Add a note hereAfter confirming your selection, the log level is now set to log all events within the current domain. We can now submit the test message to our Echo service again and view the logs. The logs can be found at Control Panel→View Logs. You can see in Figure 8-11 that the request did hit our Echo XMLFW.

[](javascript:PopImage('IMG_135','http://images.books24x7.com/bookimages/id_30903/08fig11_alt.jpg','751','535'))  
Add a note hereFigure 8-11: Echo service log entries.

### Add a note hereA Quick Tour of an XMLFW

Add a note hereIn the previous section, we demonstrated how to create an XMLFW service using the built-in wizard as well as how to test it using the built-in test tool. However, what did we actually create here? Because we used the wizard driven approach, we never actually saw what was built and how. We actually created an XMLFW service with many different options and service parameters that can be viewed and edited via the WebGUI. In this section, we will look at the major components and parameters that make up an XMLFW service.

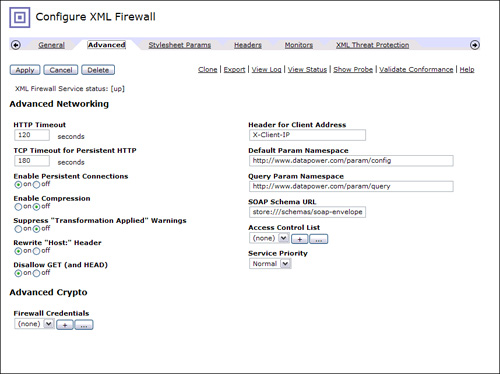
Add a note hereYou can access an XMLFW service by navigating to Control Panel→XML Firewall and clicking on the service name. This will bring up the main configuration page for the service as shown in Figure 8-12. You can see that there are several tabs at the top of the page and you are viewing the General tab. As the name suggests, this tab contains all the general information about the XMLFW. You might recognize the XMLFW shown in Figure 8-12 as it is the Echo service we configured in our previous example. Notice the Firewall Type is Loopback and the Device Address and Device Port are as we specified during the creation of the service. All other configuration parameters are set to the default values.

[](javascript:PopImage('IMG_136','http://images.books24x7.com/bookimages/id_30903/08fig12_alt.jpg','832','787'))  
Add a note hereFigure 8-12: Configure XML Firewall—General.

Add a note hereNotice on the bottom half of the screen in this tab, the Front End (bottom right) and Back End (bottom left) parameters are separate. In Figure 8-12, you do not see any Back End parameters as this is our Echo service, which is a loopback firewall type so it will not have a backend server defined. Let’s now take a look at the major service configuration parameters found within this General tab:

* Add a note here**Firewall Type—** Define the type of backend used for this service. Three possible options exist for this parameter:
  + Add a note here**Loopback—** Used when there is no backend for the service. It is the responsibility of the service to produce a response back to the client.
  + Add a note here**Static-Backend—** Used when there is a known backend address that all requests will be forwarded to. When this option is selected, a backend URL and port must be provided.
  + Add a note here**Dynamic-Backend—** Indicates that the backend server address is determined at runtime. When this option is selected, no backend URL is provided in the service configuration. It is then the responsibility of the Processing Policy to assign this at runtime.
* Add a note here**XML Manager—** The XML manager used for this service. A detailed description of the XML Manager can be found in [Chapter 7](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=868#868), [“Introduction to Services Configuration.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=868#868)
* Add a note here**Firewall Policy—** The Processing Policy used for this service. This is perhaps the most important part of the XML Firewall configuration and is likely where you’ll spend most of your service development time.
* Add a note here**URL Rewrite Policy—** The URL Rewrite Policy that is optionally invoked for each request. The URL Rewrite Policy can rewrite all or part of a URL before the Processing Policy processes the incoming request.
* Add a note here**Device Address—** The IP address on which the service should accept requests. This IP address must be an IP address, or secondary IP address of a physical Ethernet interface on the device. A configured host alias can also be used here to reference a device interface.
* Add a note here**Device Port—** The port that the service uses to listen for requests. The IP address and port combination must not be used by any other services on the same DataPower device.
* Add a note here**SSL Server Crypto Profile—** The SSL Crypto Profile used for frontside SSL connections. This object defines all the SSL information required to establish the SSL connection.
* Add a note here**Request Type—** Defines the type of request documents accepted by this service. The valid values are
  + Add a note here**SOAP—** All request documents sent to this service must be valid SOAP documents that will be validated against a SOAP schema for well-formedness. This is the default value for an XMLFW.
  + Add a note here**Non-XML—** The request might or might not be an XML document and is not schema validated.
  + Add a note here**Pass-Thru—** The request is not validated and is not processed by the Processing Policy defined within the service. No threat protection is provided by the service when this Request Type is selected.
  + Add a note here**XML—** The request document must be a valid, well formed XML document.
* Add a note here**Request Attachments—** When the request type is SOAP or XML, this setting determines what action to take if the request contains attachments. The valid values for this parameter are
  + Add a note here**Strip—** The attachment(s) is stripped from the request and then the request continues to be processed by the service. This is the default setting.
  + Add a note here**Allow—** The request (and attachment) is accepted and processed by the service. The attachment is also available for processing by the service.
  + Add a note here**Reject—** The entire request is rejected by the service if an attachment is sent.
  + Add a note here**Streaming—** Allows the message to be processed in streaming mode where the attachments are available to the service for processing.
  + Add a note here**Unprocessed—** The request is allowed with the attachments, however the attachments are not available to the service for processing.
* Add a note here**Backend—** When the Firewall Type is set to Static, parameters for the backend server become visible in this space. These parameters are identical to the Front End parameters; however they are applied to the back end connection. This is discussed and demonstrated later in this chapter.

