**Chapter 7: Introduction to Services Configuration**

Add a note hereIn the [previous chapter](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=755#755), you were introduced to the concept of a [*service*](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=5268#5268) in DataPower. We also gave an overview of the types of services available on the appliance; for example, the XML Firewall, Multi-Protocol Gateway, and Web Service Proxy. You were introduced to the three primary phases that messages pass through when being processed by the appliance as the requests or responses. You were introduced to objects such as the Processing Policy, rules, actions, and protocol handlers that are at the heart of what DataPower does. In this chapter, we take you further into the details and inner workings of these concepts.

Add a note hereAs you have learned, the services available on DataPower each have their own unique characteristics, such as the Web Service Proxy’s capability to leverage the information in a WSDL file to create a proxy in just a few simple steps and the Multi-Protocol Gateway’s capability to easily use multiple protocol handlers. Despite that, these services share many common objects and interfaces. So, before we start talking about the services individually, let’s learn about these common features and how to use them. Our discussion focuses on the features available in the “big three” services, XML Firewall, Multi-Protocol Gateway, and Web Service Proxy, although much of what you see in this chapter is available on other services such as the XSL Accelerator and Web Application Firewall.

**Add a note here****Backend Types**

Add a note hereWe begin our discussion on the three primary types of backend classifications. By *backend* we mean where the message goes after DataPower has processed it—is the destination known in advance or will it be determined dynamically at runtime? Or perhaps there is none—the appliance returns the response directly to the client. A service can use one of three types of backend destinations:

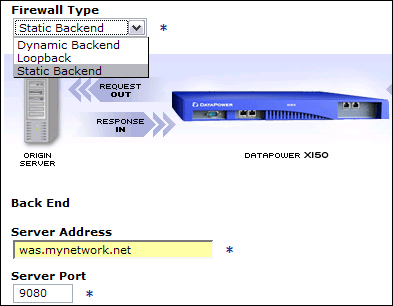
* Add a note here**Static Backend—** Statically determined by the configuration of the service
* Add a note here**Dynamic Backend—** Dynamically determined during runtime
* Add a note here**Loopback—** None at all, in which case the device returns a response to the client without a backend destination

Add a note hereEvery service requires that you specify the type of backend to use. In typical data center environments, these backend server hostnames, ports, URLs, and other attributes might be predefined, such as when DataPower is being set up to proxy applications on a single server. In other circumstances, you might need to make a runtime decision about what server to send the traffic to—perhaps due to current system load or some attribute carried by the messages themselves. Finally, there might be *no* backend server; in this case DataPower will do all processing on the message and send the response to the client.

**Add a note here****Static Backends**

Add a note hereAs the name implies, Static Backends are those whose attributes are known in advance. At configuration time, when you are creating the service on the DataPower appliance, you can simply provide the hostname, port, or other information about the backend server required by your service.

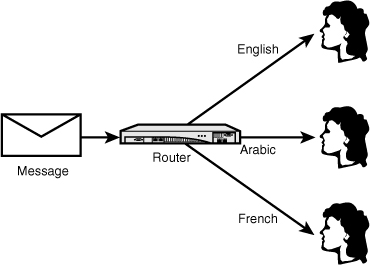
Add a note hereFigure 7-1 shows a typical service configuration for a Static Backend.

  
Add a note hereFigure 7-1: Service backend configuration.

Add a note hereIn some cases, particularly for stateless operations such as Web services, you might want to send messages to a *group* of backend servers to distribute load and facilitate high availability. This is sometimes referred to as *spraying* or *load balancing* messages. DataPower has some capability to act as a load balancer. In this case, you provide the name of a load-balancer object configuration that you have created for the backend address, rather than a single hostname. We show specific load balancer configurations and discuss this in more detail later in this chapter.

**Add a note here****Dynamic Backends**

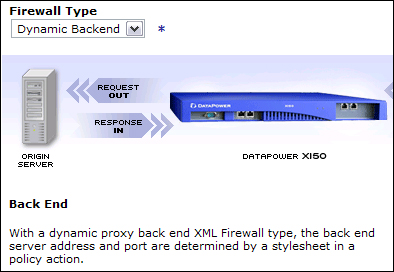
Add a note hereUnfortunately, life isn’t always simple. Business always has and always will entail complex scenarios. Often, we come across scenarios where routing decisions must be made by factoring in runtime considerations. The classic example is that of taking the value of a transaction into account—for example, any incoming purchase order with a total value over one million dollars should be routed to a special destination for high-priority handling. Another might be the scenario shown in Figure 7-2. A worldwide bank customer service system routes intercepted messages to an agent with the appropriate linguistic skills to read the message in the language in which it was written. In fact, the dynamic routing and mediation of messages around an enterprise service bus is a principle that is at the core of SOA. Thus, dynamic routing becomes important.

  
Add a note hereFigure 7-2: Dynamic routing of messages based on language.

Add a note hereTraditionally, dynamic routing has been achieved by placing information in the protocol header, such as a cookie in the HTTP header, or a URL attribute. This has been done because the infrastructure components that must make these content-based routing decisions can only understand data at that layer of the stack. They traditionally cannot look into application-layer data such as SOAP messages. Because of this, we not only must add additional logic to the applications that must be maintained, but we also potentially expose information to those who should not be able to see it. In the networking chapters, you saw how the Ethernet protocol works—message packets are sent to every node on the network, which means such messages can easily be intercepted and read by anyone on the network. This information would be outside the bounds of techniques such as using standards-based encryption for message privacy or digital signatures for message integrity, unless extra steps were taken to encrypt and sign them. In addition, the problem of bloated protocol headers occurs as more and more of this is done.

Add a note hereDataPower appliances can forego these hacks and make routing decisions based on the actual message content. Using XPath, any element or attribute in the XML message can be interrogated to make these decisions. If the routing decision is to be made based on some other data, DataPower has the capability to do so, based on any available information in the message, protocol, or other environmental factors, such as date/time, current backend response time metrics, and so on. 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) and [Chapter 24](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4365#4365), “Real World Examples of XSLT Programming,” you will see dynamic routing in action.

Add a note hereWhen choosing this configuration for your service, the entry fields for the backend server address and port are removed from the page because they are at that point unknown, as shown in Figure 7-3. At some point in the Processing Policy, the routing information must be provided or the transaction will fail. We will show you how to do this as part of the graphical drag-and-drop service policy configuration in [Chapter 8](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1082#1082), and also how to do it programmatically in [Part VI](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4077#4077), “[DataPower Development](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4077#4077).”

  
Add a note hereFigure 7-3: XML firewall Dynamic Backend.

**Add a note here****Loopback**

Add a note hereIn some cases, there might not be a backend, particularly given the broad spectrum of use cases for message processing that can be accomplished using the DataPower appliances directly. We discuss the many available processing actions that can be applied to request and response rules later in this chapter, and you can see many of them in use throughout the book.

Add a note hereAnother typical use for the loopback configuration is simple testing or development when the backends are not available and when creating utility services on the appliance that are meant solely to provide some capability to other services on a localhost interface.

Add a note hereOf course, when choosing the loopback backend configuration, the server address and port fields will also not appear on the service configuration page

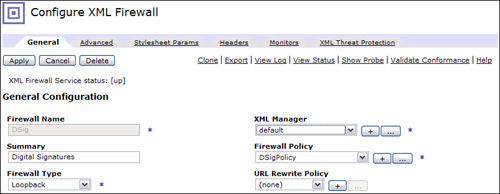
## Supporting Objects

Add a note hereIn the [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) you were introduced to the three major processing phases that occur as messages move through the device as requests and responses. These were the client/front-side area, Processing Policy (sometimes referred to as multistep), and server/back-side area. The most prominent of these is the Processing Policy, which we discuss in detail later in this chapter. Before we do so, let’s look at some of the supporting cast—three objects that can be part of the Processing Policy but also affect what happens in the client or server-side areas.

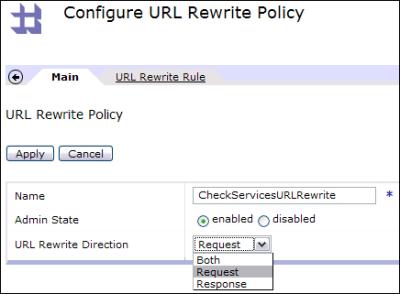
### Add a note hereURL Rewrite Policy

Add a note hereOne highly touted feature of DataPower is the capability to virtualize or hide backend implementation details. The URL Rewrite capability provides one more way to do that. There are other uses for this feature, such as to isolate the client from changes in the proxied backend applications, perhaps to preserve their bookmarks. We can also use this capability to provide a more user-friendly URI than what might exist on our backend applications, for example to expose a friendlier URI such as /MyCheckingAccount, to replace a mundane one such as /bankservices/checking. Let’s develop that example.

