**Chapter 9: Multi-Protocol Gateway**

Add a note hereThis chapter describes the implementation of the Multi-Protocol Gateway Service (MPGW), a service that is available on the DataPower XS40 and XI50. Among its extensive capabilities, the MPGW plays a significant role in the Enterprise Service Bus (ESB), an important part of many distributed and Service Orientated Architectures (SOA). This chapter introduces the concepts of the ESB, and it demonstrates several use cases and their configuration and execution utilizing the MPGW.

**Add a note here****Enterprise Service Bus Introduction**

Add a note hereAs shown in Figure 9-1, the ESB is an architectural pattern that is used as an intermediary and mediation mechanism for service providers and requestors. By supplying interconnectivity to disparate software services, the ESB can provide a virtualization of the interfaces, protocols, and patterns required by individual service providers. It can also act as the mediator between requestor and provider with the benefit of disassociating the requirements of the service from the client. Benefits of this architecture include allowing for many potential service implementations and providers whose protocol and message structure might be unknown to the requestor. For example, the service might receive requests over many protocols, including HTTP, WebSphere MQ, WebSphere JMS, NFS, IMS, or FTP, and integrate with providers over a similar range of protocols with differing message schemas mediated through a Processing Policy.

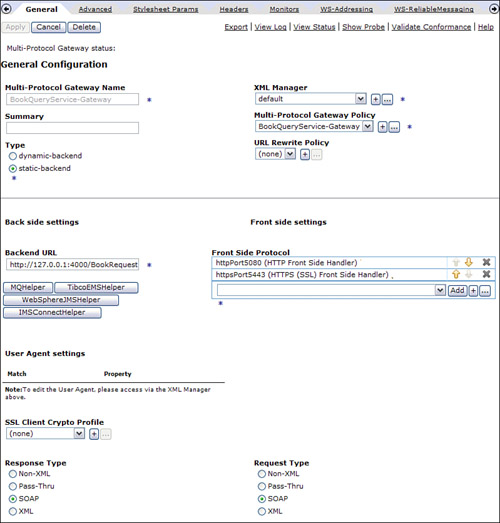
[Image from book](javascript:PopImage('IMG_161','http://images.books24x7.com/bookimages/id_30903/09fig01.jpg','432','55'))  
Add a note hereFigure 9-1: Enterprise Service Bus.

Add a note hereThe MPGW can be considered a superset of the XML Firewall (XMLFW), and although the XMLFW is convenient in many cases for rapid prototyping, the MPGW should be considered the primary vehicle for most non-Web service XML/SOAP use cases. Web service applications can be identified in most cases by the existence of a WSDL document. Therefore, the majority of use cases in which no WSDL is defined should use the MPGW for service implementation.

Add a note hereIn addition to describing the construction of the MPGW service, this discussion encompasses several objects associated with the MPGW, such as the Front Side Handler (FSH), XML Manager, User Agents, AAA Policies, and the building of Backend URLs. You should be familiar with the basic patterns offered by the DataPower services and their basic configurations. If not, you should review [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) and [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) before proceeding with this chapter.

## MPGW Basic Configuration

Add a note hereThe MPGW is primarily configured via the WebGUI interface. Configuration via other DataPower options such as the Command Line Interface (CLI) or the XML Management Interface is possible, but not recommended. Figure 9-2 shows the General tab of the MPGW configuration screen. You should be familiar with many of the properties shown on this screen such as Request and Response Type, XML Manager, and the Multi-Protocol Gateway Policy, as the MPGW shares many of these configuration parameters with the other services.

[](javascript:PopImage('IMG_162','http://images.books24x7.com/bookimages/id_30903/09fig02_alt.jpg','799','836'))  
Add a note hereFigure 9-2: Multi-Protocol Gateway Service General tab.

### Add a note hereProtocol Control Objects

Add a note hereThe MPGW accepts traffic over the FSH and provides support for several protocols. Backend URL strings are utilized for backend communication and can leverage a similar set of protocols. Both front and back side processing can be affected by actions within the Processing Policy. For example, the backend endpoint can be customized and routing information dynamically determined via a Route action, or the MQ Queue on which reply messages are placed can be changed. We see examples of this and front side customization in the examples to follow.

### Add a note hereMPGW Front Side Handlers

Add a note hereDataPower provides FSHs targeted to specific protocols. For example, you will find an FSH object specifically targeted to HTTP, another to HTTPS, one for WebSphere MQ, WebSphere JMS, NFS, and so on. You should find an FSH to handle the vast majority of protocols you’ll need to receive traffic on. As we will see in the examples to follow, each version of the FSH contains control properties to fine-tune the mediation, for example to handle MQ sync-pointing or the renaming of files fetched via FTP. Not only can the MPGW accept traffic easily over the various FSHs, but a single MPGW can implement more than one, allowing it to accept traffic over many protocols simultaneously. Can you imagine having to write Java code to do this? Table 9-1 lists the supported protocols.

| Add a note hereTable 9-1: Front Side Handler protocols  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=824352029&objectid=ch09table01&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=824352029&objectid=ch09table01&objecttype=spreadsheet) | |
| --- | --- |
| **Add a note hereProtocol** | **Add a note hereDescription** |
| Add a note hereFTP | Add a note hereThe gateway can act as either an FTP client, polling a remote FTP server for requests, or as an FTP server, accepting incoming FTP connections. |
| Add a note hereHTTP | Add a note hereReceive requests using unencrypted HTTP. |
| Add a note hereHTTPS | Add a note hereReceive encrypted requests using HTTP over Secured Sockets Layer. |
| Add a note hereIMS | Add a note hereThe gateway can accept incoming IMS protocol requests and can initiate IMS connections on the back side. |
| Add a note hereNFS | Add a note hereThe gateway can poll an NFS-mounted directory for the presence of new files and place responses on an NFS-mounted directory. |
| Add a note hereWebSphere MQ | Add a note hereThe gateway can use GET and PUT queues to communicate using MQ messages. |
| Add a note hereTIBCO EMS | Add a note hereSupports TIBCO Enterprise Messaging Service, if licensed. |
| Add a note hereWebSphere JMS | Add a note hereSupports the default WebSphere JMS server (Service Integration Bus). |

### Add a note hereBackend URL

Add a note hereWhile the front side traffic is described by the FSH, the back side of the MPGW is described by the creation of a backend URL string. This URL is constructed in a standard URL format and might include a protocol, hostname, port, URI, and query names and values. The protocol string determines which protocol will be used for the back side connection. In addition, the URL might contain custom parameters such as the name of a queue to place messages on, or the channel to use for connectivity. In fact, the backend URL provides a similar set of features as the FSH in another format.

Add a note hereWe describe the various backend URL formats later, but for now we demonstrate some examples in Listing 9-1. You might notice the protocol prefix that designates which protocol that will be used for communication. You will, of course, be familiar with HTTP, but the others beginning with dp may not be familiar. These are DataPower internal URL protocol formats that we will introduce within this chapter.

Add a note hereListing 9-1: Sample Backend URLs

Add a note herehttp://someCompany.com:8080/aService/aRequest

https://someCompany.com:4443/aService/aRequest

dpmq://mqQueueManagerObject/uri?RequestQueue=requestQueue

dpnfs://host:23/URI

dptibems://emsServerObject/?RequestQueue=reqQueueName

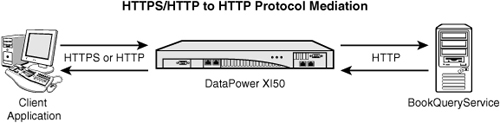
## Example Configurations

Add a note hereTo demonstrate the capabilities of the MPGW and the use of various types of FSHs and backend URLs, examples are presented to describe typical use cases. Several of these use cases demonstrate advanced features such as header injection, dynamic routing, and queue selection.

Add a note hereMany of these examples use the BookQueryService XML Firewall that was presented in [Chapter 8](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1082#1082), [“XML Firewall.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1082#1082) It is used in several instances throughout this chapter to act as a backend server. It receives a request document on port 4000, containing an author and book title, and responds with a response document containing the publisher, ISBN, and copyright for the book. Please refer to [Chapter 8](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1082#1082) to understand the functionality of this XMLFW. Many of these examples will use Pass-Thru request and response types. The primary objective in these examples is to demonstrate protocol mediation rather than message enrichment, policy enforcement, or other functions typically performed in the processing policy.

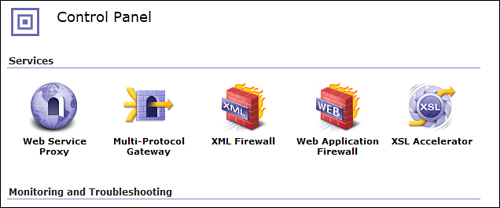
### Add a note hereProtocol Mediation: HTTPS and HTTP to HTTP

Add a note hereFigure 9-3 demonstrates an architectural perspective of simple protocol mediation. Client applications send requests over HTTP or HTTPS to a MPGW service on the XI50, which then forwards the requests using HTTP to the backend service.

[](javascript:PopImage('IMG_163','http://images.books24x7.com/bookimages/id_30903/09fig03_alt.jpg','577','142'))  
Add a note hereFigure 9-3: Protocol mediation between HTTPS and HTTP.

Add a note hereThis example demonstrates a basic implementation of the MPGW. The service is constructed to serve two classes of clients: external clients whose credentials will be validated and transport encrypted over HTTPS, and internal clients whose location will be validated via an Access Control List and will be allowed to communicate over HTTP. In each case, messages are sent to a backend service over HTTP. The use of HTTP should be limited to internal networks if the message is of a sensitive nature because a third party could potentially view it.

Add a note hereThis example introduces the fundamental concepts, such as the ability to assign multiple FSHs to a MPGW and the creation of the backend URL. As with all service creation, the entry point is the Control Panel; from there, simply select the Multi-Protocol Gateway icon as shown in Figure 9-4.

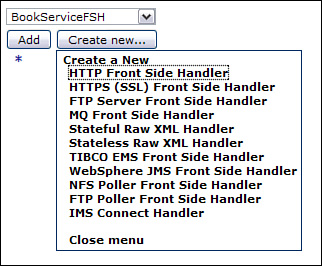
[](javascript:PopImage('IMG_164','http://images.books24x7.com/bookimages/id_30903/09fig04_alt.jpg','636','265'))  
Add a note hereFigure 9-4: Control Panel with the Multi-Protocol Gateway icon.

Add a note hereThis brings you to a list of all previously created MPGW objects. From the Multi-Protocol Gateway list screen you can click on any of the existing MPGWs to open them for editing; if you are creating a new MPGW, click the Add button at the bottom of the display.

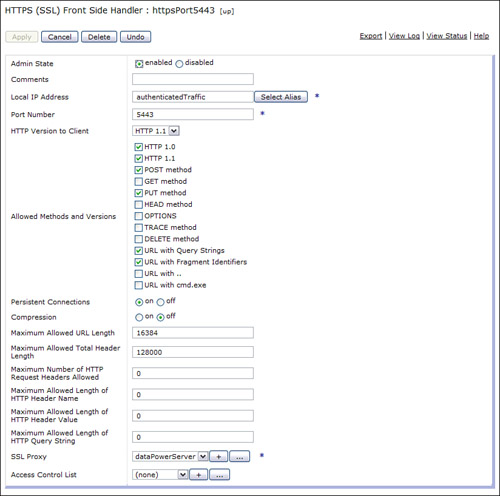
Add a note hereIn this example, a new MPGW is created, it is assigned a name (BookQueryService-Gateway), and a Multi-Protocol Gateway Policy of the same name as seen in Figure 9-5. It is often beneficial to use the same name for the policy and the service to which it is associated. The backend URL is used to designate the URL of the backend connection. In its simplest form the URL is a standard protocol, host, port, and URI designator. We can override the URI components, as will be seen in the later examples. Special query parameters may be used that affect the functionality of the connection and characteristics such as queue names and security settings.

[](javascript:PopImage('IMG_165','http://images.books24x7.com/bookimages/id_30903/09fig05_alt.jpg','831','358'))  
Add a note hereFigure 9-5: MPGW creation.

Add a note hereAs our first step, we create a new HTTPS (SSL) FSH. Click the Create new button under the Front Side Protocol (see Figure 9-6) to expose a list of possible Front Side Protocols, and then click HTTPS (SSL) Front Side Handler.

  
Add a note hereFigure 9-6: MPGW, creating a new HTTPS FSH.