Add a note hereIn addition to the basic configuration parameters found in the General tab, the Advanced tab contains many advanced options and parameters that can be set as shown in Figure 8-13.

[](javascript:PopImage('IMG_137','http://images.books24x7.com/bookimages/id_30903/08fig13_alt.jpg','858','641'))  
Add a note hereFigure 8-13: Configure XML Firewall—Advanced.

Add a note hereMany of the settings within the Advanced tab will be left at the default values; however it is important to understand what each setting represents and how it can affect the processing within your service in the event that you might need to change the default behavior. Table 8-2 provides a brief description of each of the service configuration parameters found in this tab.

| Add a note hereTable 8-2: Advanced XMLFW Configuration Parameters  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=863242058&objectid=ch08table02&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=863242058&objectid=ch08table02&objecttype=spreadsheet) | |
| --- | --- |
| **Add a note hereParameter Name** | **Add a note hereDescription** |
| Add a note hereHTTP Timeout | Add a note hereSpecifies the duration in seconds that the service maintains an idle HTTP connection with a client or server. The number of seconds entered must be between 3 and 7200. The default is 120 seconds. |
| Add a note hereTCP Timeout for Persistent HTTP | Add a note hereSpecifies the duration in seconds that the service maintains an idle TCP connection with a client or server. The number of seconds entered must be between 0 and 7200. The default is 180 seconds. |
| Add a note hereEnable Persistent Connections | Add a note hereEnables persistent connections to clients and servers that this service communicates with. When this setting is On, the service keeps a single HTTP connection open to a client or server for multiple URL requests. The connection is left open for the duration specified in the TCP Timeout for Persistent HTTP setting. The default for this parameter is On. |
| Add a note hereEnable Compression | Add a note hereEnables GZIP compression negotiation with an HTTP peer. |
| Add a note hereSuppress “Transformation Applied” Warnings | Add a note hereAs per RFC2616, if the service changes the content coding or media type, a warning header should be sent in the response. If this setting is Off, this warning is sent. If it is On, this warning is suppressed. The default for this setting is Off. |
| Add a note hereRewrite “Host” Header | Add a note hereAs a request comes into this service, a Host header is sent along with it. This header is set by the client and contains the address and method that the client used to access the service. When this parameter is set to On, the service rewrites this header to reflect the backend address and method that the request is forwarded to by the XMLFW. When set to Off, this header is not rewritten. The default for this parameter is On, indicating the header is rewritten by the service. This might not be desirable when the backend server is responding to the client with a redirect location as the redirect location could be derived using the Host header. This causes the redirect to bypass the DataPower service and go directly to the backend server. |
| Add a note hereDisallow GET (and HEAD) | Add a note hereWhen set to On, the service does not allow HTTP GET or HTTP HEAD requests. When set to Off, these requests are allowed. The default for this parameter is On. |
| Add a note hereHeader for Client Address | Add a note hereDataPower inserts an HTTP header in the request containing the client IP Address. This parameter specifies the name of the header. |
| Add a note hereDefault Param Namespace | Add a note hereThe default namespace used when passing parameters to a stylesheet via the WebGUI if the parameter is defined without a namespace or without explicitly specifying a null namespace. |
| Add a note hereQuery Param Namespace | Add a note hereThe default namespace used when passing parameters to the service via a URL query string. |
| Add a note hereSOAP Schema URL | Add a note hereThe URL for the schema the service uses to validate SOAP documents. |
| Add a note hereAccess Control List | Add a note hereAn access control list that can be created to restrict access to the service by IP address or range. |
| Add a note hereService Priority | Add a note hereAssigns a priority to the service of High, Medium, or Low. This priority is used to allocate resources across multiple services. |
| Add a note hereFirewall Credentials | Add a note hereA list of keys and certificates made available to the service. If none are specified, the service has access to all keys and certificates that are defined within the domain. |

### Tip: Setting the Rewrite “Host” Header

Add a note hereWhen the service is proxying a backend server that will respond with redirects and locations to the client, this setting should be set to Off. Because the server may use this header to specify the redirect location, you would not want this header changed to the actual backend address.

Add a note hereAll other tabs within the service configuration screen are not specific to the XMLFW service type; therefore they are covered in separate chapters throughout this book.

### Add a note hereProcessing Policy, Rules, and Actions

Add a note hereNow that we have created our first DataPower service, you might think that it wasn’t exciting. That’s because we created the simplest type of service that really doesn’t do much at all but echo back the request document to the client. You see, the real power in a DataPower service lies in the ability to apply many different types of processing to the request and response messages. This is configured via the Processing Policy within the service. This section discusses and demonstrates the creation of a Processing Policy, processing rules, and actions within an XMLFW service.

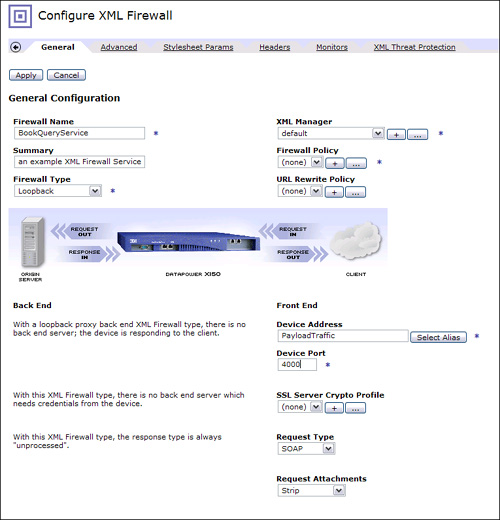
Add a note hereAs shown in the previous section, the Firewall Policy within an XMLFW can be created or accessed via the General tab on the Configure XML Firewall screen. When an XMLFW is created using the wizard, as in the previous example, a Firewall Policy object is automatically created for you. If you used the Add Advanced option to create your service, you will have to create one yourself. To demonstrate this, let’s create another example XMLFW service.