Add a note hereThe capability to rewrite URLs shows up in several places throughout the service configuration pages. Figure 7-4 shows where it appears on the XML Firewall main configuration page.

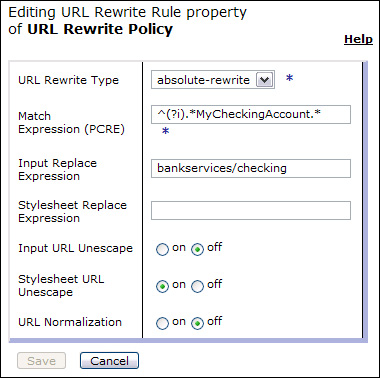
[](javascript:PopImage('IMG_94','http://images.books24x7.com/bookimages/id_30903/07fig04_alt.jpg','805','313'))  
Add a note hereFigure 7-4: URL rewrite policy on the XML Firewall main page.

Add a note hereThe URL Rewrite policy has the capability to rewrite all or part of a URL, replace a header value, or even rewrite the HTTP POST body in a message. To demonstrate this feature, we create a new URL Rewrite Policy to modify the incoming request URL. Figure 7-5 shows the base configuration page for the new object (which can be reached by clicking the “Create a new” or “+” button shown in Figure 7-4). Notice that the rewrite policy can be applied to requests, responses, or for both directions.

[](javascript:PopImage('IMG_95','http://images.books24x7.com/bookimages/id_30903/07fig05.jpg','436','321'))  
Add a note hereFigure 7-5: Creating a new URL Rewrite Policy.

Add a note hereFor requests, these rewrites are applied in the frontside of the processing phases described in [Chapter 6](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=755#755), and we describe how that is important when we get to the section on the Processing Policy later in this chapter. This pane simply asks for a name for this new policy, and which direction it should be applied to.

Add a note hereThe logic of the policy is on the tab shown next to Main, which is URL Rewrite Rule. Moving there, a list of rules can be configured. Figure 7-6 shows a rule that has been configured using Perl-compatible regular expressions (PCRE) to look for any URL (absolute-rewrite) that contains the string MyCheckingAccount and convert that piece of the URL to /bankservices/checking. We are looking for the user-friendly URI from the client and replacing it with what the backend application expects as part of the request path.

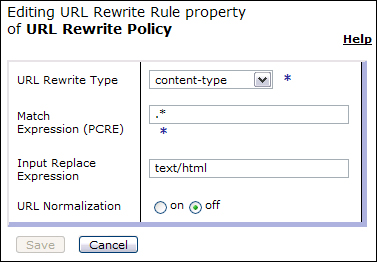
  
Add a note hereFigure 7-6: URL rewrite rule.

Add a note hereWhen a transaction is sent to the device, this policy is enforced whenever the match expression is met. For example, Figure 7-7 shows the log entry for an XML Firewall service that implements our example. Reading the log from the bottom to the top, you can see the incoming transaction has a URI of /MyCheckingAccount and that our URL Rewrite Policy has changed the URI to /bankservices/checking, leaving the rest of the URL intact.

[](javascript:PopImage('IMG_97','http://images.books24x7.com/bookimages/id_30903/07fig07_alt.jpg','790','207'))  
Add a note hereFigure 7-7: PCRE URL rewrite test.

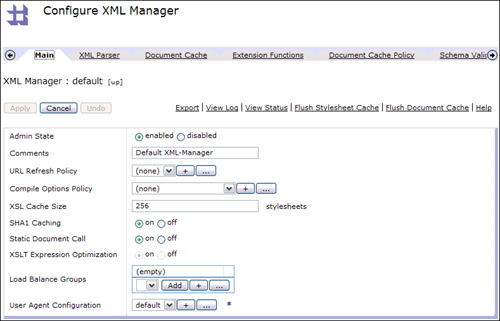
Add a note hereAnother common use of the URL Rewrite Policy is to change the HTTP Content-Type header on-the-fly. Consider a scenario where you receive an XML response from a backend application and transform this into HTML so that it can be rendered in a client’s browser. Along with the transform, our policy must also change the Content-Type header or the browser will be confused. Figure 7-8 shows an example URL Rewrite Policy to do that.

Add a note hereLater in this chapter, we show a more granular approach to rewriting URLs.

  
Add a note hereFigure 7-8: Changing the Content-Type header.

### Add a note hereXML Manager

Add a note here[Chapter 6](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=755#755) briefly mentioned the capabilities of the XML Manager object. Let’s look at the more important features in detail. This object is displayed prominently on the services’ configuration pages. In most cases, the default XML Manager is already assigned, but you might want to create derivatives to do things such as cache more aggressively, load balance, or fire scheduled rules. Figure 7-9 shows the main configuration tab of the XML Manager and some of the associated tabs horizontally across the top.

[](javascript:PopImage('IMG_99','http://images.books24x7.com/bookimages/id_30903/07fig09_alt.jpg','789','507'))  
Add a note hereFigure 7-9: The Configure XML Manager page.

### Tip: Don’t Modify the Default XML Manager

Add a note hereIt might be tempting to make changes to the default XML Manager, which is normally selected. This object is likely to be shared by other services in the domain, and changing it could unexpectedly modify their behavior as well. Even if there are no other services, some might be added later for which the default behavior is appropriate. Always create a new XML Manager, even if it will be shared across all services in the domain, and give it an appropriate name for the modifications being made (such as AggressiveCacheXMLManager, WASClusterLoadBalanceCacheManager).

#### Caching Stylesheets

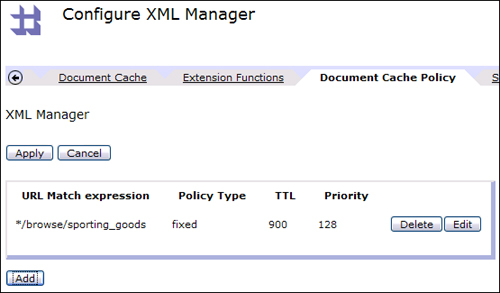
Add a note hereBy default, DataPower caches the XSLT stylesheets used by services (and the device itself) whether they exist locally or are pulled in from remote servers. Figure 7-9 shows a field on the main XML Manager page to configure the size of the cache. Take note that this cache is sized in number of stylesheets so that it could become quite large if the stylesheets being cached are large. The cache size should be monitored and sized appropriately as part of the performance and load testing process. The other option is to use SHA1 Caching, also shown in Figure 7-9. Normally stylesheets are cached by their URL, but if the same stylesheet can be pulled in from different URLs, setting this feature to use a SHA1-computed message digest can improve performance in looking up cache entries.

Add a note hereWhen stylesheets are put into the cache, they remain there until they are pushed out by the policies previously discussed, explicitly flushed (you can see the link to do this in Figure 7-9, or it can be done programmatically), or until the domain or appliance restart. However, what if you would like them to be refreshed on some periodic basis? What if you would like to specify stylesheets that should never be cached or never flushed from the cache? For any of those reasons, you would configure a URL Refresh Policy on the main XML Manager tab. In this area, simple policies can be set up to accomplish those goals based on PCRE expressions and time-based intervals expressed in number of seconds. (Where appropriate, for options like no-cache, you would not supply any time interval.)

#### Caching XML Documents

Add a note hereThe appliance can also cache XML responses to enhance application performance. This feature is application-specific and, therefore, is not active until configured for specific application URLs. Let’s take a look at the capabilities for this feature.

Add a note hereThe XML Manager Document Cache tab contains fields to tune the cache size—a field for the cache by number of documents and one to set the cache size in bytes. In this case, the byte size takes precedence; so if the cache reaches this configured size, you might not fit the maximum number of documents specified in the other field. Of course, this is related to the Document Cache Policy tab on the XML Manager. Figure 7-10 shows an example policy for caching documents that would be responses from a sporting goods catalog browsing URI. A PCRE expression is used to designate the URLs to be subject to the cache. The Policy Type drop-down designates perhaps that these URLs should have no caching done at all, or perhaps the cache size should be Fixed (in which case you must configure the Time To Live field for the number of seconds each cache entry should stay in the cache), or Protocol-Based.