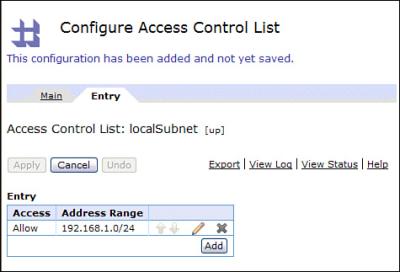
Add a note hereThe FSH as seen in Figure 9-7 enables the configuration of several protocol-specific threat protection capabilities. For example, the allowable HTTP(S) request methods may be controlled by the parameters of the FSH. Notice that the GET method is not allowed by default. The MPGW is normally used for PUT/POST method processing; however GET can also be used if this is enabled. Controls are also provided for URL options such as query strings, fragment identifiers, “..”, and filenames with .exe extensions. Finally, various lengths may be checked including that of headers, URL, Query Parameters, and so on.

[](javascript:PopImage('IMG_167','http://images.books24x7.com/bookimages/id_30903/09fig07_alt.jpg','824','818'))  
Add a note hereFigure 9-7: HTTPS FSH.

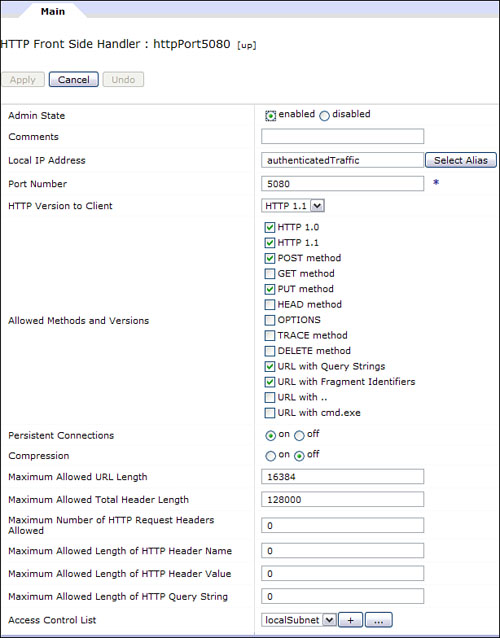
Add a note hereOur example will assign the port of 5433 for use by this HTTPS FSH. Notice that the Local IP Address has been assigned a Host Alias rather than an IP address. This allows a client request and response traffic only on the Ethernet interface assigned to the authenticatedTraffic host alias, thereby assuring that client traffic is from a known subnet. The host alias object is unique to each appliance, allowing for an environment neutral configuration in regard to interface assignment.

Add a note hereAs the FSH is using HTTPS, a critical component of this configuration is the SSL Proxy object. This object controls all aspects of the SSL communication to this FSH. It determines which X.509 private key and public certificate are used to initiate the SSL communication and determines whether validation of the client’s X.509 public certificate will take place. You can refer to [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) for a detailed discussion on the creation and use of the SSL Proxy. After completing the FSH, you need to add it to the MPGW by clicking the Add button.

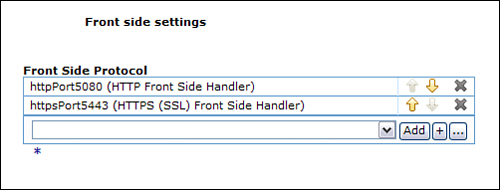
Add a note hereHaving created and added the HTTPS FSH, the same sequence is performed for the HTTP FSH. The methodology is the same: First you need to create the HTTP FSH, and then add it to the MPGW. A distinguishing characteristic of our HTTP FSH use case (only allow local traffic) is the requirement to validate the location of the client. This is done by the creation of an Access Control List (ACL) from the FSH. Please refer to [Chapter 7](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=868#868) for more details on ACLs. Figure 9-8 demonstrates the creation of the ACL, in this case only allowing clients with an IP in the 192.168.1.0–192.168.1.255 range to have permission to access the service.

[](javascript:PopImage('IMG_168','http://images.books24x7.com/bookimages/id_30903/09fig08.jpg','475','323'))  
Add a note hereFigure 9-8: ACL.

Add a note hereNow that the ACL has been created, it can be added to the FSH. Figure 9-9 shows the association of the ACL on the HTTP FSH; we are certain that only traffic from the subnet identified by the ACL will be allowed over this particular FSH. We now have a two-factor approach for authenticating traffic. Not only must traffic be received over the interface assigned by the host alias, but traffic utilizing HTTP must be from a known IP address range.

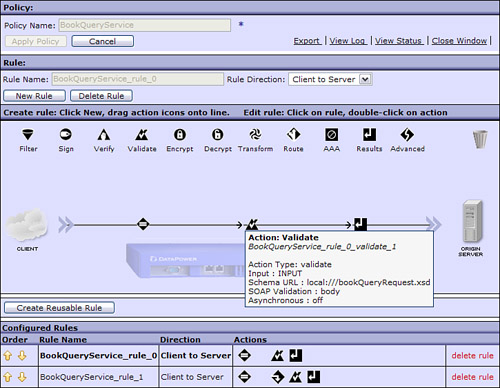
[](javascript:PopImage('IMG_169','http://images.books24x7.com/bookimages/id_30903/09fig09_alt.jpg','614','784'))  
Add a note hereFigure 9-9: HTTP FSH with ACL.

Add a note hereNow that both the HTTP and HTTPS FSHs have been created, they can be added to the MGPW. Figure 9-10 shows the configuration of both. Traffic will be allowed on either of the FSH as long as it is received over the authenticatedTraffic interface, and HTTP traffic will be accepted only from the known IP addresses identified by the ACL.

[](javascript:PopImage('IMG_170','http://images.books24x7.com/bookimages/id_30903/09fig10_alt.jpg','502','191'))  
Add a note hereFigure 9-10: MPGW with multiple FSHs.

#### The Processing Policy

Add a note hereEach MPGW will have a Processing Policy associated with it, which as in all services, will define the actions performed on transaction request and response messages from the backend service. You will remember that the request and response type play a role in policy processing; for instance, a Pass-Thru type will bypass policy execution. In the BookQueryService-Gateway policy, each request and response will be schema validated. Messages that result in schema validation errors will result in transaction failure and a SOAP fault being returned to the client. The policy, as shown in Figure 9-11, contains the request (“Client to Server”) and the response (“Server to Client”) rules and their Validate actions.

[](javascript:PopImage('IMG_171','http://images.books24x7.com/bookimages/id_30903/09fig11_alt.jpg','725','562'))  
Add a note hereFigure 9-11: BookQueryService Processing Policy.

#### Testing the Service

Add a note hereThe following request sends a sample message to our newly defined MPGW service exposed on port 5433 over SSL. We use cURL, and the -k (insecure) option is used to avoid the server name error that would have been generated by SSL by not validating the server’s SSL certificate. You can use the --help option of cURL for further explanation of its options. For more information on cURL, visit [http://curl.haxx.se/.](http://curl.haxx.se/) In this example (shown in Listing 9-2) the request document is sent to the BookQueryService-Gateway MPGW over SSL on port 5433. It is forwarded to the BookQueryService XMLFW by the MPGW and a response document is returned. Schema validation is performed on both the request and the response to ensure the integrity of the documents.

Add a note hereListing 9-2: BookQueryService HTTPS Testing

Add a note herecurl --data-binary @bookQueryRequest.xml

https://192.168.1.35:5443/bookQuery -k

<?xml version="1.0" encoding="UTF-8"?>

<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>Longman</publisher>

<isbn>0321228006</isbn>

<copyright>2007</copyright>

</book>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

Add a note hereA second request is made over port 5080 to demonstrate the HTTP, local client use case, as seen in Listing 9-3. The request is made from IP address 192.168.1.108, which is within the range specified by the ACL. Had the request been outside that range, the request would have been rejected.

Add a note hereListing 9-3: BookQueryService HTTP Testing

Add a note herecurl --data-binary @bookQueryRequest.xml http://192.168.1.35:5080/bookQuery

<?xml version="1.0" encoding="UTF-8"?>

<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>Longman</publisher>

<isbn>0321228006</isbn>

<copyright>2007</copyright>

</book>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

### Add a note hereFTP Use Cases

Add a note hereFTP is used to get files from and put files to remote servers. An FTP server uses a default port of 21 for listening to client connections, and creates a secondary connection for the actual transfer of data. Typical use cases involve polling a server for files, and the getting and putting of files onto an actual FTP device.

Add a note hereWithin the DataPower appliance, two distinct FSHs can be used to implement FTP. The FTP Poller FSH, as its name implies, retrieves messages through FTP server inquiry. The FTP Server FSH is used to implement the functionality of an FTP server on the appliance or in conjunction with an off device FTP server.

Add a note hereThe FTP Poller process and the documents it receives can be associated with and processed by Processing Policy actions. A remote FTP server can be accessed and queried for document availability. After a document is fetched it is processed like any other request. Actions such as message enrichment might be performed, or perhaps the request is authenticated. After the message is processed, it can be sent to a backend resource such as an MQ queue or to an application over HTTP. The point here is that where the message came from is irrelevant; the processing capabilities of the ESB are enacted, and the protocol has been decoupled from the service. The request that we demonstrated over HTTP can be processed by fetching the request from an FTP server just as easily!

Add a note hereThe FTP Server FSH acts as a façade and can be used in conjunction with another FTP server that resides off the appliance. It can also use the DataPower device’s on-board file system to respond to GET/PUT requests and eliminates the need for an actual off box FTP server. The FTP Server FSH property Filesystem-Type controls this behavior as we see in the examples that follow.

Add a note hereAlthough front side processing of FTP resources is provided via the FSH, backend FTP resources are accessed via the FTP URL construction, as is the case with other supported backend protocols. We’ll see some actual use cases, but for now, an example of the FTP URL is presented in Listing 9-4.

Add a note hereListing 9-4: Backend URL Example

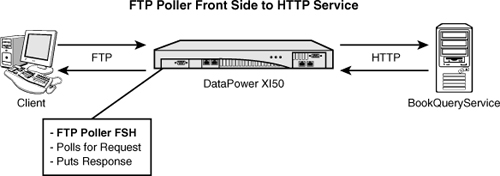
Add a note hereftp://user:password@host:port/URI?queryParameters

Add a note hereWe’re now going to put this together with three functioning examples of FTP use cases. They are

* Add a note here**An FTP Poller with User Agent for injection of Username and Password—** This example performs authenticated fetches of requests for the BookQueryService from an FTP server, processes the request, and returns responses to the FTP server.
* Add a note here**An FTP server with Transparent Filesystem and Password AAA Policy—** This example demonstrates the use of the device as an authentication gateway to an off device FTP server.
* Add a note here**An FTP server with Virtual Ephemeral Filesystem—** This example demonstrates the use of the device as a virtual FTP server, with the PUTs of BookQueryService requests, processing of the request, and the GETs of BookQueryService responses.

#### FTP Poller with User Agent for Username and Password

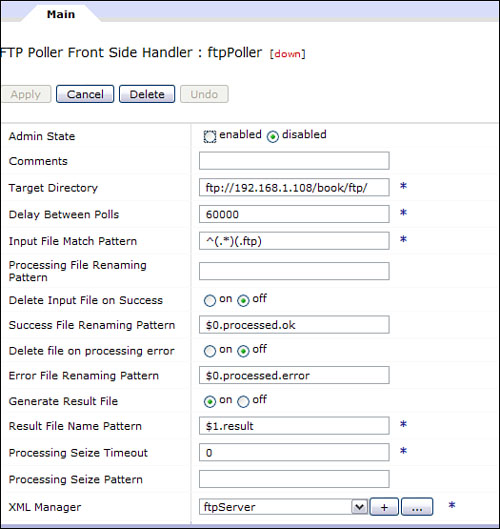
Add a note hereThis example demonstrates the extraction of a document from an FTP server via an FTP Poller FSH as seen in Figure 9-12. An XML Manager with an associated User Agent will be utilized to provide credentials to the FTP server. The User Agent is contained as a property of the XML Manager. The XML Manager is defined on the FSH. We’ll see exactly how this is done shortly.

[](javascript:PopImage('IMG_172','http://images.books24x7.com/bookimages/id_30903/09fig12_alt.jpg','584','205'))  
Add a note hereFigure 9-12: FTP Poller with User Agent example.

##### Front Side Handler

Add a note hereThe FTP Poller Front Side Handler can define the access and control the fetching of documents from an external FTP server. In this case, the external FTP server is located at [ftp://192.168.1.102/book/ftp/,](ftp://192.168.1.102/book/ftp/) which is used as the target Directory URL. The default FTP server listening port of 21 is assumed. Documents that match the regular expression “^(.\*)(.ftp)” (all documents that end in ftp) in the target directory are fetched from the server.