Add a note hereIn this example, we will build another XMLFW that acts as a loopback proxy as in the first example; however, this example actually provides a valid response to the request. This service accepts a SOAP request document that provides a book name and book author in the SOAP body and returns information about the book requested.

Add a note hereBecause we already know how to use the wizard for creating an XMLFW, let’s try to create one using the Add Advanced option. This can be found next to the Add Wizard button after navigating to the Control Panel→XML Firewall screen. After clicking this button, you are taken directly to the XMLFW Configuration screen within the General tab where all the required information must be entered.

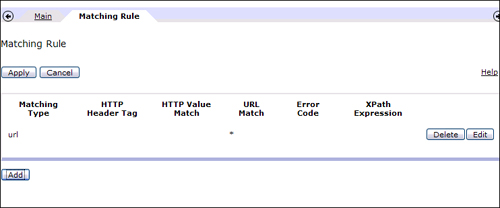
Add a note hereYou enter the same kind of information here as you did when creating the XMLFW via the wizard; however, you are not prompted with these questions. On the General tab, you enter the Firewall name and summary (optional), the Firewall Type (with backend information if applicable), and all the frontend information.

Add a note hereIn this example, let’s assume that the real backend service is not yet available, which is why we will create an XMLFW that will generate the response to the client. We named this service BookQueryService. As seen in Figure 8-14, we are creating a service that is a loopback firewall as the service itself provides a response and listens on port 4000 on the PayloadTraffic host alias.

[](javascript:PopImage('IMG_138','http://images.books24x7.com/bookimages/id_30903/08fig14_alt.jpg','765','795'))  
Add a note hereFigure 8-14: BookQueryService configuration.

Add a note hereNow that we have all the information required on the General tab, we can create a Firewall Policy. As with all objects, to edit an existing object you click the ... button next to the Firewall Policy name and to create a new object, you click the + button. Because we do not have a Firewall Policy object already created, we need to click the + to create a new one. This brings us to the policy editor screen. To start, the new policy requires a name. As a general practice, we name our policy with the same name as our service. In this case we name it BookQueryService. We now have a Processing Policy; however it is quite useless unless we add some rules and actions to it.

Add a note hereFor our example, let’s create a new request (client to server) rule to our new policy. When a new rule is created, a match rule is automatically added to it that needs to be configured. Because this is the only request rule in our policy and it should act on all requests to the service, we create a match rule that matches on all URLs as shown in Figure 8-15.

[](javascript:PopImage('IMG_139','http://images.books24x7.com/bookimages/id_30903/08fig15_alt.jpg','742','309'))  
Add a note hereFigure 8-15: Match rule for all URLs.

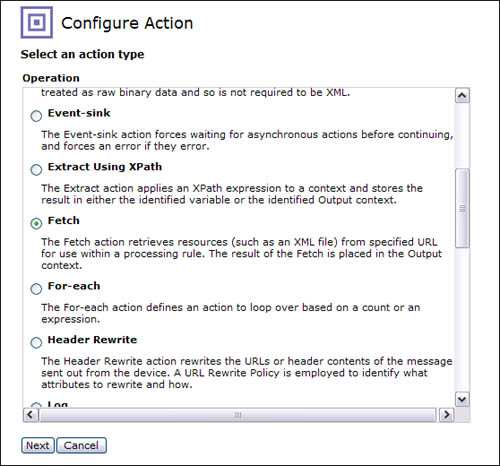
Add a note hereBecause we already know how to work with the Policy editor, let’s now take the time to explore two specific actions that can be added to a processing rule.

#### Fetch Action

Add a note hereThe Fetch action can be a useful action because it can fetch a document from a specified location using a URL provided. The URL can reference a remote location or it can reference the DataPower file system itself. This action retrieves the document from the URL specified and outputs this document to the output context specified in its configuration. For example, if the output context is OUTPUT, the document retrieved will be sent to the output of the rule. The output context of the Fetch action can also be the same as the input context of the next action. If this is the case, the document fetched will be passed directly to the next action for processing.

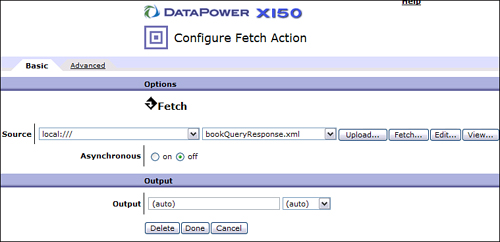
Add a note hereIn our example we said that we will return a valid response back to the client. To do this we use the Fetch action to retrieve the response document from the DataPower file system, passing it back to the client in the response.

Add a note hereYou might notice when configuring your processing rules that there is no action named Fetch in the array of actions listed above the policy line. As discussed in [Chapter 7](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=868#868), “Introduction to Services Configuration,” this, and many other actions, are grouped into a single icon called Advanced in the Policy editor and can be identified by the diamond-shaped icon labeled Advanced. When this action is added to the policy line, it will be outlined in yellow, which means that you are required to configure it. The first step in configuring it is to select the actual action that you want to use or the action type. This is where we select the Fetch action as shown in Figure 8-16.

[](javascript:PopImage('IMG_140','http://images.books24x7.com/bookimages/id_30903/08fig16_alt.jpg','558','520'))  
Add a note hereFigure 8-16: Choosing the Fetch action.

Add a note hereAfter the Fetch action is selected, we are required to enter a URL in the Source field for the location of the document to be fetched. This URL can be constructed to reference any of the supported protocols on the DataPower device by using the applicable URL scheme.

Add a note hereIn our example, we reference a document named bookQueryResponse.xml in the local folder on the DataPower file system that was previously uploaded to the local directory on the device file system. Figure 8-17 shows this configured in the Fetch action configuration screen.