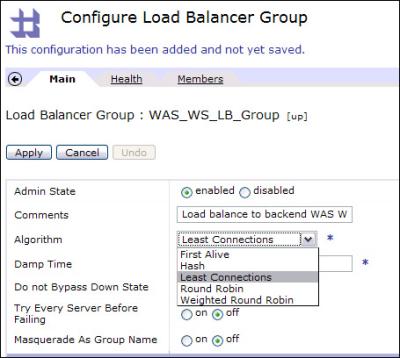
[](javascript:PopImage('IMG_100','http://images.books24x7.com/bookimages/id_30903/07fig10_alt.jpg','555','325'))  
Add a note hereFigure 7-10: XML Manager Document Cache Policy tab.

Add a note hereIn this last case, caching is controlled by virtue of several HTTP 1.0 or 1.1 headers such as Last-Modified, Vary, Cache-Control, and Expires. The WebGUI Guide has details for these settings. Finally, the Priority field allows some entries to have a longer life than others should either the maximum number of documents or cache size in bytes be reached. The priority from 1 to 255 may be entered, with lower numbers getting priority to remain. Ties are broken by a First-In First-Out (FIFO) selection. As with the stylesheet cache, this cache can be flushed by the links that appear on any tab in the XML Manager, or programmatically.

#### Load Balancer Groups

Add a note hereIn our previous section on backend types, we stated that a Load Balancer Group name could be substituted for a hostname or IP address in a backend configuration or URL. This is normally done in high-availability scenarios to distribute stateless requests over some group of peer servers, or to send requests to standby or secondary servers should the primary server be inactive for some reason. Let’s see what’s involved with creating a Load Balancer Group.

Add a note hereOur example simulates a cluster of Web service providers. Figure 7-11 shows that we have selected the Least Connections algorithm for our Load Balancer Group. This algorithm forwards the next connection to the server in the group that has the least number of current connections, based on tables held within the appliance’s memory. The Round Robin algorithm evenly distributes messages to all members, and the Weighted variant allows you to assign weights to each member, perhaps in accordance with their processing power. The First Alive algorithm is used to primarily send traffic to a main server; it sends traffic to the secondary servers further on down the list only if the primary server(s) above are designated as down.

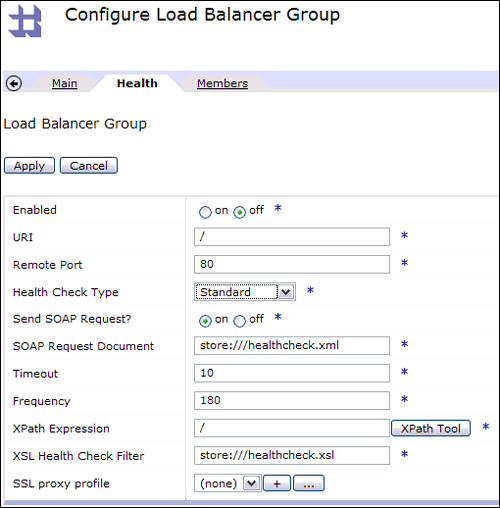
[](javascript:PopImage('IMG_101','http://images.books24x7.com/bookimages/id_30903/07fig11.jpg','455','408'))  
Add a note hereFigure 7-11: Load Balancer Group configuration.

Add a note hereThe Hash algorithm is used to apply stickiness to client messages by keeping a table of hashed IP addresses and which server they have been using. This is typically done when the server has some saved state information for clients, such as in-memory Java EE sessions. Note that this is not a foolproof way of establishing long-term session stickiness because IP addresses can change, and when certain events occur, such as the backend being down, the entire table will be recalculated. DataPower’s load balancing capabilities are best used for stateless operations in keeping with the theme of the appliance’s design.

Add a note hereThe Masquerade As Group Name setting simply determines whether the Load Balancer Group name will be placed in the message header, or the IP address or hostname of the physical server chosen to receive this message from the Load Balancer Group.

Add a note hereThe other settings on this figure relate to the health checking of the members. You may set up a health check policy where each server in the group is occasionally polled to determine whether it is up or down. This is optional, but it would be advisable in order to know that members of the group are inactive before actually sending messages there, if possible.

Add a note hereFigure 7-12 shows the Health tab of the Load Balancer Group. It is off by default. The Standard type of health check shown as selected here is typically used for Web services or Web applications. This sends an HTTP GET (or SOAP message via POST if the Send SOAP Request is turned on) to the remote port and URI defined here. The health checks will be done on the frequency requested, and if a response is not received, the member will be flagged as down.

[](javascript:PopImage('IMG_102','http://images.books24x7.com/bookimages/id_30903/07fig12_alt.jpg','510','518'))  
Add a note hereFigure 7-12: Load Balancer Health Check configuration.

Add a note hereIf you set the Send SOAP Request flag to off, a simple HTTP GET is done, and an HTTP response code of 200 will designate that the server is up. If the Send SOAP Request flag is set to On, a POST will be made using the XML file identified in the SOAP Request Document field. This should be a valid SOAP message. The stylesheet identified in the XSL Health Check Filter is used to check the response using the XPath Expression identified in the corresponding field. The defaults are shown in Figure 7-12—these files already exist on the file system, and you can see that the type of checking done is quite simple—the XPath expression simply looks for a root node in the response. You are free to customize this; in fact, this could be used to do deep health checks, where not only the immediate servers behind DataPower are checked but also layers beyond them. For example, you could do a simple SQL call to see whether a database is up.

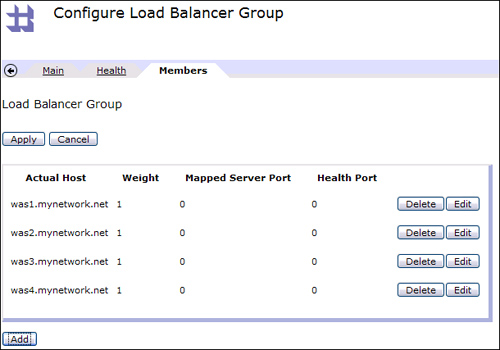
Add a note hereThe alternatives to the Standard health check type are IMS Connect (anonymous bind to IMS servers) and LDAP (anonymous bind to LDAP servers from the AAA configuration, as we will discuss in [Chapter 16](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926), [“AAA,”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926) rather than from inside the XML Manager).

Add a note hereSo what exactly happens when things go wrong? To understand this better, let’s first make sure we understand the three possible states of any Load Balancer Group member:

* Add a note here**Healthy (up)—** The member is considered to be eligible for messages.
* Add a note here**Convalescent (down)—** If a health check fails, a member is placed in this state. The member is removed from the group and does not receive messages. After the Frequency interval elapses and the health check is done again, if the server is deemed to be back, it will be placed into the up state.
* Add a note here**Quarantined (softdown)—** A member is placed into this state when a regular transaction/message (as opposed to a health check) fails. When this occurs, the member is quarantined and will not receive any messages until the Damp Time period has elapsed. At this time, the member will be placed back into service (without any checks as to its health), and messages will attempt to be sent to it. It will also be excluded from health checks during this time, so it is truly quarantined!

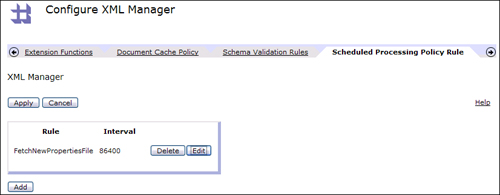
Add a note hereA few last words on health checks: They can be done over SSL (as indicated by the SSL Proxy Profile setting on Figure 7-12), and the health of any individual member can be set directly using the var://service/lbhealth variable. We will discuss just such a scenario in [Chapter 24](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4365#4365).

Add a note hereFinally, you might want to know how we add members to a Load Balancer Group. As Figure 7-13 shows, simply move to the Members tab and define each member by clicking the Add button. You can see that in this figure we have added four members, and that each one has the ability to set the weight (if a weighted algorithm was chosen), set a specific port for the transaction to be sent to, and set a specific port for the health check on each member.

[](javascript:PopImage('IMG_103','http://images.books24x7.com/bookimages/id_30903/07fig13_alt.jpg','610','427'))  
Add a note hereFigure 7-13: Load Balancer Group members.

#### Scheduled Processing Policy Rules

Add a note hereThe Scheduled Processing Policy tab provides a simple-to-configure but powerful capability. Essentially, you can configure any Processing Policy rule (we talk more about this later in the chapter) to be executed on a scheduled interval. These configured rules can provide any processing capability across the entire breadth and depth of the DataPower feature set, making it quite extensible. This means interacting with external resources or triggering some scheduled execution on the appliance itself. Figure 7-14 shows an example of a rule used to execute every 86,400 seconds (or every 24 hours) to fetch a new copy of an onboard properties file, perhaps from an external file system. This might be used at the same time to set the values from that file into global system variables (although these must be carefully used to limit the impact on memory consumption).