Add a note hereThe FSH can control whether files are deleted or renamed during successful and failed operations. Figure 9-13 shows several of the associated properties. The “$1” in the property values refers to the first component (designated by the first pair of parentheses) of the name of the file that was fetched. So, as the property Success File Renaming Pattern contains $1.processed.ok, the file is renamed with a processed.ok suffix on successful completion. When documents are fetched from the FTP server, they are temporarily renamed so that other clients do not fetch them while they are being processed. This is referred to as the “seize” mechanism. Properties on the FSH can control this behavior, though the default functionality should work in most cases and is used here. The renaming process uses a pattern of filename.serial.domain.poller.timestamp (renamed input file, serial number of the appliance, domain of the polling object, the name of the polling object, timestamp), and with the default case, the filename portion will remain the same as the original request.

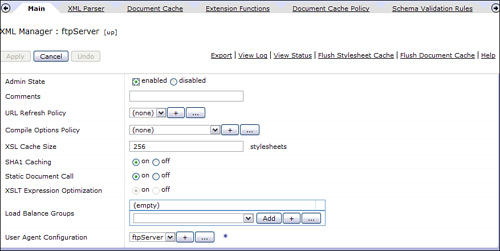
[](javascript:PopImage('IMG_173','http://images.books24x7.com/bookimages/id_30903/09fig13_alt.jpg','525','555'))  
Add a note hereFigure 9-13: FTP Poller FSH with XML Manager defined.

##### Use of User Agent to Provide Passwords

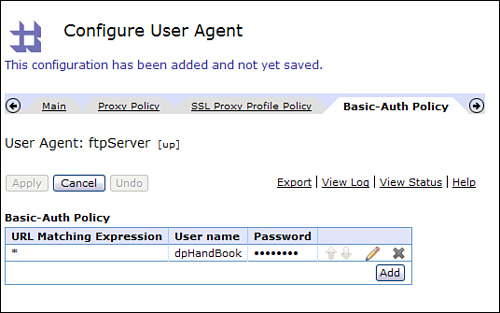
Add a note hereThe particular FTP server in use in these examples requires authenticated access. We could have coded the username and password on the FSH target Directory URL, [ftp://user:password@192.168.1.102/book/ftp/.](ftp://user:password@192.168.1.102/book/ftp/) However, this would have the undesirable consequence of displaying this potentially sensitive information in log files.

Add a note hereAn alternative and more secure method would be to use a User Agent to supply these credentials. A User Agent is defined as an object that makes a request on the behalf of another process.

Add a note hereIn this case, as you can see in Figure 9-14, the User Agent is defined on the XML Manager object identified on the FTP Poller FSH. The XML Manager controls a number of other functions, such as caching mechanisms, and can be defined at several other levels in the DataPower configuration. Figure 9-14 shows the XML Manager and the User Agent that have been created.

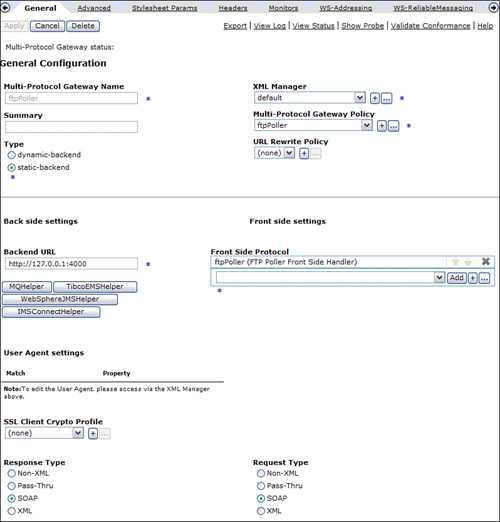
[](javascript:PopImage('IMG_174','http://images.books24x7.com/bookimages/id_30903/09fig14_alt.jpg','877','440'))  
Add a note hereFigure 9-14: XML Manager with User Agent.

Add a note hereIn this case, the User Agent Basic-Auth Policy is utilized to inject username and password information. Notice that the wildcard of \* has been used in Figure 9-15. Other match expressions could have been configured to provide a fine-grained credential policy. This policy simply replaces the username and password for all requests.

[](javascript:PopImage('IMG_175','http://images.books24x7.com/bookimages/id_30903/09fig15_alt.jpg','536','335'))  
Add a note hereFigure 9-15: User Agent with Basic-Auth Policy.

##### The MPGW Service

Add a note hereHaving defined the FSH, XML Manager, and User Agent, the configuration of the MPGW is a straightforward process. The Front Side Protocol list as shown in Figure 9-16 displays all handlers previously created. (We could create additional handlers from this screen.) The backend URL is entered as <http://127.0.0.1:4000> (the BookQueryService XMLFW). The 127.0.0.1 IP address is used to identify the local appliance. As this service will be receiving and returning SOAP documents, the Request and Response Types have been set to SOAP.

[](javascript:PopImage('IMG_176','http://images.books24x7.com/bookimages/id_30903/09fig16_alt.jpg','798','833'))  
Add a note hereFigure 9-16: ftpPoller MPGW.

##### Testing of FTP Poller MPGW

Add a note hereTo test our service, a request document ending in .ftp is placed in the directory identified in the FSH. It is fetched by the FSH from the directory as displayed in Listing 9-5, processed by the MPGW, and submitted to the backend XMLFW. A response document is returned and named according to the Result File Name Pattern specified on the FSH.

Add a note hereListing 9-5: ftpServer PUT Execution and Resulting File

Add a note hereDirectory of \ftp

03/11/2008 08:00 AM <DIR> .

03/11/2008 08:00 AM <DIR> ..

03/06/2008 08:17 PM 401 bookQueryRequest.ftp

1 File(s) 401 bytes

2 Dir(s) 14,865,592,320 bytes free

Add a note hereAfter the bookQueryRequest.ftp document is processed, input file and result file processing is determined by the properties selected on the FTP FSH. We have control over the potential deletion or renaming of the input and the naming of results returned. This example specified the properties shown in Table 9-2.

| Add a note hereTable 9-2: FSH Document Control Properties  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=466966750&objectid=ch09table02&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=466966750&objectid=ch09table02&objecttype=spreadsheet) | |
| --- | --- |
| **Add a note hereProperty Name** | **Add a note hereValue** |
| Add a note hereDelete Input File on Success | Add a note hereOff |
| Add a note hereSuccess File Renaming Pattern | Add a note here$1.processed.ok |
| Add a note hereGenerate Result File | Add a note hereOn |
| Add a note hereResult File Name Pattern | Add a note here$1.result |
| Add a note hereDelete File on Processing Error | Add a note hereOff |
| Add a note hereError File Renaming Pattern | Add a note here$1.processed.error |

Add a note hereAccording to the “Success File Renaming Pattern” property established on the FTP FSH, the client request document is renamed with a .processed.ok suffix. If there has been an error, the input file is not deleted, and if successful, the response document is renamed with a .result suffix. The results of processing of the request are displayed in Listing 9-6 and show that our request was processed successfully!

Add a note hereListing 9-6: FTP Directory After Poll

Add a note hereDirectory

03/24/2008 03:36 PM <DIR> .

03/24/2008 03:36 PM <DIR> ..

03/06/2008 08:17 PM 401 bookQueryRequest.processed.ok

03/24/2008 03:36 PM 630 bookQueryRequest.result

2 File(s) 1,231 bytes

2 Dir(s) 11,883,442,176 bytes free

Add a note herePrinting of the response document indeed shows that our intended request was received. The request for the BookQueryService was fetched by the FTP Poller FSH, submitted to the backend URL, and the response was put onto the FTP server. Listing 9-7 shows the response from the BookQueryService XMLFW.

Add a note hereListing 9-7: Display of Response Document from FTP

Add a note heremore bookQueryRequest.result

<?xml version='1.0' ?>

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

<SOAP-ENV:Body>

<book>

<name>Moby Dick</name>

<author>Herman Melville</author>

<publisher>Longman</publisher>

<isbn>0321228006</isbn>

<copyright>2007</copyright>

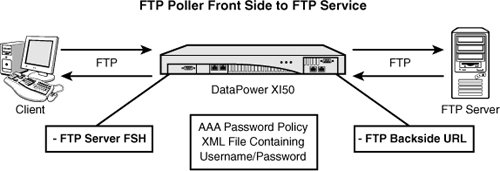
</book>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

#### FTP Server with Transparent File System and Password AAA Policy

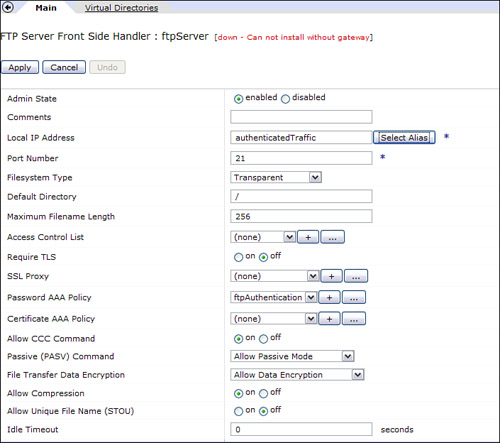
Add a note hereIn this example, an FTP Server FSH is used to control access to an FTP server residing off the appliance. DataPower acts as a proxy to the FTP server and authenticates requests to it. This is done by using the Password Policy option of the FSH. The client signs on to the FTP Server FSH using a username and password. The FTP FSH uses a AAA Policy for authentication. AAA Polices do much more than authentication, and you can learn more about AAA Policies in [Chapter 16](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926), [“AAA.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926) However, for our discussion and this example, we match the credentials to an onboard XML document of valid entries. The client supplies a username of dp, and we change that to dpHandBook and send the password that the FTP server expects. Figure 9-17 shows the configured FTP FSH with the Password AAA Policy defined.

[](javascript:PopImage('IMG_177','http://images.books24x7.com/bookimages/id_30903/09fig17_alt.jpg','574','196'))  
Add a note hereFigure 9-17: FTP Server FSH to HTTP Service.

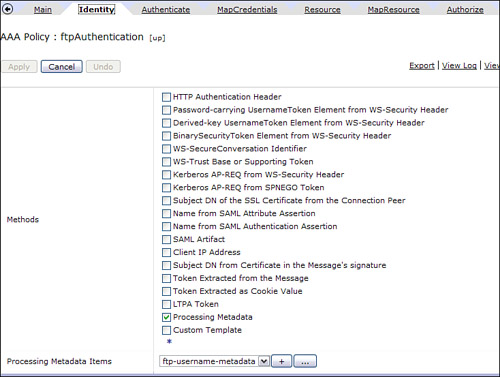
Add a note hereAn important property of the FTP Server FSH is the Filesystem Type. This allows for the utilization of the appliance’s file system or the use of an off device FTP server for the completion of FTP requests. In this fashion, the appliance *can* operate as a self-contained FTP server. However, the use of the appliance’s file system must be done judiciously. This is a finite resource, and the storing of files for FTP access should be managed as such. Acceptable Filesystem Type values are described in Table 9-3.

| Add a note hereTable 9-3: Filesystem Type Values  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=466966750&objectid=ch09table03&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=466966750&objectid=ch09table03&objecttype=spreadsheet) | |
| --- | --- |
| **Add a note hereFilesystem Type** | **Add a note hereDescription** |
| Add a note hereTransparent | Add a note hereThe FTP server FSH has a transparent file system. The files and directories shown are those on the backend FTP server of the associated Multi-Protocol Gateway. |
| Add a note hereVirtual Ephemeral | Add a note hereThe FTP server FSH has an ephemeral virtual file system with subdirectories created by configuration. The contents of this file system are private to an individual FTP control connection to the FTP server. The contents of this file system do not persist after this FTP control connection ends. |
| Add a note hereVirtual Persistent | Add a note hereThe FTP server FSH has a persistent virtual file system with subdirectories created by configuration. The contents of this file system are shared by all FTP control connections to this FTP server with the same authenticated user identity. |

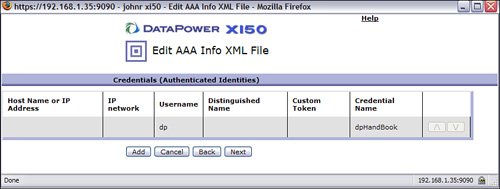
Add a note hereFigure 9-18 shows an FTP Server FSH with the Filesystem Type set to Transparent. This example is intended to use an actual off device FTP Server, and the transparent setting accomplishes that.

