|  |
| --- |
| **Chapter 10 - Web Service Proxy**  IBM WebSphere DataPower SOA Appliance Handbook  by  [Bill Hines et al.](http://www.books24x7.com/SearchResults.aspx?qdom=author&scol=%7ball%7d&qstr=Bill%20Hines%20et%20al.)  [IBM Press](http://www.books24x7.com/books.aspx?imprintid=392) © 2008 [*Citation*](javascript:ShowCitation(30903,0,null,1))  Recommend? Click to vote yesClick to vote noYes, I recommend this titleClick to changeNo, I do not recommend this titleClick to change |

|  |  |  |
| --- | --- | --- |
|  | | |
|  | |  |  |  | | --- | --- | --- | | http://www.books24x7.com/images/progress_left.gif | Progress:Progress: 25% | http://www.books24x7.com/images/progress_right.gif | |  |



**Location is bookmarkedAdd a note here****Web Services Description Language (WSDL)**

Add a note hereAfter a Web service is created and deployed, it is available to all the consumers that have access. The challenge is, “How do those consumers know how to call the service?” There are several details that you must know about the service interface in order to call it. As mentioned previously, Web services are self describing. This self description is provided in a file called a WSDL document.

Add a note hereA WSDL document is an XML document in a specific format that tells the consumer everything he needs to know to call the service. This format, as well as how the information is conveyed within the document, is dictated by an approved W3C specification. You can think of it as the instruction manual for the Web service. There are four primary things that the consumer must know to make this call: the operations available, the message formats, the protocol, and the location of the service.

Add a note hereBefore calling a Web service, you must first know the operations that are available within the service. An operation is somewhat analogous to a method or function in other programming languages. Just as other programs can have many different methods or functions, Web services can expose many different operations that can be called to provide different functionality within the service. For example, a banking Web service may have a balanceQuery operation, a withdrawFunds operation, and a depositFunds operation.

Add a note hereAfter you determine which operation you need to call, you must then know how to format the request message; knowing the response format also helps you prepare to process the returning data. As mentioned before, the request and response messages are SOAP documents. In addition, error messages can also be returned in a SOAP document called a SOAP fault. Because a SOAP document is built on XML, the request, response, and fault formats can be described through an XML schema. These schemas are included (or referenced) in the WSDL document.

Add a note hereNow that you know what operation you want to call and how to build the request message, there are two small details remaining. Because Web services support many different protocols, you must know the protocol to use for the particular operation you are calling. This is known as the *binding information*. The last piece of information you need to know is where to send the request. This is known as the *endpoint*. The endpoint is the complete URL that the request will be sent to. When calling a service over HTTP, this will look like a standard Web URL consisting of the hostname, port, and URI. For example, let’s suppose there is a banking Web service that can be accessed via HTTPS at the hostname mybank.com over port 4443 with a URI of bankservice. The endpoint for calling this service would be [http://mybank.com:4443/bankservice.](http://mybank.com:4443/bankservice)

Add a note hereYou now know what pieces of information you need and where to find them, but how is this information conveyed in the WSDL document? To be honest, it’s not pretty. That is, reading a WSDL document is not like reading the comics section in the Sunday newspaper. When viewed in their raw format (text), they can be difficult to understand. It is sometimes difficult to follow the interaction between the various sections, and to figure out *exactly* what message needs to be sent for a given operation. Fortunately, there are many tools available to help in writing and reading these files. Often a WSDL is comprised of many separate documents that are referenced from within the main document. For example, the schemas for the request and response messages are usually contained in separate documents and referenced within the WSDL. This makes maintenance and reusability for the schemas easier.

Add a note hereBeing the technologists that we are, we just can’t resist taking a look under the hood and seeing what a WSDL looks like. After all, it is just XML and how bad can that be?

Add a note hereAs we mentioned earlier in this section, there are four main pieces of information that you will need to call a Web service. Because a WSDL provides these pieces of information, the document is broken up into multiple sections.

Add a note hereAfter the root element of the WSDL document <wsdl:definitions>, in the [first section](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1556#1556), you will see <wsdl:types>. This contains all the schemas for the request, response, and SOAP fault messages for all the operations exposed by the service. These schemas can be coded directly in the WSDL itself or the schema files can be referenced. Listing 10-1 shows the schema for a request message.

Add a note hereListing 10-1: Request Message Schema Within a WSDL

Add a note here<wsdl:types>

<xsd:schema targetNamespace="http://www.example.org/BookService/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="BookQuery">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="book">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="name" type="xsd:string" />

<xsd:element name="author" type="xsd:string"/>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

Add a note hereBecause there can be many schemas and elements within the schemas, the WSDL must identify which portion of the schema(s) actually form a request or response message. This is identified in the [next section](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1587#1587) of the WSDL in the <wsdl:message> element. Listing 10-2 shows an example of a message element that has an attribute “name” with a value of “BookQueryRequest.” The <part> element in this message identifies the portion of a schema that makes up this message. In this example it refers to the <BookQuery> element, which is in the schema shown in Listing 10-1. We now have a message called BookQueryRequest, and we know that it should conform to the BookQuery element definition in the schema.

Add a note hereListing 10-2: Linking the BookQueryRequest Message to the Schema

Add a note here<wsdl:message name="BookQueryRequest">

<wsdl:part element="tns:BookQuery" name="parameters" />

</wsdl:message>

Add a note hereThe next part of the WSDL describes the operations that are available within the service. This description will provide the operation name, the request message used to call it, and the response message that can be expected. All of the operations listed are enclosed within a <wsdl:portType> element and can be identified by the <wsdl:operation> element within. Just as a port in the networking world describes a connection point to a physical server or device, the portType in the WSDL describes the connection point to the Web service. Listing 10-3 shows an operation named BookQuery. The input message for this operation is the BookQueryRequest message shown in Listing 10-2, which is defined by the schema shown in Listing 10-1.

Add a note hereListing 10-3: Describing the Operations

Add a note here<wsdl:portType name="BookService">

<wsdl:operation name="BookQuery">

<wsdl:input message="tns:BookQueryRequest" name="test"/>

<wsdl:output message="tns:BookQueryResponse" />

</wsdl:operation>

</wsdl:portType>

Add a note hereWe now know all the operations available in the Web service and how they should be formatted. The [next section](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1587#1587) of the WSDL describes the communication protocol that must be used. It is valid for different operations within the same service to be called using different communication protocols. This information is contained within the <wsdl:binding> element. Listing 10-4 shows an example of a binding element for the example WSDL that we have been building. Notice the type attribute of the binding element refers to the portType shown in Listing 10-3. This indicates that all of the binding information specified in this element applies to this portType. Notice the soap:binding element has an attribute named transport. This indicates the transport protocol to be used. In this example it is SOAP over the HTTP protocol.