[](javascript:PopImage('IMG_104','http://images.books24x7.com/bookimages/id_30903/07fig14_alt.jpg','826','322'))  
Add a note hereFigure 7-14: XML Manager Scheduled Processing Policy rule.

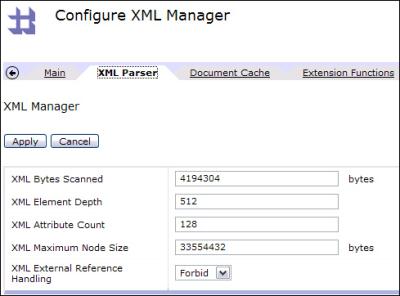
### Tip: Executing Rules Only on Startup

Add a note hereFor rules that should execute only on appliance or domain startup, we could set the interval to zero. This would be useful in the previous example if the properties file was static.

#### XML Parser Limits

Add a note hereThe XML Parser Limits tab of the XML Manager allows for profiling of message characteristics. This is important to protect against XML threats (see [Chapter 20](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3695#3695), [“XML Threats”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3695#3695)) and just run-away data. Often, despite the absence of bad guys, messages sent by our partners or even our own applications can be well-formed XML or valid SOAP but due to bugs or bad transformations have a negative impact on backend system performance. It is a common usage of the DataPower appliances to catch and filter out such messages before they have a destructive effect. Utilizing the raw horsepower of the appliance for these checks, along with stringent schema validation on request *and* response messages (as well as the result of any transformations done on the device) can prevent these types of problems.

Add a note hereAs seen in Figure 7-15, the XML Manager can profile messages by bytes scanned, which includes the parsed message tree and Data Type Definitions (DTDs), maximum nesting depth, size of any nodeset inside a message, and number of attributes. Note that these can be overridden by service-specific settings on the XML Threats Protection tab. We also have the ability on this pane to disable the referencing of external entities such as DTDs and XML Schema Definitions (XSD). Schema-substitution attacks can be thwarted in this manner—the device will look for the files onboard rather than trying to resolve the external reference.

[](javascript:PopImage('IMG_105','http://images.books24x7.com/bookimages/id_30903/07fig15.jpg','489','362'))  
Add a note hereFigure 7-15: XML Manager Parser Limits tab.

#### Compile Options Policy

Add a note hereThe Compile Options Policy object (shown on Figure 7-9) has some powerful capabilities. We can do several things with this object:

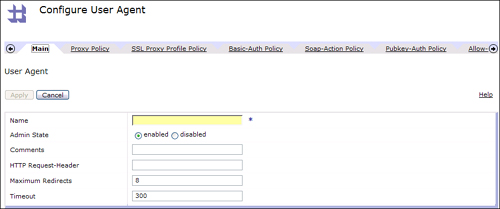
* Add a note hereControlling the compilation of stylesheets by specifying the use of XSLT 1.0, 2.0[[1](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=557360520&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ftn.ch07fn01)] or having the stylesheet determine the version. (This will be covered in more detail in [Part VI](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4077#4077), “[DataPower Development](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4077#4077).”) Policies can also be specified here to check the WSDL against conformance with profiles such as WS-I Basic Profile and to skip or use lax or strict validation of SOAP envelopes and faults or SOAP message bodies and headers.
* Add a note hereMeasuring the processing times for selected stylesheets and reporting on them (intended only for debugging purposes and the appropriate environments—not for high-traffic production sites).
* Add a note hereEnabling streaming mode processing, which enhances the ability to handle large documents.

Add a note hereRefer to [Appendix D](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=5438#5438), “Compile Options Policy Configuration,” in the CLI Reference Guide document (not the first place you would guess to look for this type of information) to help with questions.

### Add a note hereUser Agent

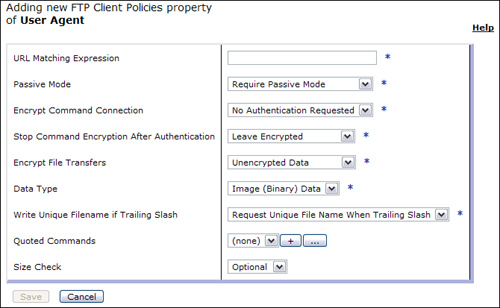
Add a note hereYou might have noticed in Figure 7-9 that the User Agent is contained on the XML Manager main configuration page. However, because it is also used in many other scenarios independent of that, and because it is a big and multifaceted object, we cover it in this separate section.

Add a note hereThe User Agent can be thought of as a utility object to be used by other higher-level DataPower objects. As you will see, it primarily handles the details for network-related outbound calls from the device. Figure 7-16 shows the main configuration tab—only a few fields exist here, but you can witness the large number of other tabs to perform these operations. The fields on the main page are used to send an HTTP request-header with information about the sending User Agent, to specify the maximum number of redirects to attempt before failing the transaction and to choose an idle timeout value to use before closing the connection. (This is unrelated to connection timeouts related to failed transactions.)

[](javascript:PopImage('IMG_106','http://images.books24x7.com/bookimages/id_30903/07fig16_alt.jpg','905','378'))  
Add a note hereFigure 7-16: User Agent Main Configuration page.

Add a note hereIn the following list, we give a brief overview of each tab on the User Agent and its purpose.

* Add a note here**Proxy Policy—** This tab can be used to have requests forwarded to a HTTP proxy server rather than the host they would otherwise be sent to by the normal service configuration. It is configured by providing a URL match and a hostname and port for the proxy server to receive the requests.
* Add a note here**SSL Proxy Profile Policy—** This tab is used to automatically initiate SSL client connections from the device for supported protocols. It is configured by providing a URL matching regular expression and a SSL Proxy Profile object (which must set up as either a client or two-way SSL). An example usage here would be to look for URIs that indicate sensitive information might be sent (such as \*/checkout) and trigger SSL for a secure (encrypted) transport.
* Add a note here**Basic-Auth Policy—** This tab is used to inject basic authentication (userid and password) into requests. It is configured by providing a URL matching regular expression and a userid and password. [Chapter 9](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1276#1276), [“Multi-Protocol Gateway,”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1276#1276) shows an example of using this to avoid putting the userid and password on an FTP URL.
* Add a note here**Soap-Action Policy—** This tab can be used to inject SOAPAction headers into Web service requests. It is configured by providing a URL matching regular expression and a value for the SOAPAction header. (For example, in some cases this must match the Web services operation name for Web services providers that require it, such as those written for .NET.)
* Add a note here**Pubkey-Auth Policy—** This tab is used to provide a private key that should be used to validate a certificate being presented in a request for authentication. This occurs with certain protocols such as SCP.
* Add a note here**Allow-Compression Policy—** This tab is used to either allow or disallow compression for certain connections. It is configured by providing a URL matching regular expression and selecting the On or Off radio button for allowing compression. The setting is valid only for protocols that use compression, such as HTTP 1.1.
* Add a note here**Restrict to HTTP 1.0 Policy—** As its name implies, this tab is used to restrict certain connections to HTTP 1.0, based on the URL. It has a URL matching field and On/Off radio button to restrict or not restrict.
* Add a note here**Inject Header Policy—** This tab is used to inject or override HTTP headers based on URL matching. In addition to the URL matching field, it has fields for the header name and value.
* Add a note here**Chunked Uploads Policy—** This tab is similar to the Allow-Compression Policy but is used for HTTP 1.1 chunked uploading.
* Add a note here**FTP Client Policies—** This tab is used to match FTP URLs with client policies to control options for the outgoing FTP connections. The configurable options and their defaults are shown in Figure 7-17. It is important to note that these can be overridden by URL query parameters, if they are present.

[](javascript:PopImage('IMG_107','http://images.books24x7.com/bookimages/id_30903/07fig17_alt.jpg','670','413'))  
Add a note hereFigure 7-17: User Agent FTP Client Policy configuration.

Add a note here[[1](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=557360520&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ch07fn01)]XSLT 2.0 and XPath 2.0 support in DataPower is limited to the decimal type. See the WebGUI guide for details.