[](javascript:PopImage('IMG_178','http://images.books24x7.com/bookimages/id_30903/09fig18_alt.jpg','705','624'))  
Add a note hereFigure 9-18: FTP Server FSH.

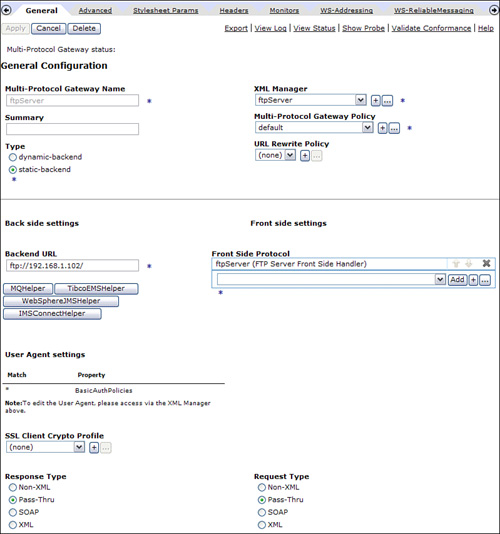
Add a note hereWithin the ftpAuthenticate AAA Policy, there are several phases such as Extract Identity, Authenticate, Extract Resource, and Authorize (see Figure 9-19). In the Extract Identity phase, selecting Processing Metadata, and ftp-username-metadata gets the client credentials (the username and password they signed on with) as we require.

[](javascript:PopImage('IMG_179','http://images.books24x7.com/bookimages/id_30903/09fig19_alt.jpg','734','553'))  
Add a note hereFigure 9-19: AAA identity extraction.

Add a note hereIn the Authenticate phase, an XML AAA Info File is used for credential matching. We match the username provided with dp, and to assign this username a new value of dpHandBook. The AAA Info entry seen in Figure 9-20 is all that is required. This causes all clients whose username is dp to be assigned the new credential value that is used to authenticate to the actual FTP server. Once again, please refer to [Chapter 16](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926) for detailed information on the use of AAA and the XML File in particular.

[](javascript:PopImage('IMG_180','http://images.books24x7.com/bookimages/id_30903/09fig20_alt.jpg','848','321'))  
Add a note hereFigure 9-20: AAA Info File.

Add a note herePutting it all together in the MPGW is again a straightforward process. Figure 9-21 shows the completed MPGW, complete with the XML Manager, backend URL, and FTP Server FSH. Of note here is the use of Pass-Thru on the Request and Response Type settings. As we will be supporting a variety of FTP commands, not all of which may be XML, we will not be parsing the documents. Pass-Thru simply sends the documents to the backend without processing. We could have created a policy by using the Non-XML setting and performed many other functions such as differentiating on the file being requested or posted and performing other authentications based on that information.

[](javascript:PopImage('IMG_181','http://images.books24x7.com/bookimages/id_30903/09fig21_alt.jpg','799','854'))  
Add a note hereFigure 9-21: ftpServer MPGW.

##### Testing the FTP Server Service

Add a note hereTo test the service, a client opens an FTP connection to the appliance and then signs on. Having successfully authenticated via the AAA Policy, the client can PUT a file onto the off box FTP server. The FTP session is shown in Listing 9-8. Notice that after the user authenticates and PUTs the file, a dir command is executed. The results of this are of the FTP Server’s file system. Again, this is because the filesystem type is transparent.

Add a note hereListing 9-8: ftpServer PUT Execution and Resulting File

Add a note hereftp 192.168.1.35

Connected to 192.168.1.35.

220 (IBM WebSphere DataPower)

User (192.168.1.35:(none)): dp

331 password please

Password:

230 user logged in

ftp> PUT fo.soap

200 port acceptable

150 starting store

226 write completed successfully

ftp: 322 bytes sent in 0.00Seconds 322000.00Kbytes/sec.

ftp>

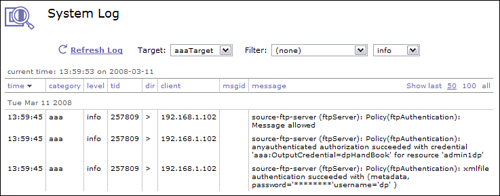
dir fo.soap

03/11/2008 02:05 PM 322 fo.soap

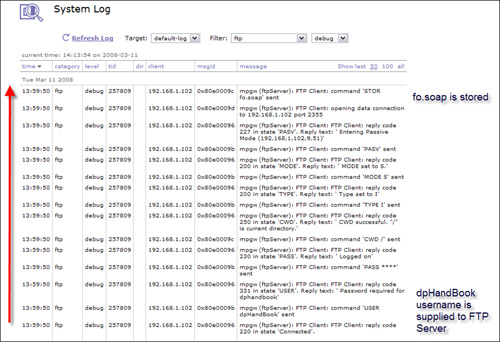
1 File(s) 322 bytes

0 Dir(s) 11,876,028,416 bytes free

Add a note hereViewing the transaction within the DataPower log, (see Figure 9-22) shows the authentication process. You can see that the user logged in with a username of dp, and notice that the password is masked. As this is the username we matched, it is replaced by dpHandBook when authenticating to the FTP server.

[](javascript:PopImage('IMG_182','http://images.books24x7.com/bookimages/id_30903/09fig22_alt.jpg','737','289'))  
Add a note hereFigure 9-22: Log target showing AAA actions.

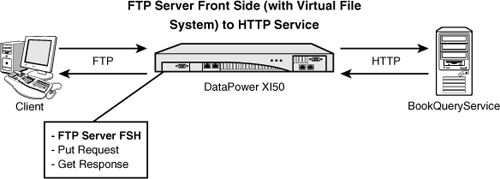
Add a note hereFigure 9-23 shows the default log filtered to only show ftp category commands. The FTP process is informative at the debug level, and you can see a great amount of detail from it. You can see the complete interaction from the DataPower device to the FTP server from user sign-on to the store of the file onto the FTP server.

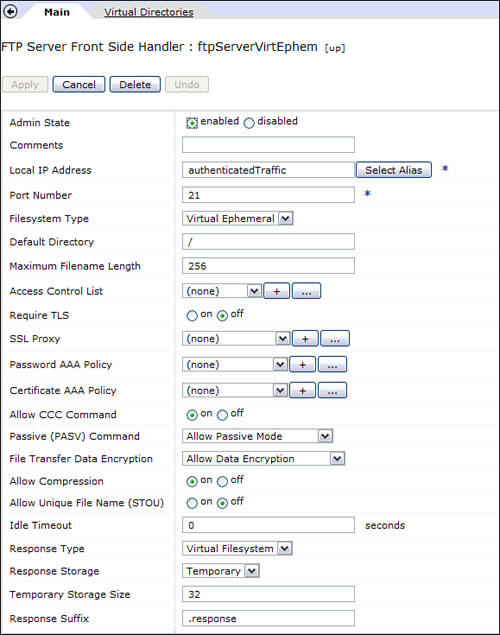
[](javascript:PopImage('IMG_183','http://images.books24x7.com/bookimages/id_30903/09fig23_alt.jpg','931','637'))  
Add a note hereFigure 9-23: FTP Logging data.

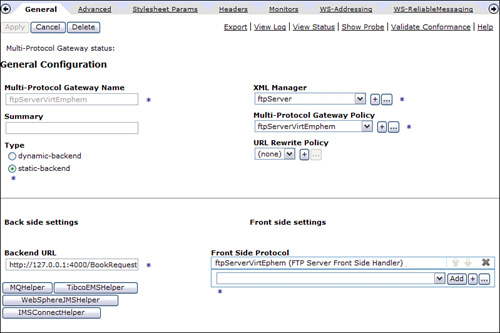
#### FTP Server with Virtual Ephemeral Filesystem

Add a note hereThe next example of FTP processing demonstrates the use of the Virtual Ephemeral Filesystem Type. This allows the DataPower appliance to utilize its own file system in the completion of FTP commands from clients. A virtual ephemeral file system is defined as existing only during the period of the client’s connection. Virtual persistent, on the other, hand maintains the files beyond the connection period. In either case, files may be PUT to and fetched from the appliance. Remember, persistent files consume resources on the appliance! Files processed by the MPGW are processed by the actions specified within the Processing Policy associated with the MPGW.

Add a note hereIn this example, the Processing Policy contains a simple Result action which will submit the request to the backend URL, the BookQueryService XML Firewall (see Figure 9-24). Again, this service will return a response document that will be stored in the virtual file system. To GET the response document, the client will perform a GET FTP command. Figure 9-25 shows the creation of the FTP Server FSH with the Virtual Ephemeral Filesystem selection, and Figure 9-26 shows the creation of the MPGW with the implementation of the FTP Server FSH. Notice the backend URL has been defined as the IP address and port of the BookQueryService XML Firewall service.

[](javascript:PopImage('IMG_184','http://images.books24x7.com/bookimages/id_30903/09fig24_alt.jpg','584','209'))  
Add a note hereFigure 9-24: FTP server to HTTP.

[](javascript:PopImage('IMG_185','http://images.books24x7.com/bookimages/id_30903/09fig25_alt.jpg','579','735'))  
Add a note hereFigure 9-25: FTP server with Virtual Ephemeral File System.

[](javascript:PopImage('IMG_186','http://images.books24x7.com/bookimages/id_30903/09fig26_alt.jpg','799','532'))  
Add a note hereFigure 9-26: MPGW implementing a Virtual Ephemeral FSH.

Add a note hereTesting of the Virtual Ephemeral Filesystem will be done by putting a SOAP request onto the FSH via FTP. This request is processed by the MPGW’s Policy, which submits the request to the BookQueryService XML Firewall. This results in the response from the firewall being placed in the virtual file system. To view the response document, a GET is executed by the client to fetch the response. As the file system is virtual, the request and response are deleted from the device when the user ends the FTP session.

Add a note hereListing 9-9 demonstrates the login, PUT and GET process, and after having quit the FTP session, the response document is displayed. Again, the processing power of the MPGW is demonstrated, this time by acting as an FTP Server!

Add a note hereListing 9-9: Request PUT, Response GET from Appliance

Add a note here\ftp>ftp 192.168.1.35

Connected to 192.168.1.35.

220 (IBM WebSphere DataPower)

User (192.168.1.35:(none)): dp

331 password please

Password:

230 user logged in

ftp> put bookQueryRequest.xml

200 port acceptable

150 starting store

226 write completed successfully

ftp: 246 bytes sent in 0.00Seconds 246000.00Kbytes/sec.

ftp> dir

200 port acceptable

150 starting directory listing

--w--w--w- 1 dp dp 237 May 2 11:44

bookQueryRequest.xml

-r--r--r-- 1 dp dp 253 May 2 11:44

bookQueryRequest.xml.response

226 Directory send OK.

ftp: 165 bytes received in 0.00Seconds 165000.00Kbytes/sec.

ftp> get bookQueryRequest.xml.response

200 port acceptable

150 starting file retrieve

226 response file read successfully

ftp: 261 bytes received in 0.00Seconds 261000.00Kbytes/sec.

ftp> quit

221 goodbye

ftp>more bookQueryRequest.xml.response

<?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>Longman</publisher>

<isbn>0321228006</isbn>

<copyright>2007</copyright>

</book>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

### Add a note hereWebSphere MQ Examples

Add a note hereThe Multi-Protocol Gateway service provides the capability to exchange messages with WebSphere MQ (WMQ) systems by acting as an MQ Client. We see examples of how FSHs and backend URLs are used to GET and PUT messages onto WebSphere MQ Queues. We also introduce a new object, the MQ Queue Manager (QM), which is used to describe the interaction between the appliance and the actual QM. The MQ QM objects are used on the MQ FSH and MQ backend URLs, and we see how they are constructed and utilized in the examples that follow. These examples use a single MQ QM; however, the appliance actually allows for the creation of Queue Manger Groups that can combine multiple QMs in a redundant and highly available architecture. You can read more about QM Groups in the DataPower WebGUI Guide.