Add a note hereListing 10-4: WSDL Binding

Add a note here<wsdl:binding name="BookServiceSOAP" type="tns:BookService">

<soap:binding style="document"

transport="http://schemas.xmlsoap.org/soap/http" />

<wsdl:operation name="BookQuery">

<soap:operation

soapAction="http://www.example.org/BookService/BookQuery" />

<wsdl:input name="bookQueryRequest">

<soap:body use="literal" />

</wsdl:input>

<wsdl:output>

<soap:body use="literal" />

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

Add a note hereYou now should have almost every piece of information required to call this Web service. We know what operations are available, the format of the request and response messages, and the transport protocol to use. The one small piece missing from this puzzle is where to send the request. The address, or endpoint of the service is provided in the <wsdl:service> element at the end of the WSDL document. In this element you can specify an address, or endpoint, for each binding specified in the binding section. This information is contained within the <wsdl:port> element, which is a child of the <wsdl:service> element. Listing 10-5 shows an endpoint example. Notice that the port element refers to the BookServiceSOAP binding defined in Listing 10-4. The <soap:address> element provides the actual endpoint where you would send your request.

Add a note hereListing 10-5: Defining the Endpoint

Add a note here<wsdl:service name="BookService">

<wsdl:port binding="tns:BookServiceSOAP" name="BookServiceSOAP">

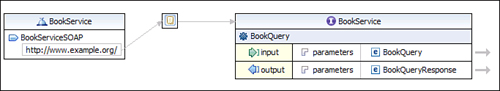
<soap:address

location="http://www.abcbooks.com/books/query/queryService"/>

</wsdl:port>

</wsdl:service>

Add a note hereSo, there you have it! One “simple” document has provided every piece of information required to call a Web service. We did warn you that it isn’t a user-friendly document. This is why there are many tools available for creating and viewing WSDL documents. Figure 10-1 shows the same example WSDL that we have been looking at as it is displayed within the Rational Application Developer (RAD) product. This is much easier on the eyes than the raw text!

[](javascript:PopImage('IMG_215','http://images.books24x7.com/bookimages/id_30903/10fig01_alt.jpg','683','124'))  
Add a note hereFigure 10-1: Viewing a WSDL in RAD.

Add a note hereAlthough this was a brief introduction to the WSDL document, it provided enough detail for you to understand how the DataPower WSP service uses this document to determine the full interface definition within the configuration.

## Creating a Web Service Proxy (WSP)

Add a note hereWe’ve already seen how DataPower provides the capability to perform many different types of processing on request and response messages such as Authentication, Authorization, and Auditing (AAA), cryptographic functions, message transformations, and dynamic routing. As you might expect, all this is available in a WSP but there is an added benefit provided in this service type. Because a Web Service Proxy uses a WSDL for configuration, it is able to parse the document, determine the hierarchy (WSDL, service, port, binding, operation) inherent in the service, and define these processing rules at any level of that hierarchy. For example, processing rules can be configured for individual operations or portTypes, at the WSDL level, or at the service level. This service also provides all the threat protection that DataPower has to offer. In [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) you’ll see how Web services, although convenient, are vulnerable to attack.

Add a note hereIt is often surprising how easy it is to configure a WSP that provides threat protection, schema validation, and service-level monitoring. All you need to provide to build a service that provides this value is three simple things: a WSDL, a name for your service, and a Front Side Handler (FSH). That is it! Although it might appear as some magic trick, it should come as no surprise how this is accomplished. As we mentioned earlier in this chapter, a Web service is described by a WSDL document that provides all the information required to call the service. So if this is a magic trick, the WSDL is the smoke and mirrors. Now let’s take a look at how this is all done.

Add a note hereMuch like the other services we have discussed thus far, a WSP can be created by navigating from the Control Panel in the WebGUI to Web Service Proxy and selecting Add. The wizard immediately asks for the first of the three pieces of information required—a name. After a name is provided, clicking the Create Web Service Proxy button prompts for the second piece of information—the WSDL. This can be uploaded to the device’s file system or fetched just as you would any other file. Figure 10-2 shows this configuration screen where the BookService.wsdl file has been selected from the DataPower file system in the local directory. Notice the options to upload and fetch the WSDL from a remote location.

[](javascript:PopImage('IMG_216','http://images.books24x7.com/bookimages/id_30903/10fig02_alt.jpg','815','534'))  
Add a note hereFigure 10-2: Adding a WSDL to the Web Service Proxy.

Add a note hereYou may also notice other available options on this screen such as options to browse UDDI or to add UDDI and WSRR subscriptions. There are other possibilities for referencing a WSDL; we will defer those discussions until later in this chapter. On the bottom of this configuration screen are two options for defining and enforcing WS-Policy sets. WS-Policy is a specification that allows security policies to be defined for a Web service that must be adhered to by the consumers. The WSP provides support for this and is discussed in detail in [Chapter 19](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3524#3524), [“Web Services Security.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3595#3595) However, most simple use cases won’t require any special WS-Policy action during configuration.

Add a note hereClicking the Next button creates a WSP, but there is one critical piece of information missing. That is the FSH that will be configured to listen for requests to be processed by this service. The next screen presented is where this will be configured. This is the WSDLs tab within the service configuration. You will notice three sections here. The top section is labeled “local” and is where you will define the FSH object(s) that will be configured to handle requests. The middle section is labeled “remote” and defines the backend that the request will be sent to. This information is provided within the WSDL, so the service already populates this for you. The bottom section on this screen is labeled “published.” This simply defines the endpoint information that is published to the client if DataPower will republish this WSDL such as to an external repository. This is typically configured to “use local,” which means that the published WSDL will contain the endpoint information that you will define in the top “local” section of this screen.

Add a note hereBecause the WSDL already provided the remote endpoint information for us, the only thing required to be configured on this screen is the FSH object in the local section. Note that any of the remote information can be changed if need be. An existing FSH object can be selected from the drop-down here or a new object can be created by clicking the + button. After creating or selecting a FSH for this configuration, a URI must be associated with it. By default the URI from the endpoint within the WSDL is populated but this can be changed. This URI is required because Web services publish endpoints to the clients that consist of the complete URL, including the URI. This also can abstract the actual URI from the client and provide a shorter, simpler one. Figure 10-3 shows where we are configuring an HTTP FSH object to our service with a URI of /bookService. This is the endpoint that the client uses to call this service.

[](javascript:PopImage('IMG_217','http://images.books24x7.com/bookimages/id_30903/10fig03_alt.jpg','708','392'))  
Add a note hereFigure 10-3: Adding an FSH.