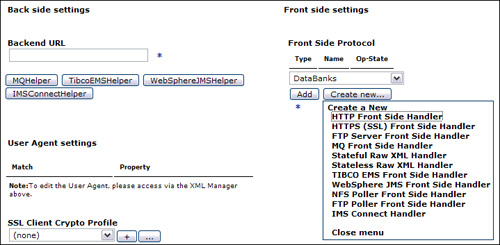
**Protocol Handlers**

Add a note hereAs you have seen, the [User Agent](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=974#974) plays a big part in the DataPower appliance’s communication with the outside world. However, this part is not as big as the group of objects we discuss in this section. For the services that use them, the protocol handlers are the conduit from the outside world to an appliance. DataPower’s mediation capabilities in many areas have been mentioned thus far, and this great array of different protocol handlers will truly bring out the protocol mediation features.

Add a note hereSome services make more extensive use of multiple protocols than others. For example, in [Chapter 6](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=755#755) we introduced you to the XML Firewall service, which can have only one front-side protocol and one back side protocol, the choice of which is limited to HTTP or HTTPS for either. Other more complex services such as the Multi-Protocol Gateway and Web Service Proxy allow for multiple protocol handlers. Figure 7-18 shows the default frontend and backend configuration for an XML firewall with a Static Backend, and you can see there is little choice available here—it is either HTTP or HTTPS (by virtue of specifying an SSL Profile).

[](javascript:PopImage('IMG_108','http://images.books24x7.com/bookimages/id_30903/07fig18_alt.jpg','726','196'))  
Add a note hereFigure 7-18: XML firewall frontend and backend protocol specification.

Add a note hereIn comparison, Figure 7-19 shows the front and back protocol configuration for a Multi-Protocol Gateway—a huge difference in terms of the options available and the ability to add multiple protocol listeners on different ports for a single service instance.

[](javascript:PopImage('IMG_109','http://images.books24x7.com/bookimages/id_30903/07fig19_alt.jpg','719','353'))  
Add a note hereFigure 7-19: Multi-Protocol Gateway frontend and backend protocol specification.

Add a note hereLet’s briefly examine the list of available protocols and their typical usages:

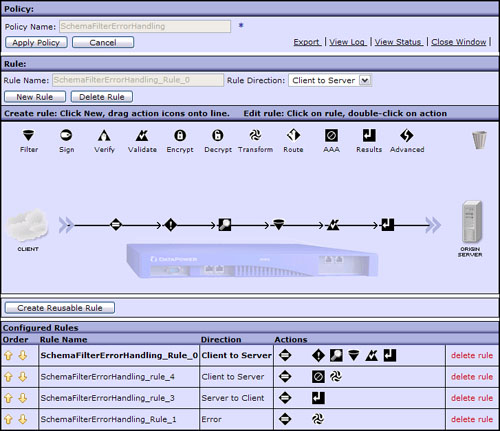
* Add a note here**HTTP—** This one is ubiquitous and by far the most commonly used protocol for application traffic to flow over.
* Add a note here**HTTPS—** Same as the previous but uses SSL for an encrypted transport to prevent the interception or viewing of message traffic by network listeners.
* Add a note here**FTP—** The appliances use FTP in several different modes; it can act as an FTP client to poll FTP servers; it can act as an FTP server itself; or it can act as a secure proxy to a true virtualized backend FTP server. However, this feature is not intended to use the appliance as a true FTP server for large files or heavy traffic. It has been implemented as simply another method in which to retrieve messages, perhaps in a scenario where a batch system places them on an FTP server to be consumed by a service running on an appliance. Several FTP scenarios are covered in [Chapter 9](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1276#1276).
* Add a note here**MQ (IBM WebSphere MQ)—** The XI50 appliance provides for tight integration for the exchange of messages with MQ Queue Managers using the MQ protocol. This is covered in [Chapter 9](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1276#1276).
* Add a note here**Stateful/Stateless RAW XML—** This is a TCP socket level handler used to exchange XML documents. (The only way to know a message has been “sent” is by the closing element for the root of the XML document.)
* Add a note here**TIBCO EMS—** The XI50 has the capability to exchange messages with TIBCO EMS, if that option is purchased with the device.
* Add a note here**WebSphere JMS—** The XI50 has the capability to exchange messages with the IBM WebSphere Application Server default messaging provider (Service Integration Bus) over the JMS JFAP protocol. This is covered in [Chapter 9](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1276#1276).
* Add a note here**NFS—** The appliances can poll NFS static or dynamic mounts for messages and consume them. This is covered in [Chapter 9](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1276#1276).
* Add a note here**IMS Connect—** The XI50 can directly connect to backend IMS instances, avoiding the need for MQ as an intermediary in some enterprise environments.

**The DataPower Processing Policy**

Add a note hereIn [Chapter 6](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=755#755), we gave a brief introduction to the Processing Policy Editor, or “multistep,” which contains rules for requests, responses, errors, and actions to carry out specific tasks such as encrypting message data. We have stressed several times that this area is truly the heart or control center of what the appliances are all about, so let’s take a closer look.

**Add a note here****Policy Editor**

Add a note hereThe Processing Policy is analogous to what might be called a work flow editor or process flow in other products. Figure 7-20 shows an entire Processing Policy that has already been configured to process messages. We’ll break down this figure into pieces to describe the various moving parts. However, first let’s discuss the major sections (Policy, Rule, and Configured Rules) at a high level, starting at the top.

[](javascript:PopImage('IMG_110','http://images.books24x7.com/bookimages/id_30903/07fig20_alt.jpg','727','627'))  
Add a note hereFigure 7-20: Processing Policy editor.

Add a note hereThe WebGUI wizards will create a default policy for you. When you are creating a new policy, the Policy Name field at the top will be blank. Think carefully about the name you provide here, because after you enter a value and move the focus away from the field, you can’t change it! The Apply Policy button is used like any other Apply button in the DataPower WebGUI, in that it commits changes to the runtime configuration and activates them. However, clicking the Apply button does not close the window here, as it does in many other places. You have to click the Close Window hyperlink in the upper-right corner for that. The other handy links at the top allow you to export the policy, view log messages for this policy, and view the status of the policy and its dependent objects.

**Add a note here****Processing Policy Rules**

Add a note hereThe Rule section (shown in Figure 7-20) shows buttons to create and delete a Rule and designate its direction. So what’s a rule? You might think of a rule as a single execution path through the device. This might be a single request message coming through the appliance from a client and on its way to some backend server, or that server’s response message flowing through the appliance on its way back to the client. We might want to perform certain actions on these request and response messages; for example, we might decrypt an encrypted request message to save the backend from that heavy lifting and then encrypt it again before it goes back to the client as a response. So a rule represents that request or response path and what *happens* along that path and is represented by a horizontal line in the policy editor.

Add a note hereThe Rule Direction drop-down that we mentioned briefly is what designates a rule as a request (Client to Server) or response (Server to Client). There is also a value in the drop-down for Both Directions to designate rules that will fire for both request and response paths, and one with the value Error to create error handling rules, which we will discuss later.

Add a note hereMost applications have a number of different paths of execution. For Web services, this is inherently defined by the operations in the WSDL. For other types of applications, these are typically defined by separate URIs that the application implements. For example, a Web banking site might have URI paths defined such as /banking/browse\_services and banking/view\_accounts. The former appears to be intended for potential customers who might want to look at rates of return on financial instruments that the bank offers. The bank would want to be as unobtrusive as possible in this case—in a brick-and-mortar store you wouldn’t run up and ask a customer browsing through brochures for their ID! The latter URI however, appears to be for customers of that bank to view their account balances, and in this case we would want to authenticate the user, perhaps asking for their account number, userid, and password. The important thing for our discussion is the fact that we often need to have multiple request (two for our example just discussed) and response rules for these various application execution paths. In fact, Figure 7-20 shows two request (Client to Server) rules in the displayed policy. The Configured Rules section at the bottom of Figure 7-20 lists all rules configured for this policy, and by clicking on any of them, the rule with the new focus is brought into the center area of the page for editing. In this figure, you can see that the first rule in the Configured Rules list is selected by virtue of it being bold, so this is also the rule that is displayed in the center editing area.

Add a note hereSo, if we have more than one request rule; for example, how do we tell the Processing Policy which to pick for a particular incoming message at runtime? We already know what the determining criterion is for the previous example—it would be the URI. The mechanism by which we match up request and response rules with messages is called the Match Rule and will be discussed shortly.