[](javascript:PopImage('IMG_141','http://images.books24x7.com/bookimages/id_30903/08fig17_alt.jpg','759','368'))  
Add a note hereFigure 8-17: Enter URL to fetch.

Add a note hereThe output of this action is the actual document that was fetched from the URL provided. You will notice after configuring this action that the Advanced action icon has been replaced by the Fetch icon on the policy line as shown in Figure 8-18.

[](javascript:PopImage('IMG_142','http://images.books24x7.com/bookimages/id_30903/08fig18_alt.jpg','704','151'))  
Add a note hereFigure 8-18: The Fetch icon.

#### Validate Action

Add a note hereIn most cases, you will be configuring an XML Firewall to process XML requests and responses. One common and useful function that a DataPower service can perform is not only to ensure that the XML is well formed, but also to validate the request and the response document to ensure that it is compliant with a specific schema. This then ensures that the backend service is receiving only valid, schema-compliant documents, freeing it up from the resource intensive process of validating the document itself.

Add a note hereTo accomplish this task, there is a specific action called Validate that validates the input document against a schema that can be specified at configuration time or dynamically determined at runtime by the input document itself.

Add a note hereThis action is shown in the actions displayed above the Policy editor and can be identified by the triangle-shaped icon with a check mark labeled Validate.

Add a note hereSo, let’s say that in BookQueryService we wanted to validate the request against a schema that has been uploaded to the DataPower file system. To do this, you simply add a Validate action to the Request rule created after the Match rule and before the Fetch action, as shown in Figure 8-19.

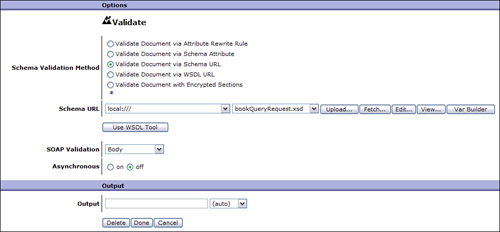
[](javascript:PopImage('IMG_143','http://images.books24x7.com/bookimages/id_30903/08fig19_alt.jpg','707','139'))  
Add a note hereFigure 8-19: BookQueryRequest rule.

Add a note hereAfter it is added to the policy line, this action must be configured by double-clicking the icon. When configuring this action, you must indicate how the input document will be validated, otherwise known as its schema validation method. The choices are

* Add a note here**Validate Document via Schema URL—** Uses the URL provided to access the schema that will be used to validate the input document
* Add a note here**Validate Document via Schema Attribute—** Uses the values of the xsi:schemaLocation and the xsi:noNamespaceSchemaLocation attributes within the input document to determine the location of the schema to be used to validate the input document
* Add a note here**Validate Document via the Attribute Rewrite Rule—** Uses the xsi:schemaLocation and the xsi:noNamespaceSchemaLocation attributes within the input document after a URL rewrite policy has been applied to the values to determine the location of the schema to be used to validate the input document
* Add a note here**Validate Document with Encrypted Sections—** Validates the document using a Schema Exception map, where some parts of the document may be encrypted
* Add a note here**Validate Document via WSDL URL—** Uses the schema within a WSDL specified to validate the input document

Add a note hereDepending on the schema validation method, other fields might be required in the configuration, such as the URL to access the schema or a WSDL file. After the Validate action is configured, it validates the request or response document based on the direction of the rule that it was added to.

Add a note hereFor our BookQueryService we know that we have a schema that we want to validate our request document against. In this case we would select the Validate Schema via Schema URL option and provide the URL to the .xsd file in the local file system. It is assumed that the schema file that we will reference for this example has already been uploaded to the device and is called bookQueryRequest.xsd. Figure 8-20 shows this configuration screen with a reference to this schema file.

[](javascript:PopImage('IMG_144','http://images.books24x7.com/bookimages/id_30903/08fig20_alt.jpg','975','452'))  
Add a note hereFigure 8-20: Request Validate action configuration.

Add a note hereIn addition to specifying the method to use to validate the input document, the Validate action also lets you specify what portion of the document will be validated in the event that the input document is a SOAP document. For example, you can choose only to validate the SOAP Body of the input document, or you can validate the entire SOAP Envelope. This is determined by the SOAP Validation field in the Validate action configuration screen shown in Figure 8-20. The options that can be selected for this field are as follows:

* Add a note here**Envelope—** Validate the entire input SOAP message, including the SOAP Envelope against the schema specified.
* Add a note here**Body—** Validate only the Body of the SOAP message against the schema specified. This is the default.
* Add a note here**Body or Detail—** Validate the Body of the SOAP message against the schema specified or validate the detail element if the document contains a SOAP Fault.
* Add a note here**Ignore Faults—** If the document contains a SOAP Fault, no validation is performed. If the SOAP document does not contain a SOAP Fault, the SOAP Body is validated.

Add a note hereIn our BookQueryService, the schema used to validate the request document does not include the SOAP Envelope so we leave the default as Body for the SOAP Validation parameter and click Done to finish the Validate action configuration. Clicking the Done button on this screen completes the action configuration and returns you to the Policy configuration screen. Clicking the Apply Policy button commits the changes made to the policy.

Add a note hereThere you have it! We now have a working service that accepts and schema validates a SOAP request document, fetches a response document, and returns it back to the client in the response. Although this is just an example service to demonstrate the configuration of the Processing Policy and the two actions discussed, this type of service can actually provide a tremendous amount of value in a real-world scenario. As we mentioned earlier, suppose there was actually going to be a service on the backend that would generate the response to the client. Many times the actual service is not available or ready when you are prepared to create your DataPower service. If the schema for the request message is defined, you can create a similar service to perform the validation of the request document and actually generate the response as shown in this example. This can then be used by the actual clients as they test their client code until the real backend service is ready and available.