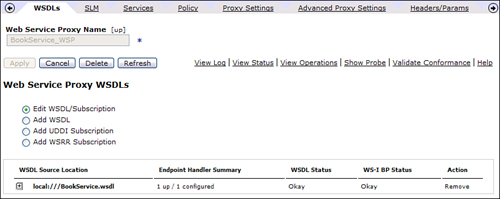
Add a note hereNotice in this example that the URI we are exposing to the client is much shorter than the actual URI that the backend Web service is expecting. Because the FSH is used in conjunction with the URI defined, it is possible to use the same FSH in more than one configuration. We discuss more about this later in this chapter. After the FSH is configured and the URI is entered, the Add button must be clicked to add this to the configuration.

### Tip: Remember to Click the Add Button!

Add a note hereAfter creating or selecting an existing FSH Object and specifying a URI to associate with it, you must click the Add button to add it to the configuration. Not clicking Add is a mistake that is easy to make as the FSH appears to have been added to the service when it is visible in the drop-down and the URI is entered in the text box.

Add a note hereFigure 10-3 also shows the Remote configuration for this service. This is the backend definition where all requests will be forwarded. This information has been automatically populated from our WSDL as it is defined in the <service> section.

Add a note hereClicking the Next button completes the configuration of the WSP and redisplays the WSDL configuration tab. The Endpoint Handler Summary indicates how many endpoints are configured for the WSDL and how many are in the “up” status. In our example, we configured only one so there should be one up and one configured, as shown in Figure 10-4. The WSDL status of “Okay” indicates that the WSDL is valid.

[](javascript:PopImage('IMG_218','http://images.books24x7.com/bookimages/id_30903/10fig04_alt.jpg','815','324'))  
Add a note hereFigure 10-4: WSDL and endpoint status.

Add a note hereIn three simple steps, we configured a WSP service. The service created is not just a proxy that knows how to listen to requests and pass them to the backend. As mentioned earlier, this proxy provides a tremendous amount of value by providing threat protection and schema validation. By virtue of the WSDL that you imported, it will allow requests only to the backend Web service that fully complies with the WSDL. Think about how much that frees up the actual service, knowing that it will receive only requests that conform to the WSDL.

Add a note hereTo demonstrate this power, let’s take a look at what happens when we send a request to the service. Let’s assume that the WSDL we imported in our example is the same WSDL we used in our examples in the [previous section](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1564#1564) of this chapter. We know from [Listing 10-3](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1571#1571) that there is an operation named BookQuery that we can call. We also know from looking at [Listing 10-1](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1567#1567) how this request should be formatted. So it is safe to say that if we send a SOAP document over HTTP for the BookQuery operation that complies with the request schema, it should be accepted by our DataPower service and passed to the backend Web service. Listing 10-6 shows such a request.

Add a note hereListing 10-6: Valid BookService SOAP Request

Add a note here<soapenv:Envelope

xmlns:q0="http://www.example.org/BookService/"

xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<soapenv:Body>

<q0:BookQuery>

<book>

<name>Moby Dick</name>

<author>Herman Melville</author>

</book>

</q0:BookQuery>

</soapenv:Body>

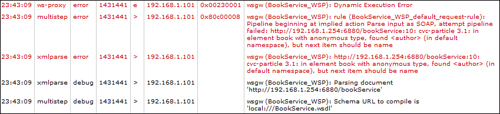
</soapenv:Envelope>

Add a note hereNow that we have a valid request document, we can send it to our new WSP. As we know, the endpoint that we should send the request to is specified in the WSDL for the service. Because we are using DataPower to proxy this Web service, we cannot use the endpoint listed in this WSDL. We must use the endpoint that is virtualized within our proxy. This is the combination of the FSH object used by the service and the URI configured with it when we added it to our service as shown in Figure 10-3. In this case, we would send our request to the IP:port that the BookService\_HTTP\_FSH object is listening on with a URI of bookService. For example, if this FSH was listening on IP address 192.168.22.5 and port 3100, we would send this SOAP request document to [http://192.168.22.5:3100/bookService.](http://192.168.22.5:3100/bookService)

Add a note hereAfter sending this valid request, the service would have performed its request schema validation and threat protection, forwarded it to the back-end Web service, and returned the response back to the client. Figure 10-5 shows the DataPower log from this valid request. Notice, reading from the bottom entry up, how DataPower recognized the IP:port and URL combination as being that of an endpoint configuration within the BookService\_WSP configuration. Next, you can see how the SOAP Body is looked at for an operation. Finally, DataPower recognizes the BookQuery operation and allows the request to proceed through the service.

[](javascript:PopImage('IMG_219','http://images.books24x7.com/bookimages/id_30903/10fig05_alt.jpg','808','174'))  
Add a note hereFigure 10-5: DataPower logs from a valid Web service request.

Add a note hereWe have seen how a valid request looks within the DataPower logs when sent to our WSP service. Now, to prove that the request document is actually schema validated, let’s send an invalid request to this service that does not comply with the schema for the BookQuery request. To do this we will simply reverse the order of the <name> and the <author> elements in the request document. After sending the request, we received a plain SOAP fault with a nondescriptive message. If we were to take a look at the DataPower log, as shown in Figure 10-6, we can see why the request was rejected. Notice that the BookService WSDL was found in the DataPower cache and then the request document was parsed. Immediately after this, the request is rejected. You can see a descriptive message indicating that the element <author> was found but the next item should have been <name>. This is proof that the request document is validated against the appropriate schema within the WSDL. All this happens without having to configure a single processing rule! This is the power of the WSP.

[](javascript:PopImage('IMG_220','http://images.books24x7.com/bookimages/id_30903/10fig06_alt.jpg','849','194'))  
Add a note hereFigure 10-6: DataPower logs from an invalid Web service request.

## The Processing Policy

Add a note hereAlthough the simple service configuration described in the [previous section](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1587#1587) produced a powerful proxy for your Web service, you will most likely want to add some other processing capabilities to your service such as AAA, message transformation, or encryption. We already know what a Processing Policy is and how it contains actions to be applied to a request, response, or on error, but this service type offers even more configuration options for the Processing Policy.

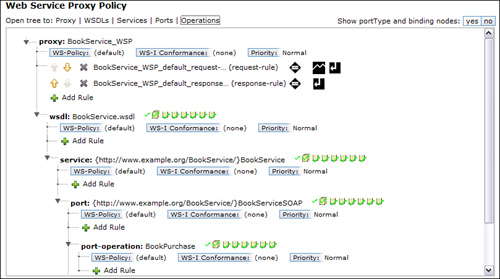
### Add a note hereProcessing Rules

Add a note hereIn the first two DataPower service types discussed in the previous chapters, you saw how a Processing Policy is used to configure processing rules to be applied to request or response messages. These rules were applied based on a match condition configured in the match rule and were evaluated for every request/response. The WSP implements the Processing Policy and rules in a slightly different manner. The WSP allows for separate processing rules to be configured at any level of the WSDL. For example, you might want to apply a separate processing rule for each operation within the WSDL. The WSP offers this flexibility at multiple levels of granularity. From the most granular to the least, a processing rule can be configured at the operations level, the port level, the service level, the WSDL level, or the DataPower proxy level. During execution, this processing rule hierarchy is also evaluated and applied in this order. So if there is a request processing rule configured at the operation level and another rule configured at the WSDL level, the rule at the operation level will be evaluated and executed. When a WSP service is created, a default request rule is created at the proxy level that will contain one action. This rule and the action contained within it are discussed later in this chapter.