**Add a note here****Processing Actions**

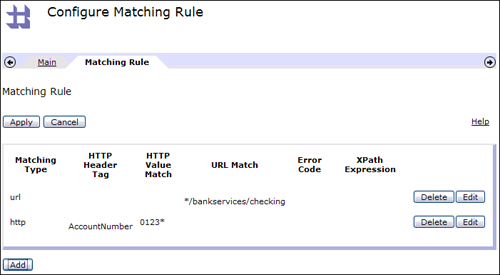
Add a note hereThe remaining mystery from Figure 7-20 is the icons along the top of the rule editing area. These are referred to as processing actions (or actions as a shortened form). They are dragged down from this horizontal palette and placed within rules to perform certain tasks. For example, in our figure you can see that the last three actions to be performed by this request rule are Filter, Schema Validate, and Send Results. Of course, it’s not just drag and drop; you must also configure these actions to do the tasks for which they are intended. For example, if you drag an Encrypt action onto the rule to encrypt messages, you would have to finish the configuration by telling the Encrypt action what certificate to use for the encryption. Shortly, we will show some of these actions and the process of configuring them well, “in action.”

Add a note hereFollowing is a quick list of the available actions and their general purpose:

* Add a note here**Filter—** This is used to filter messages based on some criteria. There are on-board filters, for example, that look for certain types of threat signatures, such as SQL injection, and reject messages that appear to contain them. You could also create a custom filter by providing your own XSLT stylesheet. This allows you to set your own filter criteria, such as to check for required fields on certain elements in the incoming XML message.
* Add a note here**Sign—** This action is used to insert digital signatures for entire messages or at the field level to facilitate message integrity and nonrepudiation.
* Add a note here**Verify—** This action is used to verify digital signatures that might already be in the incoming message.
* Add a note here**Validate—** This action is used to validate XML message structure against a schema document (XSD).
* Add a note here**Encrypt—** This action is used to encrypt message data at the message or field level.
* Add a note here**Decrypt—** This action is used to decrypt messages that have been previously encrypted.
* Add a note here**Transform—** This is probably the most commonly used action, and it is used to transform message formats using XSLT. Sometimes this action is used to execute XSLT stylesheets that do not do transformation but accomplish some utility-type tasks. It is often used as a container to execute stylesheets written as DataPower custom development, which we will discuss later in the book.
* Add a note here**Route—** This action is used to set the backend destination for dynamic routing scenarios.
* Add a note here**AAA—** This action is used to accomplish authentication, authorization, and auditing for security purposes.
* Add a note here**Results—** This action sends messages to a given destination and either waits for a result or continues without waiting. As discussed, the destination can be the backend server(s) or client, depending on the rule type, or might be a third entity, or auxiliary process.
* Add a note here**Advanced—** This action is really an abstraction containing a group of advanced actions. When you drag this to the processing rule and double-click to configure it, you are presented with a list of radio buttons for each action within, and must choose one. The choices within the Advanced action follow.
* Add a note here**Anti-virus—** This action is used to send attachments to a virus scanning server via the ICAP protocol.
* Add a note here**Call Processing Rule—** This action can call and execute another configured rule in the Processing Policy. This allows you to create an intermediary rule containing more than one action that many other rules in the policy can call, thus eliminating the need to re-create these actions over and over again.
* Add a note here**Conditional—** This action employs if/then/else processing to select an action, which might be a Call Processing action, to fire within a single action, which makes it quite powerful. The Conditional action allows you to design branching logic in your policy.
* Add a note here**Convert Query Params to XML—** This action is quite useful in processing HTTP form POSTS and GET query parameters as it will take the HTTP input and convert it to an XML nodeset, which makes the request much easier to process in DataPower!
* Add a note here**Crypto Binary—** This action is used for crypto operations on non-XML data. For example non-XML data can be encrypted, decrypted, signed, or signature-validated using Public Key Cryptography Standard (PKCS) #7.
* Add a note here**Event-sink—** This action causes the system to wait for all listed asynchronous actions to complete before proceeding to the next action in the Processing Policy. This is useful to pull together the results of one or more actions that have been processing asynchronously.
* Add a note here**Extract Using XPath—** This action uses an XPath expression to pull data out of another context (contexts are described later in this chapter) and store it either in another context or a variable.
* Add a note here**Fetch—** This action can retrieve documents from either the DataPower file system or remote resources, for example a properties file.
* Add a note here**For-each—** This powerful action executes a looping construct, as named, using counters or expressions.
* Add a note here**Header Rewrite—** This action has the capability to do rewriting similar to that available on the service configuration page, only at a more granular level—from inside a policy rule.
* Add a note here**Log—** This action is used to send message content to a remote logging facility. Care should be taken to not expose private information in the message. The log message can be encrypted and/or signed. Note that as this is used from within the Processing Policy, it is typically used for transaction/message logging, rather than event logging. Due to the larger size of these log messages, they are normally sent directly to targets off-device using protocols such as syslog.
* Add a note here**MQ Header—** This action is used to manipulate MQ MQMD headers for GETs and PUTs.
* Add a note here**On Error—** This action is used to handle errors that occur within the processing rule. It can abort the transaction and call another rule (typically an error handling rule). It is discussed at the end of this chapter.
* Add a note here**Results Asynchronous—** This rule can be used to send async or fire and forget results out from the processing rule, for example to write audit log records to a database.
* Add a note here**Route (Using Variable)—** This action can be used to set the destination URL for a message dynamically.
* Add a note here**Set Variable—** This action is used to simply set a DataPower variable to some value.
* Add a note here**SQL—** This action is used to execute SQL commands against a database, if the appliance is licensed to do so. The result can be captured and used for further processing. A typical use is to enrich message content with additional data, or to customize the AAA Policy to read a nonstandard user registry housed in a database.
* Add a note here**Strip Attachments—** This action can be used to remove attachments from a message.
* Add a note here**Transform (using Processing Instruction)—** This action transforms the XML based on some processing instructions that are contained within the input document itself.
* Add a note here**Transform Binary—** This action is used to do non-XML transforms on an XI50 such as EDI to XML or vice versa.

**Add a note here****Matching Rules**

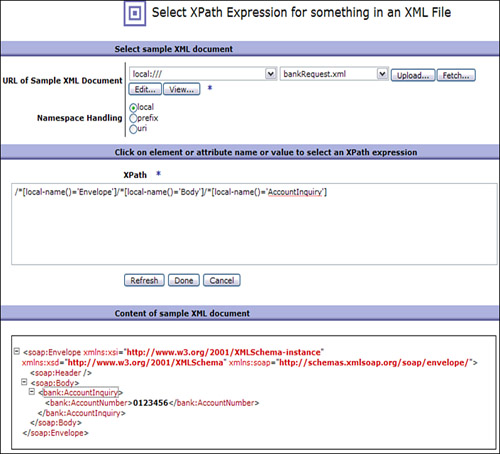
Add a note hereA Matching Rule is just what it sounds like. It matches request and response messages arriving at the device, and error conditions, with the Processing Policy rules that are meant to handle them. In this next section, we walk through a complete scenario. Before we get to that, let’s look at a few sample Matching Rules. Figure 7-21 shows an example that will match on *two* conditions—the incoming message must have a URL that ends with /bankingservices/checking *and* must contain a HTTP header named AccountNumber that must contain a value that begins with 0123. There is a radio button on the Match configuration page titled Boolean Or Combinations that can be used to specify “all must match” or “any can match” behavior for multiple matching conditions listed here.

[](javascript:PopImage('IMG_111','http://images.books24x7.com/bookimages/id_30903/07fig21_alt.jpg','701','385'))  
Add a note hereFigure 7-21: Matching Rule for a URL and HTTP header.

Add a note hereMany common network components, such as load balancers and proxies, can interrogate HTTP-level data like the URL and header values. What sets the DataPower appliances apart from these is their ability to understand deeper application layer message formats, such as XML and SOAP. Let’s look at that in our next example. Figure 7-22 shows how we begin the configuration. XPath has been selected as the Matching Type. This displays the XPath Tool, a feature that you might see in several places when configuring processing policies, such as when signing or encrypting messages at the field level. To find nodes in XML documents, the XPath language is used by this convenient utility, as shown in Figure 7-22.

[](javascript:PopImage('IMG_112','http://images.books24x7.com/bookimages/id_30903/07fig22.jpg','464','169'))  
Add a note hereFigure 7-22: Configuring an XPath Matching Rule.

Add a note hereThis lets us upload a sample input message to the device, and click on the element we are interested in, and DataPower will generate the XPath expression. Figure 7-23 shows the completed page, after we have clicked on the AccountInquiry element in the sample message bankRequest.xml, which was uploaded.

[](javascript:PopImage('IMG_113','http://images.books24x7.com/bookimages/id_30903/07fig23_alt.jpg','705','640'))  
Add a note hereFigure 7-23: Using the XPath Tool.