#### Testing the BookQueryService

Add a note hereTo prove that our service is actually validating the request and response document, we can send a test message to the service using the built-in tool as in the previous example. Listing 8-2 shows a schema-compliant sample request document that we might send to our BookQueryService.

Add a note hereListing 8-2: Sample BookQueryService Request Document

Add a note here<?xml version='1.0' ?>

<SOAP-ENV:Envelope

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body>

<book>

<name>Moby Dick</name>

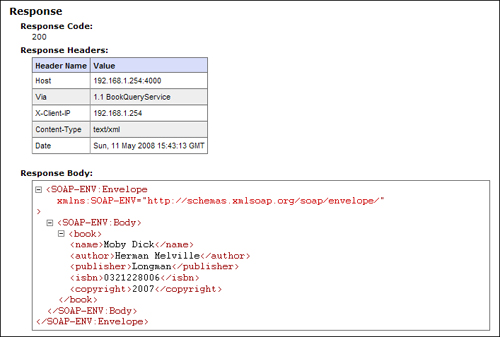
<author>Herman Melville</author>

</book>

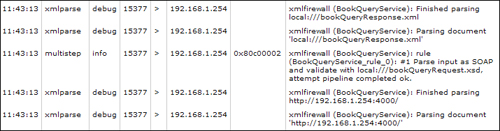
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

Add a note hereUpon sending the document shown in Listing 8-2, we would expect a valid response from the service as it should have passed the Validate action in the request and generated a response. Figure 8-21 shows the response received after submitting the request document shown in Listing 8-2. Notice that a response code of 200 was returned with the response document that was fetched from the request rule within the service. This would indicate that the request and response were both successful.

[](javascript:PopImage('IMG_145','http://images.books24x7.com/bookimages/id_30903/08fig21_alt.jpg','707','477'))  
Add a note hereFigure 8-21: BookQueryService response.

Add a note hereWe received a valid response from the service as expected but how can we be sure that the request document was validated against the schema specified in the configuration? We can take a look at the DataPower logs to see that this was accomplished. Figure 8-22 shows a portion of the log from this request. Notice that the input was parsed as SOAP and was validated against the bookQueryRequest.xsd schema.

[](javascript:PopImage('IMG_146','http://images.books24x7.com/bookimages/id_30903/08fig22_alt.jpg','715','188'))  
Add a note hereFigure 8-22: BookQueryService request log.

Add a note hereNow we have proven that a valid, schema-compliant SOAP request document will be processed by the XMLFW and return a response SOAP document. Let’s see what happens if the request document does not comply with the schema used in the validation step within the service. To do this, we can change our request document so that it no longer conforms to the schema being used to validate it within the service. To accomplish this we add an additional node to the document called <publisher> as shown in Listing 8-3.

Add a note hereListing 8-3: Invalid BookQueryService Request Document

Add a note here<?xml version='1.0' ?>

<SOAP-ENV:Envelope

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body>

<book>

<name>Moby Dick</name>

<author>Herman Melville</author>

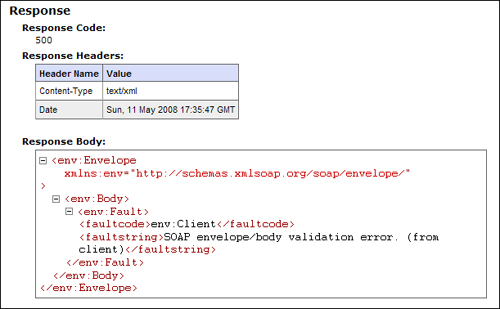
<publisher>xyz Publishing</publisher>

</book>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

Add a note hereAfter submitting this invalid request, you can see in Figure 8-23 that we get a return code of 500 with a SOAP fault returned in the response body.

[](javascript:PopImage('IMG_147','http://images.books24x7.com/bookimages/id_30903/08fig23_alt.jpg','611','378'))  
Add a note hereFigure 8-23: Response from Invalid BookQueryService request.

Add a note hereIf we take a look at the log from the request, shown in Figure 8-24, you can see that the request was rejected due to a validation error. It indicates that the service found a node called <publisher> when it expected to find an end element. This can be handy when determining the cause of the validation error.

[](javascript:PopImage('IMG_148','http://images.books24x7.com/bookimages/id_30903/08fig24_alt.jpg','1183','173'))  
Add a note hereFigure 8-24: Invalid BookQueryService Request log.

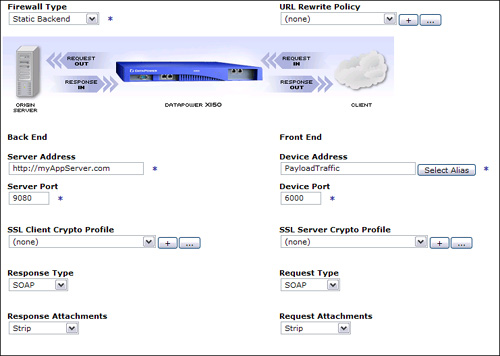
### Add a note hereConfiguring the Backend

Add a note hereUp to this point, we have been focusing on the XML Firewall’s ability to be configured in loopback mode. It is entirely possible for the XMLFW service to proxy an actual backend application or service providing all the threat protection, schema validation, and multistep processing capabilities that the XMLFW has to offer.

Add a note hereWhen creating an XMLFW service to proxy an application or a service running on a server, the URL of the server that the service forwards requests to must be defined. In many cases, this server is the same for all requests and can be specified in the XMLFW configuration itself. This is known as a static backend. There might, however, be times where this will be different based on information that can be obtained from the request such as data found in the payload or request headers. This is defined as a dynamic backend within the service configuration.