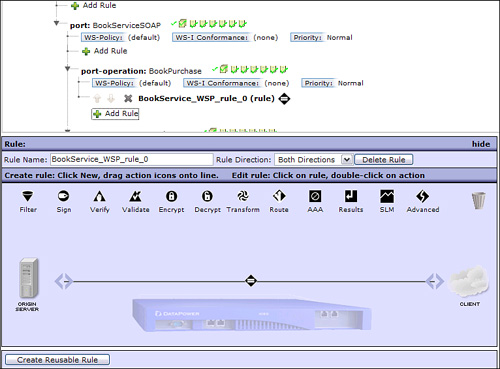
Add a note hereTo illustrate how this rule granularity can be useful, let’s take a look at the BookService\_WSP that we demonstrated earlier in this chapter. We saw that this service has an operation called BookQuery. Let’s suppose there is an additional operation that the service exposes called BookPurchase. As you would expect, this operation would be used to purchase a book. It may be that only registered users should be permitted to purchase books through this service and must first be authenticated. As we know, a DataPower service can provide this authentication and authorization within a processing rule by implementing a AAA policy. We will not discuss the configuration of the AAA policy here as it will be discussed in detail 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) and [Chapter 17](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3157#3157), [“Advanced AAA.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3157#3157) For now we know that it is an action that can be used within a processing rule to perform the authentication and authorization required.

Add a note hereA processing rule within a WSP can be configured by first navigating to the Web Service Proxy configuration page starting from the Control Panel, clicking the Web Service Proxy icon, and finally the Web Service Proxy name. This renders the WSDLs tab in the service configuration screen. Clicking the Policy tab at the top of the screen displays the Web Service Proxy Policy screen. This is where the Processing Policy can be configured by adding rules and actions.

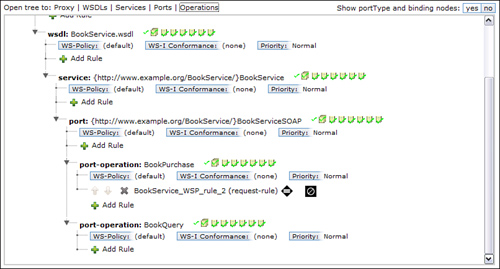
Add a note hereIn the main section of this configuration screen is a tree-like structure containing the five different levels of the proxy/WSDL at which rules can be configured. This tree will not be fully expanded, but it can be expanded one branch at a time, or the links above the tree can be utilized to open it to any of the five levels. Figure 10-7 shows this screen for our BookService\_WSP service. Notice that the tree is fully expanded to the operations level. You can see that default request and response rules were created for you at the proxy level. The response rule contains only a Match and a Results action. The request rule, on the other hand, has an additional action configured. This is a SLM (Service Level Monitor) action that can be used to monitor the traffic flowing through the service. We discuss this action in detail later in this chapter.

[](javascript:PopImage('IMG_221','http://images.books24x7.com/bookimages/id_30903/10fig07_alt.jpg','809','452'))  
Add a note hereFigure 10-7: Web Service Proxy Policy configuration screen.

Add a note hereAt each level of the policy tree is a plus (+) sign and the text Add Rule. This indicates that a processing rule can be configured here. Clicking this link opens up the Processing Policy line where you can configure a processing rule. This portion of the screen should look somewhat familiar as it is identical to the policy editor used in the Multi-Protocol Gateway and XML Firewall policy configuration. Figure 10-8 shows this screen after clicking on the Add Rule link under the BookPurchase operation within our BookService\_WSP service.

[](javascript:PopImage('IMG_222','http://images.books24x7.com/bookimages/id_30903/10fig08_alt.jpg','766','566'))  
Add a note hereFigure 10-8: Configuring a processing rule.

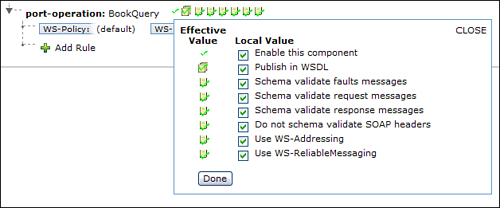
Add a note hereThe processing rule can now be configured by dragging actions onto the policy line and specifying the rule direction. Notice that a match rule is still required within the processing rule. A default match rule is created matching on all URLs; though only messages destined for this service operation will reach this rule, you can further filter the inputs by creating a more specific match rule. After configuring the processing rule and clicking the Apply button, the new rule is added to the service for only the BookPurchase operation. Figure 10-9 shows this new request rule containing a AAA policy action added to the BookPurchase operation. Now only requests for this operation will be authenticated and authorized per the AAA policy configuration contained within this request rule.

[](javascript:PopImage('IMG_223','http://images.books24x7.com/bookimages/id_30903/10fig09_alt.jpg','806','433'))  
Add a note hereFigure 10-9: New rule added to the BookPurchase operation.

Add a note hereWe now have in our BookService\_WSP two different request rules configured. We have the newly configured request rule at the BookPurchase operation level and the default request rule at the proxy level. Now when a request comes in for the BookPurchase operation, it will execute the rule containing the AAA policy, assuming that the match condition contained within this rule is satisfied. If the match rule was not satisfied, the service would continue to search up the tree until a match was found for the incoming request. Because we do not have a rule specified at the operation, port, or service level for the BookQuery operation, any requests for it will execute the default rule at the proxy level.

### Add a note hereUser Policy

Add a note hereIn addition to configuring processing rules at the various levels of the WSDL, there are also options that can be configured at each level. You will notice a series of icons with check marks just to the right of each level of the Processing Policy tree. Each of these icons represents an option that can be enabled or disabled at that level. This is called the user policy. Figure 10-10 shows options for the BookQuery operation in our BookService\_WSP service.

[](javascript:PopImage('IMG_224','http://images.books24x7.com/bookimages/id_30903/10fig10_alt.jpg','628','261'))  
Add a note hereFigure 10-10: User Policy options.

Add a note hereBy default, all the options in each of the user policies are enabled as indicated by a green check mark. To modify these options, simply click on the user policy icons and a checklist of all of the options is displayed as shown in Figure 10-10. The option can be selected and deselected as desired.