Add a note hereFigure 7-24 shows the completed XPath Matching Rule. Now that’s powerful!

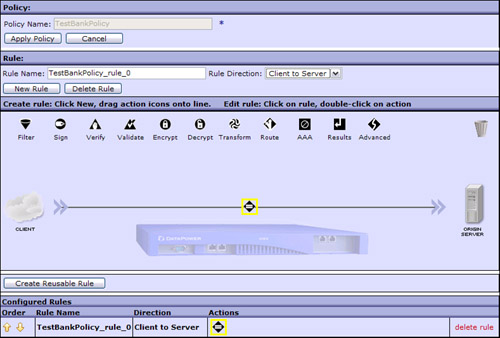
[](javascript:PopImage('IMG_114','http://images.books24x7.com/bookimages/id_30903/07fig24_alt.jpg','1003','351'))  
Add a note hereFigure 7-24: Completed XPath Matching Rule.

Add a note hereIn addition to the URL, Header, and XPath matches we have shown, Match Rules also allow for matching on error codes. We will demonstrate this in [Chapter 24](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4365#4365). The other two choices in the Matching Type drop-down, Host and Full URL, should not be used as they are legacy configurations. HTTP and URL should be used instead, respectively.

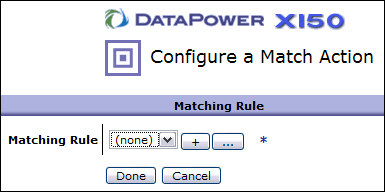
Add a note hereThe terms Matching Rule and Match Action are often used synonymously, however there is a distinction. Referring again to Figure 7-20, you can see that each configured rule has at its start an icon with an equal sign at its center. Looks like an action, doesn’t it? Well...it is. This is the Match Action, which does nothing more than house a Match Rule, as you will see.

**Add a note here****Creating an Example Processing Policy**

Add a note hereTo gain a better understanding of what we have discussed thus far, let’s create the scenario for one of our banking URIs. Figure 7-25 shows a new policy. (How you get here will be discussed in our upcoming chapters on services.) The TestBankingPolicy Policy Name has been entered, the New Rule button has been clicked, and Client to Server has been selected in the Rule Direction drop-down list. When the New Rule button is pressed, a brand new horizontal rule line is presented in the workspace, and a Match Action is presented on it bordered in yellow highlight, which indicates some configuration is necessary. The system is saying “You have created a new request rule, so now first configure this Match Action on it to tell me when this new request rule should be fired.” You can see also that the rule has appeared in the Configured Rules list at the bottom, even though we haven’t truly configured it yet.

[](javascript:PopImage('IMG_115','http://images.books24x7.com/bookimages/id_30903/07fig25_alt.jpg','789','533'))  
Add a note hereFigure 7-25: New Processing Policy Rule.

Add a note hereSo our first course of action would be to double-click the Match Rule to configure it. When this is done, a new page pops up, as shown in Figure 7-26, asking us to configure the Match Action. As you can see, the only thing to configure is a Match Rule embedded inside. This is a simple config page but it allows us to point out something about how the DataPower WebGUI works. Notice that there is no Matching Rule selected in the drop-down, and there are plus (+) and ellipsis (...) buttons next to it. This pattern is seen throughout the DataPower WebGUI for editing objects. The plus button is always used to create a new object, so in this case if we were to click it a new page would pop up letting us configure a new Match Rule. If we had an existing Match Rule selected in the drop-down list, we might want to click the ellipsis button to edit that object. So, plus means create new, and ellipsis means edit existing. Memorize this, because you will see it a lot!

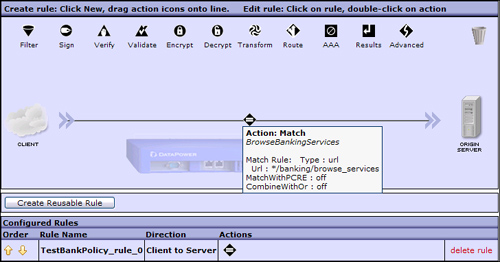
  
Add a note hereFigure 7-26: Match configuration.

Add a note hereIn this example, we are showing how to create the first banking rule scenario, so we have clicked the plus button and entered a Matching Rule name of BrowseBankingServices on the main tab and then moved to the Matching Rule tab and added a new Matching Rule, as shown in Figure 7-27.

  
Add a note hereFigure 7-27: BrowseBankingServices URL Match.

Add a note hereNotice the format of the URL Match expression; it doesn’t look like PCRE, does it? Matching Rules use simple Unix-style matching syntax by default; thus, the star (\*) character represents 0 (zero) or more characters of any value, and a question mark (?) represents any one character. The first tab of the Matching Rule configuration page lets you select the match syntax. If you wanted to use PCRE, you could select that syntax instead (in which case, the above URL Match would be expressed as “.\*/banking/browse\_services”).

Add a note hereAfter pressing the Save, Apply, and Done buttons to get back to the Policy Editor, our Match Action is no longer highlighted, as it has been configured, and hovering over it with the cursor shows a convenient pop-up bubble with the configuration (see Figure 7-28). Notice also how the GUI is showing the request flow graphically from the client cloud to the server.

[](javascript:PopImage('IMG_118','http://images.books24x7.com/bookimages/id_30903/07fig28_alt.jpg','728','382'))  
Add a note hereFigure 7-28: Request Rule with Banking match.

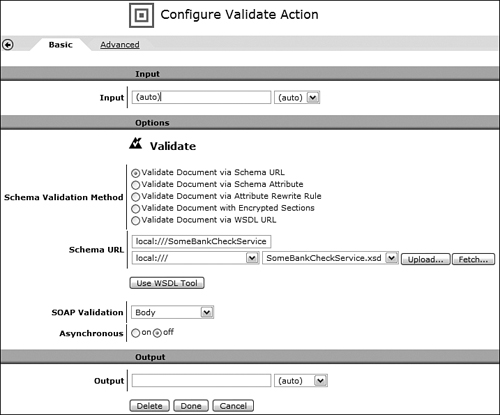
**Add a note here****Processing Rule Priority**

Add a note hereRefer for a moment back to Figure 7-20. Notice that we had two request rules there. Now that we know how to configure Matching Rules, and we know that the appliance uses those to pick the correct rule, how does that happen, exactly? It is important to understand this process, because a failure to do so, or to not pay attention to it, can cause some difficult to debug problems. The process is that the runtime starts at the top of the configured rules and checks each rule for a match until it finds one. This happens for request, response, and error rules. So consider what would happen if the top-most rule had a match on a URL of \* and the ones beneath it were more granular, for example /banking/browse\_services and /banking/view\_accounts.

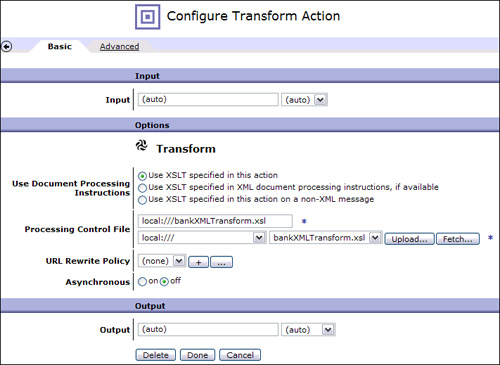
Add a note hereIf you think about this, you will realize that the request rules under the top one will never get fired, because a URL match on broad wildcard such as \* consumes each incoming message, as it matches everything. How do we fix this? Notice the column titled Order at the left of the Configurable Rules table. This is where you would use the arrows to move a more loosely defined match to the bottom of the list. Check these carefully as part of your configuration reviews and always consider this issue in your troubleshooting endeavors!

**Add a note here****Configuring a Policy Rule**

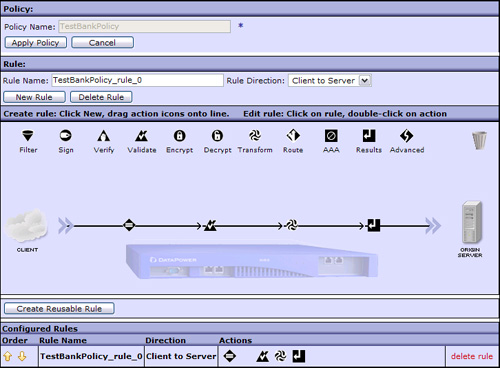
Add a note hereNow let’s get back to our example policy rule. Alas, a request rule cannot live by a simple match alone! Well, it could but then it wouldn’t accomplish anything. There must be something additional that we want to do on this request rule flow. Let’s assume that we want to schema validate the incoming request XML message, transform the incoming XML to a schema more suitable to our backend application, and then send those results to the backend. Of course, we will need actions to accomplish these tasks. You can probably already guess which ones. To begin, we have dragged down the Validate action to the request rule and placed it to the right of the Match. To configure, we double-click and select the radio button to validate with an external schema (XSD) document, which we then upload from our workstation file system, as shown in Figure 7-29. This schema definition file now resides in the local: directory of our domain’s file system.