#### Defining a Static Backend

Add a note hereWhen the XMLFW service forwards all requests to one backend server[[3](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=863242058&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ftn.ch08fn03)], the URL for the backend server can be defined in the service configuration within the General tab of the XML Firewall Configuration screen. Before specifying the actual URL, you must first specify that the service contains a static backend server. This can also be set within the General tab in the XML Firewall Configuration screen by setting the Firewall Type to Static Backend. After this option is selected, the screen redraws, presenting additional fields for defining the backend server address, port, response type, and the SSL Client Crypto Profile to use if the backend connection is using SSL. Figure 8-25 shows the General tab of an XMLFW, where the Firewall Type has been set to Static Backend. Because of this, the configuration parameters used to specify the backend connection information are displayed. Notice that these parameters are similar to those used on the front end connection but are used to establish the connection to the backend server.

[](javascript:PopImage('IMG_149','http://images.books24x7.com/bookimages/id_30903/08fig25_alt.jpg','741','527'))  
Add a note hereFigure 8-25: Configuring a static backend.

Add a note hereNotice that there is a separate parameter to specify the port that is used as part of the Server Address; therefore, the port is not specified in the URL itself.[[4](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=863242058&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ftn.ch08fn04)] The Server Address parameter also dictates whether the backend connection uses SSL. If the protocol specified is HTTPS, the backend communication will be over SSL. The service then uses the SSL Client Crypto Profile specified for all the SSL information. Configuring a service to use SSL is covered in detail in [Chapter 18](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3288#3288), [“DataPower and SSL.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3288#3288)

#### Dynamic Backend

Add a note hereYou might want to forward requests to differing backend servers based on some information contained in the request. For example, suppose a hotel reservation system was being proxied by an XMLFW service. It might be desirable to send reservation requests to a different server on a different network based on the hotel preferred status of the customer making the reservation. So a customer with platinum preferred status might be sent to a faster network used only to process requests for platinum customers where all other customer requests would go to an entirely different server and network. In this case, the service can make the decision for the backend server at runtime based on the field in the message content that specifies the status of the customer.

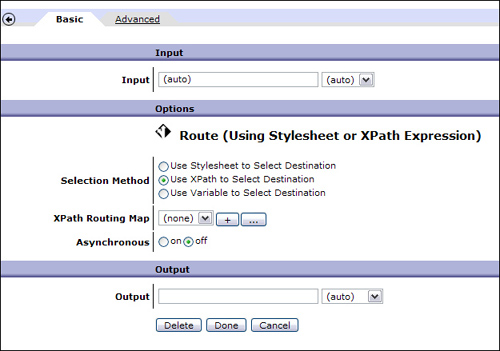
Add a note hereTo accomplish this dynamic routing scenario, you first need to set the Firewall Type to Dynamic-Backend in the service configuration. Note that if you are creating the XMLFW via the wizard; this option will not be presented. You must first create the service with a different Firewall Type and change it in the service configuration screen. It is most convenient to initially select loopback-proxy in this case because you are not required to enter a backend URL.

Add a note hereAfter the Firewall Type is set to Dynamic-Backend, it is then the responsibly of the Processing Policy configured to determine the URL for the server that the request will be forwarded to at runtime. This is typically configured by adding a Route action to the request rule(s) of the Processing Policy as shown in Figure 8-26. Note that this must be done for all request rules within the Processing Policy that are expected to forward traffic to a backend location.

[](javascript:PopImage('IMG_150','http://images.books24x7.com/bookimages/id_30903/08fig26_alt.jpg','701','204'))  
Add a note hereFigure 8-26: Drag the Route action onto the policy line.

Add a note hereAfter the Route action is added to the request rule, you must configure it to specify how to determine the backend location. The Route action configuration screen provides three methods for setting the backend server as shown in Figure 8-27. These three methods are

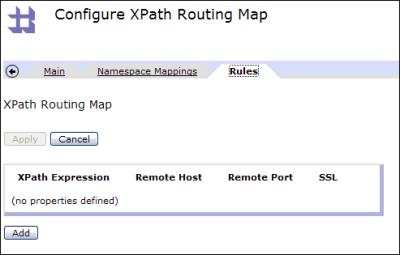
* Add a note here**Use Stylesheet to Select Destination—** Allows you to reference an XSLT stylesheet that determines the backend server.
* Add a note here**Use XPath to Select Destination—** Executes an XPath expression against the input message to the Route action. If a match is found, the backend is set to the URL specified in the XPath routing map that will need to be configured.
* Add a note here**Use Variable to Select Destination—** Uses the value of a previously set DataPower variable. It is also possible to hard code the destination URL here.

[](javascript:PopImage('IMG_151','http://images.books24x7.com/bookimages/id_30903/08fig27_alt.jpg','625','439'))  
Add a note hereFigure 8-27: Route action configuration screen.

Add a note hereIn our hotel reservation example, we will set the backend destination based on the status of the customer making the request. Because this information is contained within the request document, we can use the Use XPath to Select Destination option to make this decision. This allows us to create an XPath routing map that contains XPath expressions and corresponding backend addresses to set the backend destination.

Add a note hereAfter the Use XPath to Select Destination is selected, you see a field labeled XPath Routing Map appear on the configuration screen. This references an object that contains the XPath expressions and backend addresses. We create a new XPath Routing Map for our example.

Add a note hereThe main configuration page of the XPath Routing Map object requires a name for this new object. This name should be reflective of the function that this object performs. We call our XPath Routing Map HotelStatusRouting for the hotel reservations example. The configuration of the XPath expressions and mappings is done on the Rules tab of the configuration page as shown in Figure 8-28. As you can see by the column headings, this is where each XPath expression maps to a remote host and port.