Add a note hereAs with all other protocol handlers and backend URLs, WebSphere MQ services may be used in conjunction with any of the other supported protocols, and messages traversing a MPGW implementing WMQ may use the standard features of the DataPower Processing Policies to implement message enrichment, security, dynamic routing, logging, orchestration, or any of the other available actions. The examples that follow demonstrate some fundamental features of the DataPower MQ interaction, such as:

* Add a note hereOne-way and request-response message patterns
* Add a note hereTransactionality or the implementation of a Unit of Work pattern to control the message behavior in error situations
* Add a note hereRouting or the dynamic selection of MQ QM attributes such as replyToQueue, and the interaction with MQMD Headers in the assigning of these values

Add a note hereBackend MQ resource functionality is implemented in the use of a backend URL. As in the case of other backend URL implementations, various parameters may be utilized to fine-tune the MQ interaction. The following examples illustrate how the MPGW provides a helper function (MQ Builder) that makes the creation of the backend URL a simple process; however, knowing the details helps in understanding the URL. These URLs use the DataPower protocol string dpmq, which is used to designate the DataPower MQ client protocol. The use of parameters controls features such as messaging patterns (one-way, request-response), timeouts, and synchronization. Lower-level MQ features can also be set using the PUT Message Options (PMO) parameter. PMO parameters are used to customize the process of putting messages onto a queue. Examples include forcing the PUT operation to fail if the queue is “quiescing,” or closing. You should consult your MQ documentation for a complete list of options.

Add a note hereListing 9-10 shows an example of a dpmq URL, and Table 9-4 contains an explanation of the possible parameters.

Add a note hereListing 9-10: Sample dpmq URL

Add a note heredpmq://QueueManager/URI?RequestQueue=PUTQ;ReplyQueue=GETQ;Sync=true;

Model=true;Timeout=1000;PMO=2048

| Add a note hereTable 9-4: DPMQ URL Structure  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=466966750&objectid=ch09table04&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=466966750&objectid=ch09table04&objecttype=spreadsheet) | |
| --- | --- |
| **Add a note hereURL Component** | **Add a note hereDescription** |
| Add a note hereQueueManager | Add a note hereThe name of an MQ QM object defined on the appliance. |
| Add a note hereRequestQueue | Add a note hereSpecifies the required name of the MQ request queue that receives the message. |
| Add a note hereReplyQueue | Add a note hereOptionally specifies the name of the MQ reply queue that contains the message the service retrieves as a response. |
| Add a note here**Optional Query Parameters** | |
| Add a note hereSync | Add a note hereInsert Sync=true to ensure the transactional nature of backend. The PUT to the Request Queue is committed upon PUT regardless of specification of Reply Queue. |
| Add a note hereTransactional | Add a note hereInsert Transactional=true to ensure the transactional nature of backend. If there is both a Request Queue and Reply Queue specified, the PUT to the Request Queue is committed when a message is retrieved from Response Queue. If there is only a Request Queue it is committed upon PUT. |
| Add a note hereSetReplyTo | Add a note hereInsert SetReplyTo=true to automatically set the MQMD ReplyToQ to the value specified as the ReplyQueue of the dpmq URL. If disabled, the MQMD ReplyToQ must be manually set via a header injection. |
| Add a note hereModel | Add a note hereCan be true or false. When true, instructs the DataPower appliance to connect to the RequestQueue indicated and use the dynamic, temporary Model queue defined by the ReplyQueue value to GET the response. When the response is received, the connection to the temporary queue is closed. This advanced option is seldom used. Refer to WebGUI Guide for additional details. |
| Add a note hereTimeout | Add a note hereSets a timeout, in milliseconds, for the transaction. |
| Add a note herePMO | Add a note hereOptionally sets MQPMO.Options. |

Add a note hereWhile the dpmq protocol string provides the vast majority of features that will be required of MQ message exchanges, you can also construct a dynamic MQ URL. This backend URL format can dynamically define MQ QM attributes. This is not necessary in cases where the backend URL is statically defined on the MPGW service but could conceivably be valuable in dynamic routing situations where the MQ Server is unknown at service definition time, for example, if the destination QM is embedded in the message or needs to be looked up from some other resource. An example of this URL is shown in Listing 9-11, and additional parameters for this protocol are detailed in Table 9-5.

Add a note hereListing 9-11: Sample mq URL

Add a note heremq://ipAddress:Port/example?Channel=chanName;QueueManager=qmName;

UserName=uName;ChannelLimit=maxChannels;ChannelTimeout=timeout;

RequestQueue=requestQueueName;ReplyQueue=replyQueueName

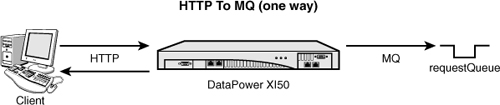
| Add a note hereTable 9-5: MQ URL Structure  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=466966750&objectid=ch09table05&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=466966750&objectid=ch09table05&objecttype=spreadsheet) | |
| --- | --- |
| **Add a note hereURL Component** | **Add a note hereDescription** |
| Add a note hereipAddress | Add a note hereThe dotted decimal address or hostname of the MQ server. |
| Add a note herePort | Add a note hereThe port of the MQ server. |
| Add a note here**Optional Query Parameters** | |
| Add a note hereChannel | Add a note hereThe name of the channel used to access the MQ QM. |
| Add a note hereQueueManager | Add a note hereThe name of the target MQ QM. You can optionally use the name of a QM Group to implement failover. |
| Add a note hereUserName | Add a note hereThe plaintext string used to identify the client to the MQ server. |
| Add a note hereChannelLimit | Add a note hereAn integer within the range 2 to 5000 that specifies the maximum number of open channels allowed between the appliance and MQ QM. |
| Add a note hereChannelTimeout | Add a note hereAn integer specifying the number of seconds that inactive channels will be cached (remain open)—the dynamic channels will be cached until they timeout—after all channels have timed out, the dynamically defined QM is deleted from the appliance. |
| Add a note hereRequestQueue | Add a note hereThe name of the queue to PUT messages to. |
| Add a note hereReplyQueue | Add a note hereThe optional name of the queue to GET reply messages from. |

Add a note hereNow that we have an understanding of the objects required to GET MQ messages and PUT MQ responses, the mechanics of sending messages to an MQ Queue and the role that the MQ QM object plays in tying together the DataPower appliance’s MQ Client with an actual MQ QM, it’s time to see some real-world examples:

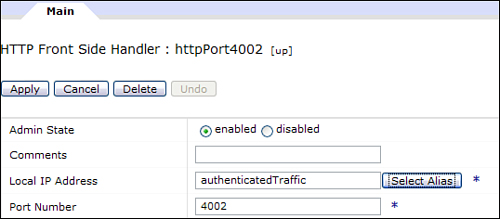
* Add a note here**An HTTP to MQ Protocol Bridge—** This example demonstrates the ability to accept a message over HTTP and place it on an MQ queue in a one-way message model.
* Add a note here**Request Queue fetching and processing with Transactionality—** This example demonstrates the ability to use an MQ FSH to process BookQueryService requests, and the implementation of a Unit of Work to control the removal of messages from the request queue.

#### HTTP to MQ Pattern

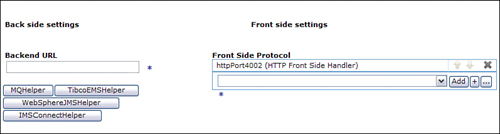
Add a note hereFigure 9-27 demonstrates the architecture of the HTTP to MQ Protocol Bridge. Clients send messages over HTTP to the DataPower MPGW service. The MPGW then delivers the message over MQ to a specific queue.

[](javascript:PopImage('IMG_187','http://images.books24x7.com/bookimages/id_30903/09fig27_alt.jpg','598','126'))  
Add a note hereFigure 9-27: HTTP to MQ one-way pattern.

Add a note hereThe first step in this use case is to define an HTTP FSH that exposes traffic (we’ll use port 4002) to the service. This is straightforward and simple as demonstrated previously. Note that you can create the FSHs before beginning the creation of the MPGW, or in conjunction with the MPGW creation. As previously discussed, threat protection mechanisms can be implemented on the HTTP FSH. Figure 9-28 shows the FSH being created.

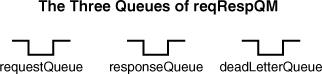
[](javascript:PopImage('IMG_188','http://images.books24x7.com/bookimages/id_30903/09fig28_alt.jpg','538','236'))  
Add a note hereFigure 9-28: HTTP FSH on port 4002.

Add a note hereFigure 9-29 shows the FSH drop-down box after httpPort4002 FSH has been added. Notice also the MQHelper button below the backend URL. We use this when we define the backend MQ resource. It will build the dpmq protocol URL required for backend communication.

[](javascript:PopImage('IMG_189','http://images.books24x7.com/bookimages/id_30903/09fig29_alt.jpg','797','213'))  
Add a note hereFigure 9-29: FSH being added.

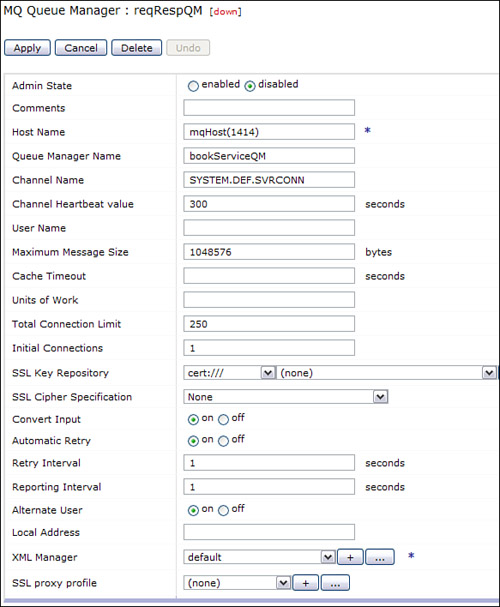
Add a note hereNext, the MQ QM object must be defined. Again, you can build this object in conjunction with the MPGW creation, or you can create it prior to the MPGW. Because this object can be shared across multiple services and FSHs, it is often constructed first to ensure appropriate connectivity. After defining with the correct parameters, it can be reused without modification. It is helpful to create it first, and to ensure that the object is properly enabled prior to beginning the MPGW configuration. This allows you to resolve any network issue, such as an inaccurate IP:port, first.

Add a note hereThis service integrates with an actual WebSphere QM that contains three queues, one from which to GET requests (requestQueue), one on which to PUT responses (responseQueue), and one to place undeliverable messages (deadLetterQueue). Figure 9-30 shows the three queues.

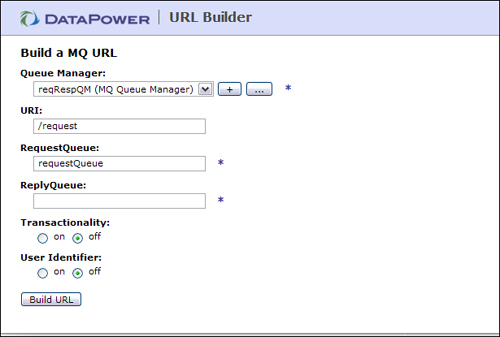
  
Add a note hereFigure 9-30: ReqRespQM, three queues.

##### MQ Queue Manager

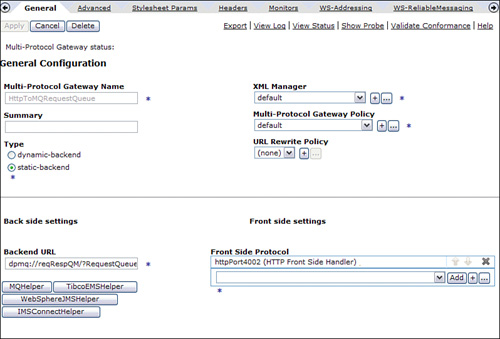
Add a note hereThe QM object is depicted in Figure 9-31, and has only one required field: the hostname or IP address of the WebSphere QM. Notice that the port is assigned within parentheses. If the QM desired is not the default for the host identified, a QM Name can be entered. The default channel SYSTEM.DEF.SVRCONN is used. Among the other parameters are those allowing for the assignment of SSL connection details for accessing MQ over SSL, credential information, and connection limits. We also demonstrate in the second MQ example, the Unit of Work parameter that controls whether transactionality, or the rolling back of messages in the event of transaction failure, is implemented. We see how this QM is associated with Front Side Protocol Handlers and backend URLs to perform MQ message processing.

[](javascript:PopImage('IMG_191','http://images.books24x7.com/bookimages/id_30903/09fig31_alt.jpg','583','708'))  
Add a note hereFigure 9-31: Queue Manager object.

Add a note hereNow let’s put the FSH and MQ QM to work on the MPGW. We’ve already discussed how to add the FSH; now let’s look at the backend URL. You saw the MQ Helper button in Figure 9-29, and how it makes the job of constructing a dpmq:// URL simpler. We need an MQ backend as our intention is to place a message on an MQ queue. Clicking the MQ Helper button reveals a dialog box, as seen in Figure 9-32, that requests basic information such as the MQ QM to use and the queues to use for request and response. The response queue is optional, and not assigning one is typical of a “fire and forget” one-way message pattern—just what we want!