Add a note hereMost of the options shown in Figure 10-10 are self-explanatory; however, two may not be. These are the two WS-\* options. These are two of the WS-\* specifications that are supported within the WSP service. The first one in the list is WS-Addressing. WS-Addressing is a specification that enables routing information to be carried in the SOAP header. The second WS-\* option in this list is WS-ReliableMessaging. In short, this is a specification that allows for messages to be delivered reliably between endpoints. These two specifications and the support for them within a DataPower service are covered in more detail in [Chapter 19](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3524#3524).

## Front Side Handler Configuration (FSH)

Add a note hereWe already discussed FSH objects in [Chapter 9](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1276#1276), [“Multi-Protocol Gateway,”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1276#1276) so you should already know how to configure them. The FSH objects that are configured for a Web Service Proxy are no different than the ones you configured in the Multi-Protocol Gateway (MPGW); however, there are some differences in the way that they are implemented. We already saw in our first WSP example in this chapter how the FSH object is used in conjunction with a specified URI to form an endpoint for the client, so this is one way that it is configured differently. In this section we discuss other ways that this FSH/endpoint configuration is the same and also how it can be different from the MPGW service.

Add a note hereAs you already know, there are different types of FSH objects to support different protocols, as shown in [Chapter 9](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1276#1276); however, the MPGW is not the only service type that can take advantage of different protocols. Even if the backend Web service that your WSP is proxying supports only HTTP(s), your service can also receive requests from MQ. Keep in mind that the request document sent to the service must comply with the WSDL referenced by the WSP service. For example, suppose you had an MQ queue that holds valid Web service requests. You could add an MQ FSH to your Web Service Proxy and pull these requests from the queue, process them as you wish within your Processing Policy, and send them the backend Web service over HTTP(s). Be aware that the Web service will send a response back to the service. This response can be written to a response queue; however, if the response is to be ignored, you must set the ReplyToQ HTTP header to a null value. This can be done in the header injection tab of the service.

Add a note hereAnother important thing to remember about configuring the FSH in your WSP is that it can be reused. This is different from any other service you configure, where your FSH must be used by only one service. This is possible because the WSP does not only use the IP:port combination to listen for requests, it also includes the URI specified in the configuration as well as the operation being requested. This means that a FSH can be reused in the same or different Web Service Proxies within a single domain as long as the combination of the IP address, port, URI, and operation are unique. This is a common thing to do when your WSDL contains multiple bindings. This is because an endpoint must be configured for each binding specified in the WSDL. As an example, let’s take the BookService WSDL that we used in our BookService\_WSP service. We saw that this WSDL specified two different operations, but they were all contained within the same binding within the WSDL as shown in Listing 10-7. This is why when we showed the configuration for the WSP service, we were required to configure only one FSH and URI combination.

Add a note hereListing 10-7: Two Operations Within the Same Binding

Add a note here<wsdl:binding name="BookServiceSOAP" type="tns:BookService">

<soap:binding style="document"

transport="http://schemas.xmlsoap.org/soap/http"/>

<!-- -->

<wsdl:operation name="BookQuery">

<soap:operation

soapAction="http://www.example.org/BookService/BookQuery"/>

<wsdl:input name="BookQueryRequest">

<soap:body use="literal"/>

</wsdl:input>

<wsdl:output>

<soap:body use="literal"/>

</wsdl:output>

</wsdl:operation>

<wsdl:operation name="BookPurchase">

<soap:operation

soapAction="http://www.example.org/BookService/BookPurchase"/>

<wsdl:input name="BookPurchaseRequest">

<soap:body use="literal"/>

</wsdl:input>

<wsdl:output>

<soap:body use="literal"/>

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

Add a note hereLet’s suppose that this WSDL had the same two operations but they were listed in two separate bindings within the WSDL, such as the example shown in Listing 10-8. Notice that there are two binding elements specified, each containing an operation. This is a common occurrence, so let’s take a look at what happens when this is referenced by our BookService\_WSP service.

Add a note hereListing 10-8: Two Binding Elements with Two Operations

Add a note here<wsdl:binding name="BookQuerySOAP" type="tns:BookQuery">

<soap:binding style="document"

transport="http://schemas.xmlsoap.org/soap/http"/>

<!-- -->

<wsdl:operation name="BookQuery">

<soap:operation

soapAction="http://www.example.org/BookService/BookQuery"/>

<wsdl:input name="BookQueryRequest">

<soap:body use="literal"/>

</wsdl:input>

<wsdl:output>

<soap:body use="literal"/>

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

<!-- -->

<wsdl:binding name="BookPurchaseSOAP" type="tns:BookPurchase">

<soap:binding style="document"

transport="http://schemas.xmlsoap.org/soap/http"/>

<wsdl:operation name="BookPurchase">

<soap:operation

soapAction="http://www.example.org/BookService/BookPurchase"/>

<wsdl:input name="BookPurchaseRequest">

<soap:body use="literal"/>

</wsdl:input>

<wsdl:output>

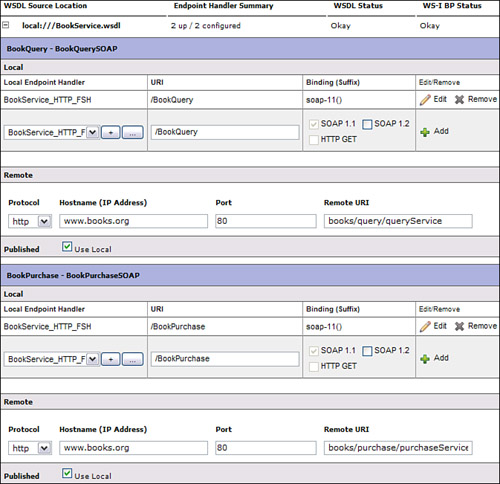
<soap:body use="literal"/>

</wsdl:output>

</wsdl:operation>

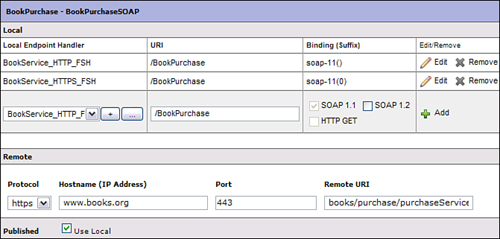
</wsdl:binding>

Add a note hereAs the WSDL referenced by our BookService\_WSP service is changed to contain the two bindings as shown in Listing 10-8, you will see that instead of one local, remote, published configuration, there are two. Figure 10-11 shows the service with the new BookService WSDL referenced. Notice that we now have two sections to complete, each containing a FSH/URI combination and a remote endpoint configuration. Also notice that we are using the same FSH object for each with a different URI. That could have just as easily been different FSH objects. For example, because the BookPurchase operation may contain sensitive data, you would probably configure an HTTPS FSH for that operation. As you can see, it is also possible to configure different backends, or endpoints, for each operation—so we could also configure this operation to go to the backend Web service over an encrypted transport such as HTTPS.