[](javascript:PopImage('IMG_152','http://images.books24x7.com/bookimages/id_30903/08fig28.jpg','493','315'))  
Add a note hereFigure 8-28: XPath routing map rules.

Add a note hereLet’s suppose that we wanted to create a routing map rule for setting the remote host and port for our hotel reservation requests that contain a <status> element value of Platinum in the request document. A sample request document that would match this requirement is shown in Listing 8-4. We can click the Add button on the Rules tab to create a new rule for this. We then see the four fields that need to be entered that correspond to the column headings in Figure 8-28. Here is where you would provide the XPath expression that matches the element of interest. In our case, this is an element named <status> with a value of Platinum. You then enter the remote host and port that the request should be routed to if the XPath expression evaluates to true. The last field is a radio button to indicate whether the connection should be over SSL.

Add a note hereListing 8-4: Hotel Reservation Request Document

Add a note here<?xml version='1.0' ?>

<SOAP-ENV:Envelope

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body>

<reservation>

<name>Joe Traveler</name>

<checkin-Date>2008-05-12</checkin-Date>

<checkout-Date>2008-05-16</checkout-Date>

**<status>Platinum</status>**

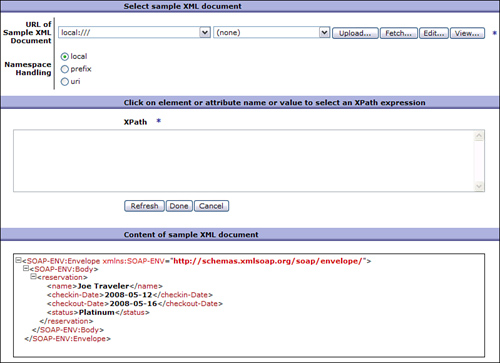
</reservation>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

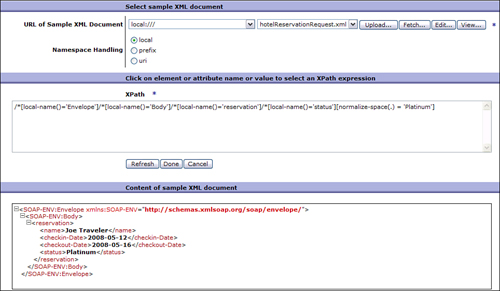
Add a note hereOf these four fields, three of them should be straightforward and simple to configure. The one that might not be so simple is the actual XPath expression that will be applied to the request document. Luckily, you don’t have to be an XPath expert to determine the correct XPath expression to perform the appropriate match. There is a built-in XPath tool that creates the XPath expression for you by simply referencing a sample request XML document in a point-and-click fashion.

Add a note hereNotice next to the XPath Expression field there is a button labeled XPath Tool. Clicking this button brings up this tool. Here is where you reference a sample XML document used for building the XPath expression. Note that this file can be fetched from a remote location or referenced on the DataPower file system. There is a mechanism here to upload the file to the local file system if required. After the file is referenced, the contents of the XML document shows in the bottom of the screen labeled Content of Sample XML Document as shown in Figure 8-29.

[](javascript:PopImage('IMG_153','http://images.books24x7.com/bookimages/id_30903/08fig29_alt.jpg','799','580'))  
Add a note hereFigure 8-29: XPath tool.

Add a note hereAfter this file is referenced in the URL of Sample XML Document field, you can then specify how to handle the namespaces contained within the document. For example, selecting Local creates an XPath expression that ignores the namespaces within the document when applying the expression. This option should be used if you are sure that there will not be duplicate element names that will need to be namespace qualified. In our hotel reservation example, there is no concern of duplicate names, so there is no need to build an XPath expression that requires the namespace qualified element names; so we will select this option. Doing this also helps to avoid exceeding the XPath string length limit when there are many complex namespaces.

Add a note hereNow that we have specified the sample file and how to handle namespaces, we can build the XPath expression. This can be done by simply clicking the value in the element of interest within the sample XML document at the bottom of the screen. In our example, because we are interested in the value Platinum in the <status> element, we will click the word Platinum in our sample document. The tool will then generate the corresponding XPath expression in the XPath field in the middle of the screen as shown in Figure 8-30. This XPath expression evaluates to true when the <status> element contains a value of Platinum in a similarly formed XML document.

[](javascript:PopImage('IMG_154','http://images.books24x7.com/bookimages/id_30903/08fig30_alt.jpg','964','561'))  
Add a note hereFigure 8-30: XPath expression created.

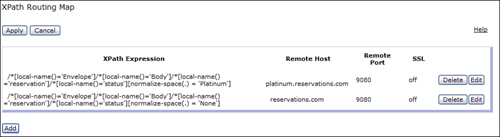
Add a note hereClicking the Done button adds this XPath expression to the Rule configuration screen. Here you can specify the remote host and the port that the request should be forwarded to when this XPath expression evaluates to true. In our hotel reservation service, we are sending all requests with a customer status of Platinum to a remote host address platinum.reservations.com on port 9080 as shown in Figure 8-31. Notice that the Remote Host is an actual server address and not a URL. This can be an IP address, a DNS-resolved hostname, or a Load Balancer Group name.

[](javascript:PopImage('IMG_155','http://images.books24x7.com/bookimages/id_30903/08fig31.jpg','465','268'))  
Add a note hereFigure 8-31: Platinum route rule.

Add a note hereNow that we have configured a routing rule for our platinum members, we need to configure a route rule for requests from members that do not have status with the hotel and the <status> element contains a value of None. This is configured the same way we configured the routing rule for platinum status; however, the XPath expression matches on the <status> element that contains None instead of Platinum. This route rule points to the remote host and port for the server that handles these types of requests. Figure 8-32 shows our route rule configuration where this XPath match routes to a remote host with an address of reservations.com on port 9080.