[](javascript:PopImage('IMG_192','http://images.books24x7.com/bookimages/id_30903/09fig32_alt.jpg','615','415'))  
Add a note hereFigure 9-32: DPMQ URL Builder.

Add a note herePutting all the pieces together on the MPGW completes the creation of the HTTP to MQ Service. SOAP requests are submitted to the MPGW service, which are then forwarded to the MQ Queue, requestQueue. Notice in Figure 9-33 that the request and response types are set to Pass-Thru. It was our stated intention in this example to simply accept messages over HTTP and send them to MQ. No processing of the message is necessary, hence no need to define a Processing Policy (notice that default is selected), and the message can traverse the MPGW without processing. A more robust implementation would include authentication of the client, but this is not our focus here.

[](javascript:PopImage('IMG_193','http://images.books24x7.com/bookimages/id_30903/09fig33_alt.jpg','798','541'))  
Add a note hereFigure 9-33: HTTP to MQ MPGW.

##### Testing the HTTP to MQ Service

Add a note hereWe can test the newly created service by sending a request SOAP document to the MGPW via the HTTP FSH, as demonstrated in Listing 9-12.

Add a note hereListing 9-12: Testing HTTP to MQ Service

Add a note herecurl --data-binary @BookQueryRequest.xml http://192.168.1.35:4002 -v

\* About to connect() to 192.168.1.35 port 4002 (#0)

\* Trying 192.168.1.35... connected

\* Connected to 192.168.1.35 (192.168.1.35) port 4002 (#0)

> POST / HTTP/1.1

> User-Agent: curl/7.16.4 (i586-pc-mingw32msvc) libcurl/7.16.4

OpenSSL/0.9.7e zlib/1.2.2

> Host: 192.168.1.35:4002

> Accept: \*/\*

> Content-Length: 246

> Content-Type: application/x-www-form-urlencoded

>

< HTTP/1.1 200 OK

< X-Backside-Transport: OK OK

< Connection: Keep-Alive

< Transfer-Encoding: chunked

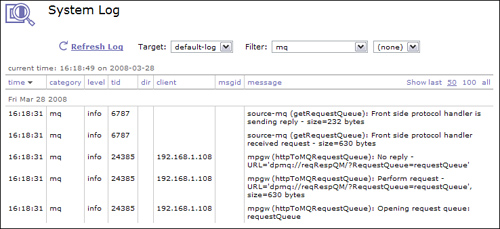
< X-Client-IP: 192.168.1.108

<

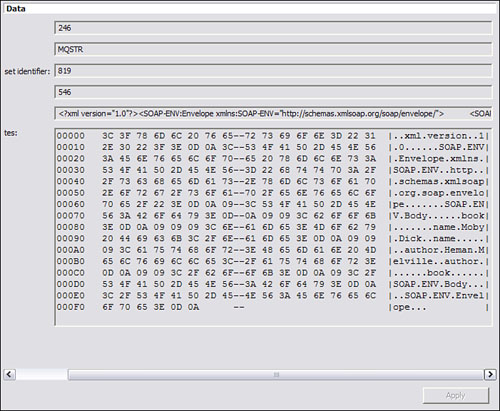
\* Connection #0 to host 192.168.1.35 left intact

\* Closing connection #0

Add a note hereNo response is returned from the service as it is a one-way message pattern. The MPGW replies with an HTTP Status of 200 and with an empty reply body. This can be verified by using the verbose option of cURL, as is shown in Listing 9-12, with the use of the -v parameter. We can construct a response message if we want. However, by viewing the log for MQ events shown in Figure 9-34, we can see the successful processing of the request. Notice that a filter has been applied to the log display only showing MQ event messages.

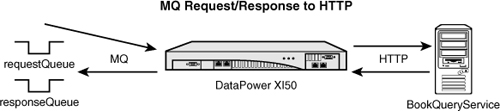
[](javascript:PopImage('IMG_194','http://images.books24x7.com/bookimages/id_30903/09fig34_alt.jpg','738','338'))  
Add a note hereFigure 9-34: HTTP to MQ log.

Add a note hereIn addition, to ensure that our message was delivered to the MQ Queue, we can use MQ Explorer[[2](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=466966750&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ftn.ch09fn01)] to browse the messages on the requestQueue. Sure enough, as we can see in Figure 9-35, our SOAP request was placed on the queue.

[](javascript:PopImage('IMG_195','http://images.books24x7.com/bookimages/id_30903/09fig35_alt.jpg','701','576'))  
Add a note hereFigure 9-35: MQ Explorer viewing messages.

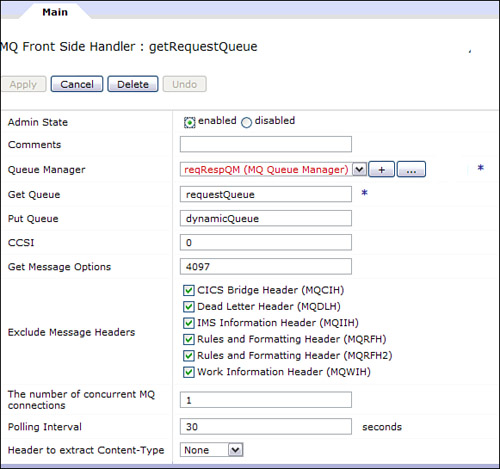
#### MQ Request/Response to HTTP Pattern

Add a note hereFigure 9-36 demonstrates the architecture of the MQ Request Response to HTTP pattern. Messages are fetched from the MQ request queue, and then posted over HTTP to the Book Query Service. The response from the Book Query Service then delivers to the MQ reply queue.

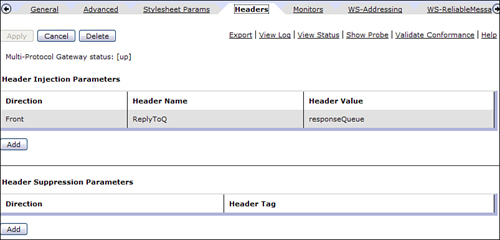
[](javascript:PopImage('IMG_196','http://images.books24x7.com/bookimages/id_30903/09fig36_alt.jpg','583','129'))  
Add a note hereFigure 9-36: MQ Request Response to HTTP.

Add a note hereThe previous example demonstrates a mechanism for the posting of messages onto the MQ queue. Our next example demonstrates a request-response pattern in which an MQ FSH is used to fetch a request message, post it to the BookQueryService XML Firewall, and then if successful, place the response from the XMLFW on the Response queue. In addition, while we can statically assign the reply queue PUT Queue name on the FSH, this example demonstrates a more dynamic methodology of queue name assignment via header injection.

Add a note hereAs you can see in Figure 9-37, our configuration of the MQ FSH does not contain a PUT Queue assignment. It does contain a reference to the DataPower QM object that was previously created. An MQ FSH also contains options for the processing of MQ Headers. Various headers may be included with messages such as Rules and Formatting Headers RFH and RFH2. To have these headers accessible, and not to become part of the message being parsed, the Exclude Message Headers options should be chosen, or the headers will be consumed and parsed as part of the message body.

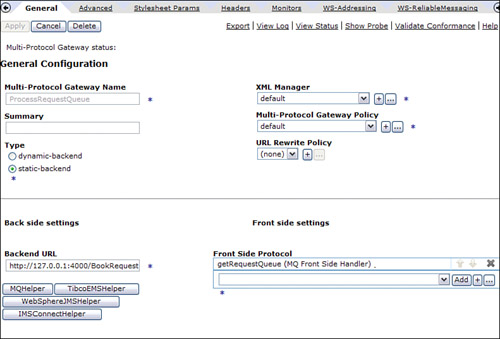
[](javascript:PopImage('IMG_197','http://images.books24x7.com/bookimages/id_30903/09fig37_alt.jpg','581','545'))  
Add a note hereFigure 9-37: MQ FSH.

Add a note hereDataPower provides multiple ways to inject headers into a message stream, including the use of XSLT. We are interested in injecting a ReplyToQ header for the purposes of dynamic queue selection. Although this designation might be better suited to a programmatic XSLT solution in which dynamic queue selection can be performed, this particular example uses the header injection capabilities of the MPGW Header tab. As you can see in Figure 9-38, all that needs to be done is to assign the header name and a value. The direction is specific to either the front (Front Side Handler), or back (the backend URL).

[](javascript:PopImage('IMG_198','http://images.books24x7.com/bookimages/id_30903/09fig38_alt.jpg','803','385'))  
Add a note hereFigure 9-38: Static MQMD header injection.

Add a note hereIf the SetReplyTo option had been utilized on the backend dpmq URL, any ReplyQueue specified would have been entered automatically into the MQMD ReplyToQ header. The SetReplyTo parameter and other dpmq options were discussed earlier in Table 9-4, and the use of the MQ URL Builder as was seen in Figure 9-32.

Add a note hereFigure 9-39 shows the completed MPGW with the FSH assigned and the HTTP backend URL for the posting of the message to the XML Firewall. Again, as our intention here is to perform protocol mediation and not to enrich the message or perform other actions upon the request, there is no need to define a Processing Policy. The request and response types can be set to Pass-Thru for maximum efficiency, but as was mentioned previously, this precludes the ability to apply an authentication of the client request and would normally only be valid for a test or production implementation which does not require authentication.

[](javascript:PopImage('IMG_199','http://images.books24x7.com/bookimages/id_30903/09fig39_alt.jpg','791','537'))  
Add a note hereFigure 9-39: ProcessRequestQueue MPGW.

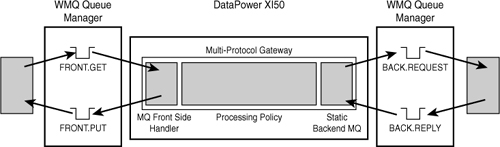
Add a note hereSubmitting another request to the HTTP To MQ service HttpToMQRequestQueue places another request message on the request queue. Now the ProcessRequestQueue service fetches the message, submits it over HTTP to the Book Query service, and places the response on the queue as determined by the ReplyToQ header injection. You can refer back to Figure 9-36 if you want a graphical depiction of this process.

##### Error Processing

Add a note hereThe final MQ example raises the subject of error processing. In this scenario we want to control the behavior of the request messages on the request queue such that messages are removed from this queue only when a successful response is received from the backend service. The Book Query XML Firewall is disabled to simulate a failure.

Add a note hereThis topic introduces the concept of commitment of MQ GET and PUT operations. When a message is fetched from a queue via a GET, it is not physically removed from that queue until a commit is performed via the MQCOMMIT operation. Similarly, when a message is PUT onto a queue, it is not actually available for another operation to GET until the commit is performed.

Add a note hereThe image in Figure 9-40 provides a graphical depiction of a scenario where MQ operations are being performed on both the front side and backend of the MPGW. In this example two Queue Managers are utilized, and while they may in fact be the same QM instance, they could represent two physically independent Queue Managers; in either case, separate connections are utilized.

[](javascript:PopImage('IMG_200','http://images.books24x7.com/bookimages/id_30903/09fig40_alt.jpg','733','215'))  
Add a note hereFigure 9-40: MQ Transaction support

Add a note hereThe DataPower MQ client library provides transactional support on both sides of the MPGW service. There are three properties that affect MQ transactional characteristics: the Units of Work setting on the DataPower MQ QM object, which controls front side synchronization; and two variations of backend synchronization, which are implemented by the Transactional and Sync parameters of the backend dpmq URL.

Add a note hereThe result of synchronization (parameters or MQ properties) is dependant upon the completion of each transaction. This is defined as when the response message is delivered to the client. If there is a reply queue specified on the FSH, transaction completion is defined as when a message is processed by the response rule (if specified) and delivered to FRONT.PUT. If there is no front side reply queue (which is typically associated with one-way messaging and no specification of backend reply queue), the transaction is completed when the initial request message is delivered to BACK.REQUEST.

Add a note hereFront side synchronization is managed via the Units of Work (UOW) property of the MQ QM Object associated with the FSH. Messages retrieved via the FSH from FRONT.GET may be synchronized (by setting UOW to ‘1’) with successful delivery to the backend designation BACK.REQUEST and completion of the transaction. Commitment removes the message from FRONT.GET.

Add a note hereBackend synchronization may be managed via two optional parameters of the backend dpmq URL: Transactional=true and Sync=true. Their implementation share similarities, so it’s important to understand the nuances.