[](javascript:PopImage('IMG_225','http://images.books24x7.com/bookimages/id_30903/10fig11_alt.jpg','673','651'))  
Add a note hereFigure 10-11: Configuring multiple bindings within a WSP.

Add a note hereBecause these two operations are separated into two separate bindings, the Processing Policy would reflect that and display two binding sections each with one operation. In this case, you could configure processing rules at the binding level or at the operation level as we saw in the first example.

Add a note hereIt is also possible to configure more than one FSH object for the same WSDL and binding. For example, suppose we wanted to provide the ability to call the BookPurchase operation in our BookService\_WSP over HTTP or HTTPS. We could add a second HTTPS FSH object to this binding in our configuration as shown in Figure 10-12. A new FSH configuration was added called BookService\_HTTPS\_FSH to the BookPurchaseSOAP binding. The operations contained within this binding (BookPurchase) can now receive requests by both FSH objects. Although it is not shown in this example, the URI that is used can also be different for this new configuration.

[](javascript:PopImage('IMG_226','http://images.books24x7.com/bookimages/id_30903/10fig12_alt.jpg','671','321'))  
Add a note hereFigure 10-12: Adding a second FSH object.

Add a note hereNow that we have two FSH objects configured for our BookPurchaseSOAP binding, it is important to take a look again at our Processing Policy. This is another area where the WSP service is different from the MPGW service. When we added additional FSH objects to a MPGW, all requests handled by any of these objects would evaluate the same set of processing rules to determine what rule would be executed on the request, response, or error. When multiple FSH objects are added to a WSP, however, this is not the case. You saw what happened to our Processing Policy when we put each of our operations within the WSDL into separate bindings. Another binding was introduced in the policy where rules could be configured. When we add the additional FSH to our service, the same thing happens. This is because adding another listening protocol and port is adding another binding to your service. This is how the DataPower service virtualizes the endpoints. So just because you added another FSH to your service does not mean that the same processing rules will be applied to each even though they are the same operation. Figure 10-13 shows our Processing Policy after this second FSH was added to our BookPurchaseSOAP binding. Notice that the BookPurchaseSOAP binding is still there containing a BookPurchase operation along with the rule that we configured for it. Looking a little closer you will realize that a second binding was created called BookPurchaseSOAP.0. This is the binding for our new HTTPS FSH. So this means that the rule containing the AAA action for our BookPurchase operation will not fire if the request is sent over HTTPS to the new FSH configured. We must configure another rule for this binding.

[](javascript:PopImage('IMG_227','http://images.books24x7.com/bookimages/id_30903/10fig13_alt.jpg','802','432'))  
Add a note hereFigure 10-13: Processing policy with multiple FSH objects.

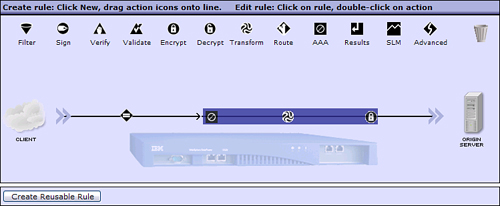
### Add a note hereReusable Rules

Add a note hereAlthough the concept of reusable rules is not unique to the Web Service Proxy Service, this might be a good place to discuss them, because it can solve the problem we previously discussed when adding multiple FSH objects to a single binding.

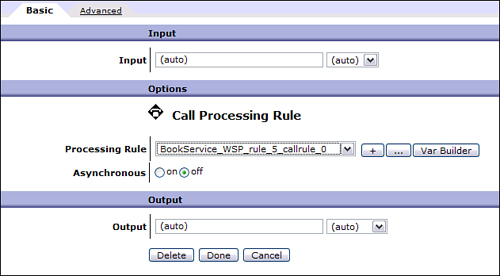
Add a note hereA reusable rule is simply a rule, or part of a rule, that can be called from within another rule. After the callable rule is finished, it returns the results back to the calling rule where the rule processing resumes. This call can be made by implementing a call or on-error action in a processing rule. To demonstrate how to configure and call these rules, let’s take a look at our previous example in Figure 10-13.

Add a note hereIn this example, we demonstrated what happens to a Processing Policy when multiple FSH objects are added to a single binding within a WSDL. In this example, there was only one AAA action in the BookPurchase operation, so it would be easy enough to just add that same AAA policy to the second BookPurchase request rule. This would not be as simple if there were several actions within the request rule for this operation. For example, let’s suppose that after the AAA action there were a Transform action and an Encrypt action in this request rule. This would be more difficult to duplicate in the second rule. By making this rule a reusable rule, we could have requests handled by the second FSH object execute the same processing rule by calling it. Let’s see how this is done.

Add a note hereTo create a reusable rule, you must first start with an existing rule that has already been configured. In our example we have the request rule containing the AAA action, a Transform action, and an Encrypt action for the BookPurchase operation. To begin, the rule must be displayed in the policy editor, which can be done by simply clicking the rule within the Web Service Proxy Policy. After the rule is displayed in the policy editor, clicking the Create Reusable Rule button causes the cursor to change to a crosshair. Left-clicking and dragging the crosshair over the actions that are to be contained in the reusable rule creates a new reusable rule consisting of those actions. The actions to be included are identified by a blue highlighted box. Figure 10-14 shows our request rule for the BookPurchase operation where we have combined all three actions to form a new reusable rule.

[](javascript:PopImage('IMG_228','http://images.books24x7.com/bookimages/id_30903/10fig14_alt.jpg','749','308'))  
Add a note hereFigure 10-14: Creating a reusable rule.

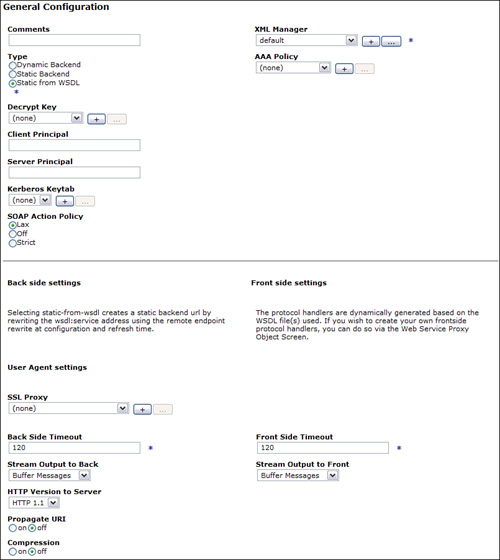
Add a note hereNow that we have a reusable rule for the BookPurchase operation, we can call it from the second binding that was created when we added our HTTPS FSH to the BookPurchase binding. To do this we added a new request rule to this second BookPurchase operation and added an Advanced action to the rule. We then configured this Advanced action as a Call Processing Rule type, selecting the new callable rule as the processing rule to call, as shown in Figure 10-15. Notice that the name given to the reusable rule when it was created was the name of the actual request rule that it was created from with \_callrule\_0 appended to it.