[](javascript:PopImage('IMG_156','http://images.books24x7.com/bookimages/id_30903/08fig32.jpg','470','267'))  
Add a note hereFigure 8-32: No Status route rule.

Add a note hereWe have now seen how we can configure route rules within a route map. In our hotel reservations service example, we used two separate XPath expressions to match on the same field but with different values to determine the backend server for the service. Figure 8-33 shows the final hotelRoutingMap configured for this example service. Notice that the XPath expressions are almost identical for each rule with the only difference being the last string in the expression where one is Platinum and the other is None. The corresponding remote host and ports are then listed for each.

[](javascript:PopImage('IMG_157','http://images.books24x7.com/bookimages/id_30903/08fig33_alt.jpg','920','252'))  
Add a note hereFigure 8-33: HotelRoutingMap.

Add a note hereAfter you configure your Route action, you can then add additional actions and rules to the policy as required. To demonstrate the Route action, we will not add any additional actions in our example. This service simply routes requests to the appropriate backend server.

Add a note hereTo demonstrate this dynamic routing capability in action, we send a request to the hotel reservation service that was configured in the previous example. First, we send a request to the service that has a member status of Platinum as shown in Listing 8-5.

Add a note hereListing 8-5: Platinum Request

Add a note here<?xml version='1.0' ?>

<SOAP-ENV:Envelope

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body>

<reservation>

<name>Joe Traveler</name>

<checkin-Date>2008-05-12</checkin-Date>

<checkout-Date>2008-05-16</checkout-Date>

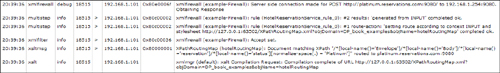
**<status>Platinum</status>**

</reservation>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

Add a note hereAfter submitting this request, we can look at the log shown in Figure 8-34 to see where the request was forwarded to. Notice that the hotelRoutingMap was used to evaluate the request and set the remote host and remote port. In this case the request was sent with the <status> element containing a value of Platinum, so we see that the server side connection was made to the remote host that was configured to receive requests from platinum hotel members.

[](javascript:PopImage('IMG_158','http://images.books24x7.com/bookimages/id_30903/08fig34_alt.jpg','1179','172'))  
Add a note hereFigure 8-34: Platinum request log.

Add a note hereTo demonstrate the routing for requests from customers with a status of None, we send the request document shown in Listing 8-6.

Add a note hereListing 8-6: No Status (None) Request

Add a note here<?xml version='1.0' ?>

<SOAP-ENV:Envelope

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body>

<reservation>

<name> Joe Traveler</name>

<checkin-Date>2008-05-12</checkin-Date>

<checkout-Date>2008-05-16</checkout-Date>

**<status>None</status>**

</reservation>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

Add a note hereNotice the logs from this request in Figure 8-35. You see that the same routing map was used; however, the remote server and port were set to the appropriate values that were configured for customers with no status.

[](javascript:PopImage('IMG_159','http://images.books24x7.com/bookimages/id_30903/08fig35_alt.jpg','1181','168'))  
Add a note hereFigure 8-35: No Status Request Log.

Add a note hereThis section demonstrated the power and flexibility of dynamic routing. Although our example showed this routing based on the value of an element in the request document, it is entirely possible to perform this routing based on any information that can be obtained from the request itself, such as request header values. This information used to determine the backend server does not need to be in the request at all but can be any information the service has access to. For example, you can perform dynamic routing based on the time of day or day in the month. This is all possible due to the fact that the route determination can be easily configured within the Route action, or it can be customized within an XSLT stylesheet.

Add a note hereWe have mentioned several times that a loopback proxy can be handy as a replacement for the actual backend server when the real server is not available. To show how this can be done elegantly without disrupting the service configuration when the server becomes available, let’s take a closer look at this hotel reservation example.

Add a note hereIn our routing map, we hard-coded two separate backend servers that will be used based on some information in the request document. Suppose these servers were not available and we wanted to use a loopback proxy to emulate these servers until they became available. It would not be very convenient to have to go back to the routing map when we wanted to add our real backend servers as the destination servers. Fortunately, there is a solution. Much like the host alias is used to provide an abstraction for IP address of the DataPower interface that the service will listen on, we can use a static host to define the backend servers that the service will forward requests to. In our hotel reservation example, we forwarded platinum requests to a server address platinum.reservations.com and nonstatus customers to reservations.com. By configuring a static host for these backend servers, as shown in Figure 8-36, we can change the static host entries to the actual backend server addresses when they are available, and the routing map never has to be touched. A static host must be defined in the default domain and can be navigated to from the left navigation menu under Network→DNS Settings→Static Hosts. This topic is discussed in detail in [Chapter 15](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2720#2720), [“Build and Deploy Techniques.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2720#2720)

[](javascript:PopImage('IMG_160','http://images.books24x7.com/bookimages/id_30903/08fig36.jpg','472','301'))  
Add a note hereFigure 8-36: Static Host definition.

Add a note here[[3](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=863242058&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ch08fn03)]The URL for the backend server can be a URL to one server location or it can reference a Load Balancer Group defined on the device that balances the traffic across multiple servers.

Add a note here[[4](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=863242058&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ch08fn04)]If a Load Balancer Group object is specified for the Server Address, the ports specified in the members of the group take precedence over the port specified in the service configuration

## Summary

Add a note hereThis chapter explained, and demonstrated through examples, some common use cases for an XMLFW as well as some basic fundamentals of service configuration. You saw how to create a simple loopback firewall, implement a Fetch action, schema validate a request document, and perform dynamic routing. Many of the lessons learned in this chapter are not only applicable to the XMLFW service type but can also be applied to many of the other service types. This knowledge will prove to be extremely valuable as you build upon it throughout the rest of the chapters in this book.