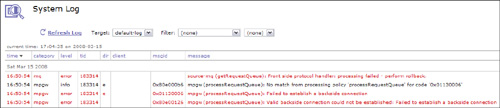
Add a note hereTransactional=true synchronizes the PUT to BACK.REQUEST with the GET from BACK.REPLY. The GET and PUT are committed upon the GET from BACK.REPLY and completion of the transaction. Hence, if there is no reply queue specified, then the PUT to BACK.REQUEST is committed immediately.

Add a note hereSync=true is a less restrictive commit of the PUT to BACK.REQUEST. The PUT will be committed immediately upon successful delivery of the message. This allows backend applications to see messages on BACK.REQUEST so that they may GET and process them for potential delivery to BACK.REPLY. Sync=true must be set if there is both a backend request and reply queue. If not set, the backend application will not see the message placed on BACK.REQUEST.

Add a note hereWhen a transaction fails and a message is left on the FRONT.GET (Front Side Handler’s MQ QM UOW property=‘1’), thought must be given to the effect of leaving this message on the request queue. This results in the message being processed continuously. This is sometimes referred to as a “poison message” and might not be the desired result. In this case, the QM object has three properties to adjust this behavior: Automatic Backout, Backout Threshold, and Backout Queue Name, as seen in Figure 9-41. As the MQ QM object continues to re-GET the message, the backout count increases. After the backout count exceeds the backout threshold, the MQ QM moves the message to the backout queue. If automatic backout is not enabled, the bad message would remain on the GET queue and continue to be reprocessed by the appliance until the QM managing the GET queue removes it or the appliance reroutes the offending message.

[](javascript:PopImage('IMG_201','http://images.books24x7.com/bookimages/id_30903/09fig41.jpg','427','80'))  
Add a note hereFigure 9-41: FSH setting backout queue.

Add a note hereNow when a request is submitted to the requestQueue, the Backend request fails because we disabled the XMLFW service, the message rollback procedure is initiated, and the message is moved to the deadLetterQueue, as seen in Figure 9-42.

[](javascript:PopImage('IMG_202','http://images.books24x7.com/bookimages/id_30903/09fig42_alt.jpg','1135','246'))  
Add a note hereFigure 9-42: Log display showing rollback of request message to request queue.

##### Advanced Messaging Options

Add a note hereThe MPGW fetches messages from request queues, delivers to backend services, and processes messages from response queues in a highly optimized fashion. Depending on the latency of a particular message, the order of processing may not always be serial in nature. This may result in messages being processed by request rules, sent to backend services and subsequently processed by response rules in an order differing from their original order in the source queue.

Add a note hereThis processing can be altered to maintain message ordering. The Advanced tab of the MPGW offers three options under Message Processing Modes: Request Rule in Order, Backend in Order, and Response Rule in Order. These options will ensure that each sequence of processing maintains original message order. Any or all of the options may be selected.

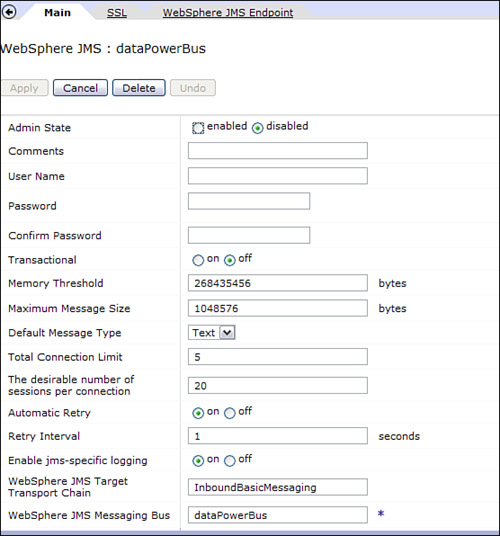
Add a note hereThese processing modes apply to MQ and JMS message processing.

### Add a note hereWebSphere JMS

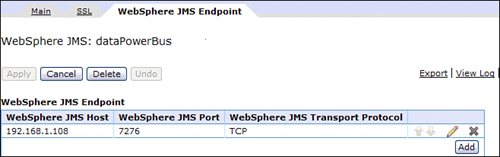
Add a note hereThe DataPower appliance might interact with the default WebSphere JMS providers on the front side and backend of MPGW services. Front Side WebSphere JMS Handlers may be configured to GET and PUT messages on JMS Queues, as well as publish and subscribe to JMS Topics. Backend URLs can be employed to interact with JMS providers on the backend using the dpwasjms protocol string. Support is provided for the WebSphere Default Messaging provider or Service Integration Bus (SIB). JMS Headers may be accessed and set, enabling a mechanism to provide for dynamic queue determination and message correlation, among other JMS options.

Add a note hereSupport for front side and backend JMS processing is enabled by the creation of a WebSphere JMS object on the DataPower appliance. This object encapsulates parameters related to the particular instance of a WebSphere Application Server that will be utilized. Connection variables such as hostname, user credentials, and connection properties can be set. Constraints such as maximum message size and connection limits can also be defined. JMS defines five forms of message bodies: text, byte, stream, map, and object; DataPower supports only the text and byte formats, and does not provide Java object serialization support. Text is the default message type, and should be used for XML messages, whereas byte would be used for other use cases. When processing non-XML data, it will be necessary to use a Non-XML or Pass-Thru setting on the MPGW to avoid a parsing error.

Add a note hereThe minimal configuration of the WebSphere JMS object (as seen in Figure 9-43) requires only the WebSphere JMS Messaging Bus assignment and endpoint information.

[](javascript:PopImage('IMG_203','http://images.books24x7.com/bookimages/id_30903/09fig43_alt.jpg','556','596'))  
Add a note hereFigure 9-43: WebSphere JMS object.

Add a note hereFigure 9-44 shows the WAS endpoint information. The required port value is the SIB\_ENDPOINT\_ADDRESS port of the bus member. Please refer to the JMS documentation for more information on WebSphere JMS Administration.

[](javascript:PopImage('IMG_204','http://images.books24x7.com/bookimages/id_30903/09fig44_alt.jpg','658','206'))  
Add a note hereFigure 9-44: WebSphere JMS object endpoint information.

Add a note hereThe WebSphere JMS FSH provides the information necessary to interact with JMS queues and topics. As in the case of the MQ FSH, the presence of GET and PUT queues can be an indication of the messaging model. For example, one-way behavior is indicated when no PUT queue is described. However, in the case of dynamic queue selection, the reply to queue (PUT Queue), can be programmatically determined. Dynamic queue selection can be done by injecting a JMS header, JMSReplyTo.

Add a note hereIn addition to the standard JMS Headers, three custom DataPower headers are also available as described in Listing 9-13:

Add a note hereListing 9-13: Custom JMS Headers

Add a note hereDP\_JMSMessageType - for the selection of message type as sent to WAS

DP\_JMSReplyToServer - for the dynamic selection of a reply server

DP\_JMSReplyToTopicSpace - for the dynamic selection of a reply to topic

space

Add a note hereAs is the case with MQ headers, these values might be injected into the message stream via the Header tab of the MPGW service, or via the use of DataPower extension functions such as dp:set-response-header. See [Chapter 22](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4078#4078), [“Introduction to DataPower Development,”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4078#4078) for more information on programming in XSLT and the use of the extension functions.

Add a note hereIt is possible to transport JMS messages over an MQ system. The DataPower appliance views JMS messages over MQ as a JMS body and headers within an MQ body and headers. JMS headers such as MQRFH2 can be accessed and injected just as other headers. It is important to note that constraints exist in the construction of these headers; for example, the MQMD.Format field must contain MQFRH2 when it is followed by an RFH2 header. Readers requiring detailed knowledge of this and other constraints should consult the WebSphere MQ documentation.

Add a note hereThe backend URL used to access JMS resources takes the following form shown in Listing 9-14.

Add a note hereListing 9-14: Sample dpwasjms URL

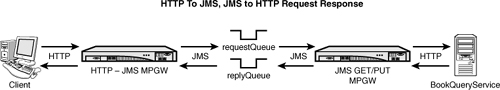
Add a note heredpwasjms://WebSphereJMS?RequestQueue=QueueOrTopicName;   
ReplyQueue=QueueOrTopicName;RequestTopicSpace=TopicSpace;   
ReplyTopicSpace=TopicSpace;Selector=value;TimeOut=value

Add a note hereUsing the WebSphere JMS URL builder makes construction of this string straightforward. The possible values are shown in Table 9-6.

| Add a note hereTable 9-6: WebSphere URL Construction  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=466966750&objectid=ch09table06&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=466966750&objectid=ch09table06&objecttype=spreadsheet) | |
| --- | --- |
| **Add a note hereURL Component** | **Add a note hereDescription** |
| Add a note hereWebSphere JMS | Add a note hereName of an enabled WebSphere JMS object. The object provides the necessary information to access a remote WebSphere JMS provider. |
| Add a note hereRequestQueue | Add a note hereName of the WebSphere JMS request queue. |
| Add a note hereReplyQueue | Add a note hereName of the WebSphere JMS reply queue. |
| Add a note hereRequestTopicSpace | Add a note hereName of the JMS request topic space. |
| Add a note hereReplyTopicSpace | Add a note hereName of the JMS reply topic space. |
| Add a note hereSelector | Add a note hereThe message selector provides an SQL-like expression used to select messages from the reply queue. The message selector is a conditional expression based on a subset of SQL92 conditional expression syntax. JMS headers and properties may be used in message selection statements. |
| Add a note hereTimeOut | Add a note hereInteger that specifies the operational timeout in seconds. |

#### JMS Example

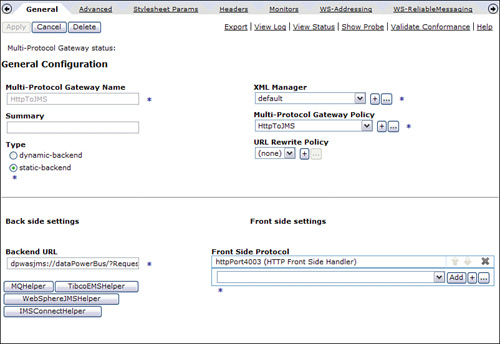
Add a note hereFigure 9-45 demonstrates the architecture of our JMS example. In this use case, a client sends Book Query Requests over HTTP to an MPGW, which then places the request onto a request queue. Another MPGW (assumedly on another device) then fetches the requests from the request queue and makes the HTTP request to the Book Query Service. The response from the Book Query Service is delivered to the JMS reply queue, and then picked up by the originating MPGW and returned to the client.

[](javascript:PopImage('IMG_205','http://images.books24x7.com/bookimages/id_30903/09fig45_alt.jpg','763','137'))  
Add a note hereFigure 9-45: HTTP to JMS, JMS to HTTP request response.

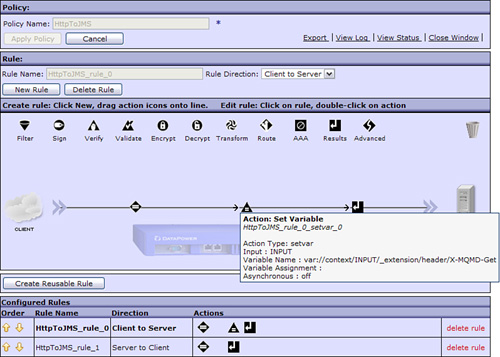
Add a note hereThe following JMS MPGW demonstrates some fundamental features of an MPGW and JMS. First an HTTP to JMS MPGW takes the sample request and PUTs it to a JMS queue. Usually, the response to a message traversing a queuing system is correlated based on the message-id of the original message. In this case, the message-id is moved to the correlation-id of the message placed on the response queue. We can override this behavior and simply GET any message on the response queue by setting a special header named X-MQMD-GET. We can inject this header, or we can set a system variable to produce the same effect. In this example, we’ll demonstrate the latter method.

Add a note hereA second MPGW is configured with a JMS FSH to read the request message from the defined JMS queue and post it to the BookQueryService XML Firewall. It then places the response from this firewall on the response queue. This response is then picked up by the backend URL of the first MPGW and finally returned to the client over HTTP. You can see this flow graphically in Figure 9-45.

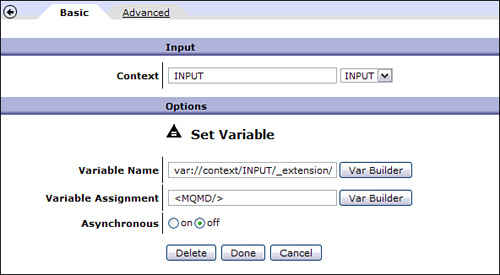
Add a note hereHaving configured the WebSphere JMS object, the WebSphere JMS Builder is used on the MPGW screen to define the backend JMS resource for the HTTP to JMS MPGW. Just as with WebSphere MQ, you can construct the backend URL manually, or as is usually the case, you can use the helper buttons. Notice in Figure 9-46 that an HTTP FSH has been created to use port 4003, and that the backend URL is populated based on the WebSphere JMS Helper button.