[](javascript:PopImage('IMG_229','http://images.books24x7.com/bookimages/id_30903/10fig15_alt.jpg','631','348'))  
Add a note hereFigure 10-15: Configuring the Call Processing Rule action.

Add a note hereWe have now configured the three actions in the BookPurchase operation to be a reusable rule. The second BookPurchase operation that was added for the HTTPS FSH now contains a request rule with a Call Processing Rule action to call this reusable rule. Now any request to the BookPurchase operation will execute the same three actions as desired.

## Proxy Settings

Add a note hereAs you have seen, the WSDL file referenced by the Web Service Proxy describes many of the details for calling the actual Web service, enabling the proxy to automatically produce a base configuration. There are, however, many configuration options within the service that cannot be derived from the WSDL and that can be set at the proxy level. Many of these configuration options are found in the Proxy Settings tab on the service configuration screen. Figure 10-16 shows this configuration tab with all of the default settings. You should notice the similarities between this configuration screen and the General service configuration screen in the MPGW service.

[](javascript:PopImage('IMG_230','http://images.books24x7.com/bookimages/id_30903/10fig16_alt.jpg','805','901'))  
Add a note hereFigure 10-16: Proxy Settings tab

### Add a note hereDefining the Backend

Add a note hereWhen the Web Service Proxy is created, the endpoint listed in the WSDL is used to automatically populate the endpoint, or backend, within the service. There are times however, when this is not the actual endpoint that you want your DataPower service to forward requests to. This is when you can utilize the flexibility that the WSP offers when defining the endpoint.

Add a note hereWhen defining the backend for your Web Service Proxy, there are three different options that you can choose. You can keep the endpoint described in the WSDL, you can provide your own static backend, or you can specify a dynamic backend. The default behavior is to use the endpoint specified in the WSDL, but this can be changed under the Type heading in the Proxy Settings tab.

Add a note hereIf you choose not to use the endpoint specified in the WSDL but still want to define a static backend, you can specify the new endpoint in two different ways. The first option is to type directly over the values in the Remote section on the WSDL tab with the new endpoint. This might be the best option if there are multiple endpoint configurations and you want to change only one. But what if you have multiple bindings, each requiring an endpoint definition and you want all requests to go to one single endpoint? This is where you can use the Static Backend option. Figure 10-17 shows this option selected. Notice what happens when the Type is set to Static backend. The Backend URL field is shown where you would specify the backend—but notice the buttons below it. There are buttons for generating the Backend URL for protocols other than HTTP. All the DataPower supported protocol options are available for the backend. In fact, the backend does not even have to be an actual Web service.

[](javascript:PopImage('IMG_231','http://images.books24x7.com/bookimages/id_30903/10fig17_alt.jpg','783','495'))  
Add a note hereFigure 10-17: Selecting a Static backend in a WSP.

Add a note hereSuppose you had many clients on different platforms and programming languages that all needed to send files to an FTP server and you needed them to all be in a specific format. There are a couple of challenges you might face. First, you would need to be sure that they can all communicate over FTP. If they can communicate with your FTP server, you would then be required to provide them with the details for connecting to the server. If this is possible, you will need to communicate the format of the message to each potential client. This could be difficult if the message is not an XML document. Now think about how convenient it would be to generate and publish a WSDL that would represent the message to be sent to the FTP server but in an XML format. You could then create a Web Service Proxy on the DataPower appliance that would reference that WSDL and enforce it on all requests. You could even authenticate and authorize the request before sending to the backend FTP server. Any response codes back from the server can then be converted back into a SOAP response and sent back to the client. In this scenario the clients never know the backend implementation details because they think they are sending a simple Web service request over HTTP(s). Now when the backend server changes the end clients will never have to know.

Add a note hereThe last option available for the backend is to have a dynamic backend. As we discussed 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) a dynamic backend simply indicates that the backend server will be decided within the Processing Policy as each request is processed. This is no different in a WSP service; however, the flexibility of the Processing Policy makes it easier to implement. For example, you might have a different backend for each operation within the Web service. Because the Processing Policy allows for rules to be configured at the operation level, a rule can be configured for each operation containing a Route action. The hierarchy of the WSDL within the Processing Policy can also be taken advantage of when specifying a backend. Suppose you had a Web service with many operations and you wanted requests for all of the operations, except for one, to be forwarded to the same backend address. You could configure a rule at the WSDL level that sets the backend server via a Route action for the majority of the operations. You would then configure a rule with a different Route action for the one operation requiring a different backend.

### Add a note hereDecrypt Key

Add a note hereThere may be times when the Web service request message being sent is expected to be encrypted by the client. Because the DataPower device supports most of the popular encryption algorithms, these messages can be decrypted within your service. In all other service types, this decryption is performed via a Decrypt action that would be added to a request rule. This is also possible to do in a WSP service; however, it presents some issues. Before the message even gets to the Processing Policy, it is validated for WSDL compliance, which includes schema validation and valid operations being requested. If the request message is encrypted, this would not be possible. This is why the Decrypt key option is available.

Add a note hereConfiguring a Decrypt key within the service provides the service the appropriate key to decrypt the request message. This key is used to decrypt the message as it enters the service before it is validated for WSDL compliance. Now the decrypted message can be validated and passed to the Processing Policy.

Add a note hereCreating this Decrypt Key object is simple. Clicking the plus (+) sign next to the Decrypt Key drop-down box on the Proxy Settings tab shown in Figure 10-16 will display the Configure Crypto Key screen. The private key can then be selected that should be used for decrypting the message. More information about key and certificate management can be found in [Chapter 18](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3288#3288), [“DataPower and SSL.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3288#3288)

Add a note hereAlthough the Decrypt Key object provides a way to specify the key that will decrypt the incoming document, it is not always necessary. If the request document contains a Binary Security Token (BST) in the WS-Security header, the WSP will use the certificate within it and attempt to find the corresponding key within a configured Crypto Identification Credential object on the device. If the key is found, it is used to decrypt the message.

### Add a note hereSOAP Action Policy

Add a note hereOften a WSDL specifies a soapAction parameter with a URI as its value. This indicates that a soapAction HTTP header with the corresponding value should be sent along with all requests for that particular operation. The purpose of this header is to indicate the intent of the request; however, there are no restrictions on what this value must or must not be as long as it is a URI.