[](javascript:PopImage('IMG_119','http://images.books24x7.com/bookimages/id_30903/07fig29_alt.jpg','717','595'))  
Add a note hereFigure 7-29: Configuring the Validate Action.

Add a note hereAfter closing the Validate action, we drag a Transform action to the right of it on the processing rule and double-click again to edit. Similar to the Validate, we rely on an external file here—the actual XSLT processing file that contains the transform instructions. Figure 7-30 shows the configuration of this action.

[](javascript:PopImage('IMG_120','http://images.books24x7.com/bookimages/id_30903/07fig30_alt.jpg','711','519'))  
Add a note hereFigure 7-30: Configuring the Transform Action.

Add a note hereFinally, for completeness, we drag a Results action to the end of the request rule. You might have noticed that one of our request rules in Figure 7-20 had a Results action at the end, and the other request rule did not (it ended in a Transform). This might look quite peculiar and incomplete. The truth of the matter is that certain actions are capable of sending results on their own if they end a processing rule, whereas others cannot and require a Results action to follow. Rather than try to remember which are which, some users might simply insist on always closing a rule with a Results action (in fact, DataPower will add this for you automatically), whereas others might consider this wasteful. Neither approach is wrong. Our completed policy is shown in Figure 7-31.

[](javascript:PopImage('IMG_121','http://images.books24x7.com/bookimages/id_30903/07fig31_alt.jpg','726','534'))  
Add a note hereFigure 7-31: Completed Banking Processing Policy.

**Add a note here****Contexts**

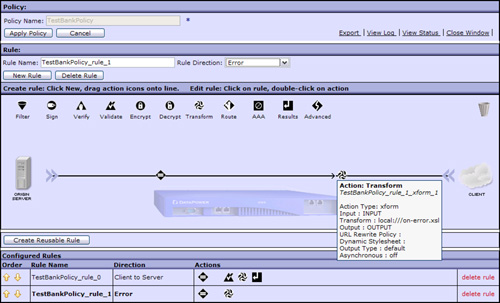
Add a note hereThere is more to understand about DataPower Processing Policy rules and actions. An important concept is the context. These are named memory segments in which transient data can be stored as the message moves across our processing rule flow. Two contexts that are created automatically for you are called INPUT and OUTPUT. For a request message, the INPUT context can be referenced by any action that wants a clean copy of the message as it arrived at the Processing Policy. (It might have been altered subsequently in other contexts by transformation and so on.) To get this, the action would have to specify the INPUT context as its input. Refer to Figure 7-30, where we configured our Transform Action. Do you see the input context designation drop-down at the top and the output context designation at the bottom? Right now they are set to Auto, which means the appliance will do its best job to figure this out for you. Our next example shows how well it did. Because this transform was acting on the input message, we could have just selected INPUT from that drop-down and gotten the same results. If our transform were going to pass the newly modified XML to another transform, it would use a temporary variable name in the output context (or PIPE, as discussed next), and this would then be used as the input context for the next Transform action. The OUTPUT context is simply the data to be sent to the backend server in a request rule. For response rules, the INPUT context is what comes from the backend server as a response, and the OUTPUT context is what we would end up responding to the client with.

Add a note hereThere are two other reserved context names—PIPE and NULL. PIPE is used in situations such as we just described for Transforms where the output of one action is the input of an adjacent action. PIPE allows this data to be streamed through as it is processed, which is good for performance! There are some restrictions on its use, such as if you are setting variables in the stylesheet housed by this transform. NULL is used as an efficiency construct for actions that do not modify the message content at all.

Add a note hereLet’s see how the contexts were set up for us in our sample policy. If you re-create our example and hover the mouse over each of the actions, you will see how the contexts flow together. Note that the user interface has made a “best guess” about what input and output contexts to use for each action in the processing rule. Alas, the best guess is sometimes not the intended behavior. Always double-check your context chain through an entire rule to make sure the desired data gets to each individual action and makes it to the OUTPUT context.

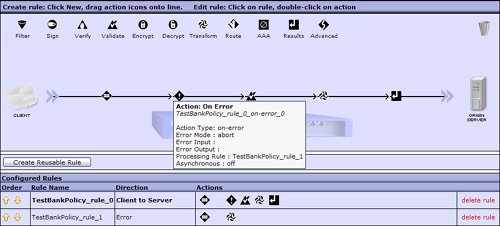
**Add a note here****Error Handling**

Add a note hereWhat about when things go wrong? Bad things sometimes happen to good policies. We must be prepared. Let’s use our example to facilitate a quick introduction to DataPower Processing Policy error handling. One fast way to handle errors is to simply create an error rule. Figure 7-32 shows that we have done just that. And it was simple. We just clicked the New Rule button, configured the resulting Match on a URL of \* (although we could have matched on specific types of errors), and dragged down a Transform action and included a stylesheet of on-error.xsl, which formats an HTML response to the client. (We assume it is a browser.)

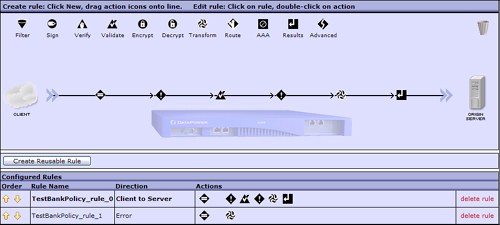
[](javascript:PopImage('IMG_122','http://images.books24x7.com/bookimages/id_30903/07fig32_alt.jpg','932','565'))  
Add a note hereFigure 7-32: Processing policy with error rule.

Add a note hereYou are probably searching for some way to determine when this error rule will be invoked. The answer is that it’s automatic. When an error condition occurs in a Processing Policy, the appliance searches for any error rules that have been configured by moving down the list from top to bottom, as we discussed earlier with the request and response rules. If it finds one that matches, it will move control there and fire it. This is the fundamental usage of error rules.

Add a note hereHowever, there is another, more fine-grained approach available. You might remember that earlier in the chapter, we listed the available processing actions, and one of them (under Advanced) was the On Error action. This action can be placed in a Processing Policy to gain control when errors occur. Figure 7-33 shows that we have pulled down an Advanced action just before our schema validation, selected the On Error action from the radio button list, and then configured the action to abort the transaction and call our error rule, should any errors occur. Note that in this case, due to the presence of the error action, DataPower does not comb down through the list of error rules looking for a match. It gives control to the error action in scope, which decides what to do. (The transaction could be allowed to continue, rather than abort.)

[](javascript:PopImage('IMG_123','http://images.books24x7.com/bookimages/id_30903/07fig33_alt.jpg','912','412'))  
Add a note hereFigure 7-33: On Error action.

Add a note hereAs we said previously, when the policy runs into the On Error action during the rule processing from left to right, it puts that action in scope for any errors that might occur from that point forward. However, what if you need to handle transformation errors differently from schema validation errors? As Figure 7-34 shows, you can simply add in a new On Error action. Now, when the second error action is encountered, before the transform, it will fire upon any errors that happen past that point.

[](javascript:PopImage('IMG_124','http://images.books24x7.com/bookimages/id_30903/07fig34_alt.jpg','914','412'))  
Add a note hereFigure 7-34: Request rule with two On Error actions.

Add a note hereIn the preceding example, we took you through the usual business of building a Processing Policy. We intentionally kept this independent of the services that will house the policy, because we don’t cover the services until the remaining chapters in this part of the book. We showed a few of the most commonly used actions and listed the rest with a brief overview of each. Many of these will be shown in action in the chapters to come.

## Summary

Add a note hereThis chapter covered a lot of information that introduced you to many of the common objects used in DataPower services—from the different types of backend configurations utilized (static, dynamic, and loopback) to the fundamental utility objects (XML Manager and User Agent), through the more complex topics such as how to configure processing policies and the rules and actions within them. You have now graduated from the prerequisite section of background information and are ready to start diving into actual DataPower services! [Chapter 8](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1082#1082) will get you started.