[](javascript:PopImage('IMG_206','http://images.books24x7.com/bookimages/id_30903/09fig46_alt.jpg','802','552'))  
Add a note hereFigure 9-46: HTTP to JMS MPGW front and back settings.

Add a note hereWe have stated the goal of fetching any response message from the reply queue. The setting of the correlation ID variable is done in a Set Variable action, as you can see in Figure 9-47, setting var://context/INPUT/\_extension/header/X-MQMD-GET, within the Processing Policy.

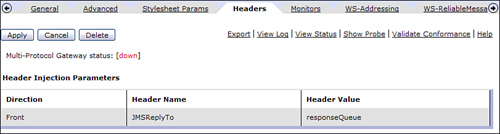
[](javascript:PopImage('IMG_207','http://images.books24x7.com/bookimages/id_30903/09fig47_alt.jpg','798','569'))  
Add a note hereFigure 9-47: JMS SetVar Processing Policy.

Add a note hereThe variable var://context/INPUT/\_extension/header/X-MQMD-GET is set on the Request or Client to Server processing rule, and is set to a null value, or <MQMD/> as seen in Figure 9-48. This allows for the getting of any message on the reply queue, not just one with a message ID equal to the correlation ID of the request message.

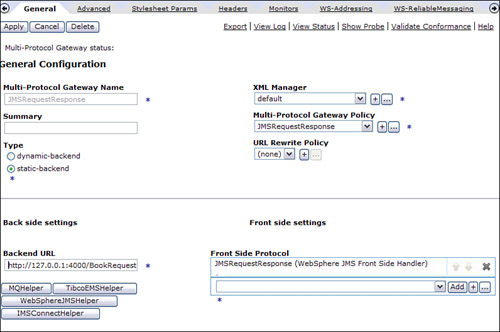
[](javascript:PopImage('IMG_208','http://images.books24x7.com/bookimages/id_30903/09fig48_alt.jpg','592','326'))  
Add a note hereFigure 9-48: Setting var://context/INPUT/\_extension/header/X-MQMD-GET.

Add a note hereHaving constructed an MPGW to accept a message over HTTP and to place it on a JMS queue, the second MPGW will fetch the request message, post it to the backend XML Firewall, and wait for a response. In this case, we dynamically assign the reply-to queue to demonstrate the injection of the JMS header, JMSReplyTo.

Add a note hereThis is done on the Header tab of the MPGW (see Figure 9-49). Again, although we use static values, these actions could have been implemented programmatically with a Transform action using XSLT.

[](javascript:PopImage('IMG_209','http://images.books24x7.com/bookimages/id_30903/09fig49_alt.jpg','811','217'))  
Add a note hereFigure 9-49: Setting Reply-To queue header.

Add a note hereFigure 9-50 shows the completed JMSRequestResponse MPGW. It takes a request from the request queue, sends it over HTTP to the XML Firewall, and waits for a response, which it places on the reply queue. Notice, as was the case for previous examples, this MPGW is not modifying the message, nor performing other actions; there is no need for a Processing Policy. The default policy is selected, and the request and response types have been set to Pass-Thru.

[](javascript:PopImage('IMG_210','http://images.books24x7.com/bookimages/id_30903/09fig50_alt.jpg','796','528'))  
Add a note hereFigure 9-50: JMSRequestResponse MPGW.

##### Testing the HTTP to JMS MPGW

Add a note hereSubmitting the request document to the HTTP to JMS MPGW performs the actions as described in Figure 9-45. Listing 9-15 shows the execution of the request and the response document returned from the MPGW.

Add a note hereListing 9-15: HTTP to JMS to HTTP Test

Add a note herecurl --data-binary @BookQueryRequest.xml http://192.168.1.35:4003

<?xml version="1.0" encoding="UTF-8"?>

<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 hereIt might be helpful to refer back to the diagram in Figure 9-45 to get a better understanding of what is happening on this request. Here are the steps that occur.

1. Add a note hereHTTPtoJMS MPGW receives message from the HTTP client.
2. Add a note hereHTTPtoJMS MPGW places message on request queue.
3. Add a note hereJMSRequestResponse MPGW retrieves message from request queue and posts to BookQueryService XML Firewall.
4. Add a note hereBookQueryService XML Firewall responses with the response document.
5. Add a note hereJMSRequestResponse MPGW places message on reply queue.
6. Add a note hereHTTPtoJMS MPGW retrieves response from reply queue and returns to HTTP Client.

### Add a note hereNFS Support Example

Add a note hereRead and write access is provided to NFS remote servers via both the NFS Poller FSH, and a dpnfs: backend protocol URL string. NFS mounts can be statically or dynamically mounted. Dynamic mounts are constructed via a URL in the form of: “dpnfs://hostname/path/file”, causing the directory hostname:/path to be automatically mounted by NFS. It remains mounted until it times out due to inactivity. Defining a static mount allows for the referencing of the NFS Static Mount object in the URL and avoids the connection overhead associated with dynamic mounting. In addition to supporting FSHs and backend URLs, mounted NFS file systems are exposed as a folder with the appliance’s file system. This allows for NFS endpoints to be utilized in other configuration objects such as the XSLT to be used in a Transform action within a Processing Policy. Keep in mind that if the NFS file system is not highly available, this may not be a good option to use."NFS Endpoints in Transform Actions"

**NFS Endpoints in Transform Actions**

Add a note hereUsing NFS endpoints in Transform actions is not advisable in a production environment.

Add a note hereDue to the possibility of an NFS mount not being available, NFS endpoints should not typically be used in a production environment, unless accessibility is guaranteed.

Add a note hereThe directory display of an NFS Mounted file system is viewable only from the CLI interface, not the WebGUI. So the only way to view files on an NFS mounted file system is to use the CLI, as demonstrated in Listing 9-16.

Add a note hereListing 9-16: NFS File System Display via CLI

Add a note herexi50[book](config)# dir nfs-nfsStaticMount:

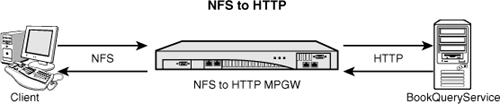
File Name Last Modified Size

--------- ------------- ----

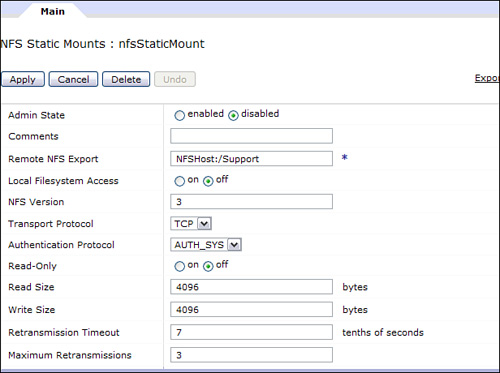
aaaIdentity.xml Wed Feb 28 21:26:15 2007 305

AAAINOUT.txt Thu Jun 21 16:29:37 2007 13223

Add a note hereIn Figure 9-51, an NFS Export is statically mounted and a request document is fetched and processed. The request is sent to the BookQueryService XML Firewall, and the response message is placed onto the NFS file system. This procedure requires the creation of an NFS Static Mount, an NFS FSH, and the MPGW Service itself.

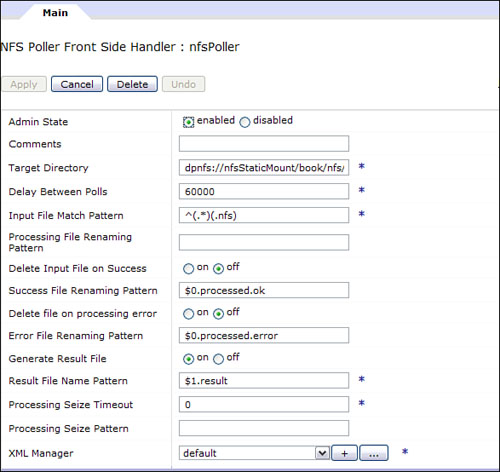
[](javascript:PopImage('IMG_211','http://images.books24x7.com/bookimages/id_30903/09fig51_alt.jpg','556','116'))  
Add a note hereFigure 9-51: NFS to HTTP.

Add a note hereThe NFS Static Mount requires only an NFS-exported file system to mount, in the form host:/path (notice only a single slash is used), where host is the DNS name or IP address of the NFS server, and path is the path exported by the host to mount. In Figure 9-52 the IP address and mount point are defined. This object is navigated to via the default domain Objects menu→Network→NFS Static Mount.

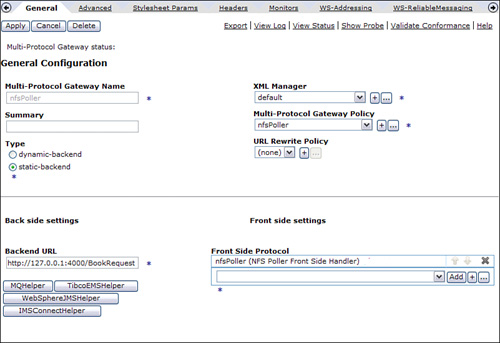
[](javascript:PopImage('IMG_212','http://images.books24x7.com/bookimages/id_30903/09fig52_alt.jpg','616','460'))  
Add a note hereFigure 9-52: Static NFS Mount.

Add a note hereThe NFS server should be configured to accept requests from the IP address of the DataPower device. This example uses AUTH\_SYS authentication, and the NFS server must also be configured to accept that form of authentication. Kerberos could alternately be used for authentication, and it might be a better choice as it provides data integrity and confidentiality if it is supported by the NFS server.

Add a note hereThe NFS Front Side Poller shown in Figure 9-53 is similar to the FTP Poller described earlier in this chapter, and requires a target directory in the form dpnfs://{staticMountName} /{directory below export point}. The input file match pattern is using a regular expression and will extract files with the .nfs suffix from the export directory. Response files are placed onto the directory with name suffixed with .ok, or .error in the case of error occurrence. As was the case with an FTP FSH, there are options for the deletion and renaming of input and result files.

[](javascript:PopImage('IMG_213','http://images.books24x7.com/bookimages/id_30903/09fig53_alt.jpg','587','554'))  
Add a note hereFigure 9-53: NFS Front Side Poller.

Add a note hereCompleting the configuration of the MPGW brings together the NFS Poller FSH, Static Mount, and backend URL construction as seen in Figure 9-54. Again as we are not manipulating the message or performing other actions on the request, there is no need for a Processing Policy and the request and response type has been set to Pass-Thru.

[](javascript:PopImage('IMG_214','http://images.books24x7.com/bookimages/id_30903/09fig54_alt.jpg','801','549'))  
Add a note hereFigure 9-54: NFS Poller MPGW.

#### Testing the Service

Add a note hereAll that is required to test the service is to place the bookQueryRequest.nfs file in the polled directory. The MPGW picks up the request, submits it to the XML Firewall backend, and places the response, named bookQueryRequest.ok, back on the NFS export directory. Listing 9-17 shows this interaction. First the request is copied into the request directory and then the response document is displayed.

Add a note hereListing 9-17: Result from NFS Poller MPGW Test

Add a note herenfs>copy ..\ftp\bookQueryRequest.xml bookQueryRequest.nfs

1 file(s) copied.

more bookQueryRequest.result

<?xml version="1.0" encoding="UTF-8"?>

<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 here[[2](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=466966750&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ch09fn01)]MQ Explorer is a product from IBM for browsing an MQ Queue.

## Summary

Add a note hereThe Multi-Protocol Gateway provides many of the functions of an Enterprise Service Bus architecture. Protocol mediation is accomplished via the use of FSHs and backend URL construction. Message mediation is provided via the implementation of Processing Policies and XSLT transformations.

Add a note hereThe examples in this chapter demonstrated the ease of implementation of the primary protocols that are supported by the MPGW. We started by demonstrating the simplicity of bridging protocols such as HTTP, HTTPS, and MQ with the BookQueryService. Then we extended these examples by the use of additional protocols such as FTP, NFS, and JMS. In each case, the fundamental nature of the MPGW remained the same. All we did was change the type of Front Side Protocol Handler. Some of the examples were quite complex, with messages being picked up from queues, moved from request to response queue, and submitting requests to the XMLFW service.

Add a note hereCan you imagine writing Java code to do all this? Aren’t you glad the Multi-Protocol Gateway made it so easy!