Add a note hereThe Web Service Proxy has an option on the Proxy Settings tab shown in Figure 10-16 named SOAP Action Policy. This indicates how strictly this header specified in the WSDL will be enforced. The default value is Lax, which indicates that an empty header is considered a match. The other two options are Strict, which indicates that the request must contain the soapAction header with an exact match, and Off, which does not check the header

## WSDL Configuration and Management

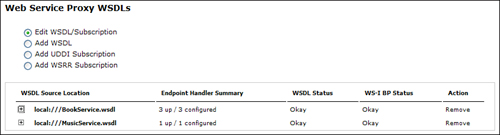
Add a note hereAs we have stated several times throughout this chapter, the WSDL describes the backend Web service and provides many of the configuration details for the Web Service Proxy. Because the service uses much of the information specified in the WSDL for its configuration, it is important that the DataPower service be notified when it is changed. This section explains the different methods for managing these changes and the different methods for dynamically looking up a WSDL or even combining multiple WSDLs within a single service.

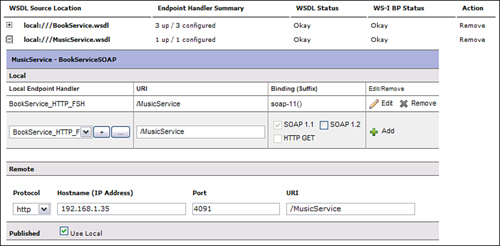
### Add a note hereAdding Additional WSDLs

Add a note hereAfter a Web Service Proxy is created using a single WSDL that describes the backend Web service, it is possible to add additional WSDLs to the service. This would provide the convenience of combining two WSDLs within a single DataPower service. Let’s take our BookService\_WSP service as an example. Suppose the company providing this service acquired an online music store and can now offer music as well as books. It just so happens that the online music store that was acquired already has a Web service for this. It would be very convenient for our customers to view both of these Web services as one. This is entirely possible using our Web Service Proxy service by simply adding this new WSDL to our service.

Add a note hereAdding multiple WSDLs to an existing service is easy to do. From the WSDL tab in the Web Service Proxy configuration screen, selecting the Add WSDL radio button renders a screen with the familiar Upload and Fetch buttons that can be used to add the new WSDL. This process is the same as the process used to add the first WSDL when the service was initially created. An FSH and URI must be entered in the Local section to reflect the endpoint that the client will use to access the service. As we saw in the previous examples, the Remote information is taken from the WSDL. As we discussed earlier in this chapter, this backend information can be changed in several different ways.

Add a note hereAfter all the required information for the new WSDL is entered, you will see that the second WSDL has been added to the service. Now, as you can see in Figure 10-18, we have two WSDLs configured within our BookService\_WSP. Also, notice the Endpoint Handler Summary for each. The BookService.wsdl indicates 3 up / 3 configured. This is because we have two bindings within that WSDL and we also added one additional FSH to it. The MusicService.wsdl only shows 1 up / 1 configured. This is because there is only one binding within this WSDL and we added only one FSH for it. If we expand the MusicService.wsdl as shown in Figure 10-19, we can see that we configured it to use the same HTTP Front Side Handler object as we did with the BookService.wsdl. You can also see that this new WSDL requires another Remote configuration that provides a new backend protocol, Hostname, port, and URI. This allows us to have separate backends for each of the WSDLs added to the service.

[](javascript:PopImage('IMG_232','http://images.books24x7.com/bookimages/id_30903/10fig18_alt.jpg','816','220'))  
Add a note hereFigure 10-18: Second WSDL added to the Web Service Proxy.

[](javascript:PopImage('IMG_233','http://images.books24x7.com/bookimages/id_30903/10fig19_alt.jpg','777','382'))  
Add a note hereFigure 10-19: MusicService.wsdl endpoint configuration.

Add a note hereNow that we have seen the WSDL configuration for adding additional WSDLs to a service, let’s see how the Processing Policy looks. Because the Processing Policy reflects the hierarchy of the WSDL configured with the proxy itself being the highest level, adding a new WSDL adds a new WSDL sub-tree under the proxy within the policy. This enables processing rules to be configured for the new WSDL providing the same level of granularity as the first WSDL added. Figure 10-20 shows the Processing Policy in our example service after adding the MusicService.wsdl. We have expanded only the MusicService level to show the port and operations within it. We now can configure rules for both WSDLs at any level of granularity.

[](javascript:PopImage('IMG_234','http://images.books24x7.com/bookimages/id_30903/10fig20_alt.jpg','801','403'))  
Add a note hereFigure 10-20: Processing Policy with the MusicService.wsdl added.

### Add a note hereWSDL Cache Policy

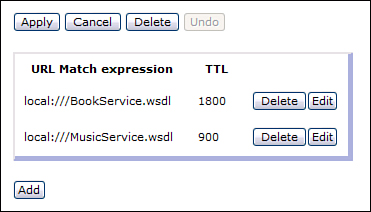
Add a note hereAfter a service has been configured, it is possible that the WSDL that it is configured to use will change over time. Because the service actually compiles and caches the WSDL, there needs to be some mechanism to tell the service that it has changed and to refresh the cache. To do this, we can configure a WSDL refresh policy within our WSP that dictates how often this cache will be refreshed.

Add a note hereTo configure a WSDL cache policy, you must access the Web Service Proxy configuration screen from the left navigation menu; go to Objects→Services→Web Service Proxy, and then select the service you want to configure it for. After the service configuration screen is rendered, the WSDL Cache Policy tab can be selected from the top menu. Note that this link will not be visible at first. Scrolling this top menu to the right is required to access it. Figure 10-21 shows this configuration screen.

[](javascript:PopImage('IMG_235','http://images.books24x7.com/bookimages/id_30903/10fig21_alt.jpg','830','299'))  
Add a note hereFigure 10-21: WSDL Cache Policy configuration screen.

Add a note hereFrom this screen you can now click the Add button to configure a new WSDL cache policy for your service. There are two fields required for each policy. The first field is the URL Match expression. This field enables you to define which WSDLs are refreshed by this policy. This is the URL of the WSDL file which can be a literal representation or a wildcarded URL. For example, if we wanted to just have a refresh policy that applies only to the BookService.wsdl, which resides in the local directory of the DataPower file system, you could add local:///BookService.wsdl in this field. However, if you wanted this policy to be applied to all WSDLs within the service, you simply put an \* as a wildcard indicating all. Although our example shows a WSDL cache policy for a WSDL that resides on the local file system, the real benefit of this policy is realized when it is retrieved remotely. In this case the match expression might be something like <http://services.mycompany.com/>\*.wsdl.

Add a note hereThe second field you can specify is the TTL (Time to Live) parameter. This specifies how often the WSDL will be refreshed. This is entered in seconds and a default value of 900 seconds is pre-populated. If a new value is entered, it must be an integer between 5 and 86400. Figure 10-22 shows a WSDL cache policy for our BookService\_WSP. As you can see, the BookService.wsdl is refreshed only every 1800 seconds, whereas the MusicService.wsdl is refreshed every 900 seconds.

  
Add a note hereFigure 10-22: WSDL cache policy.

Add a note hereAfter a WSDL cache policy is created and applied to the service, the WSDLs will be refreshed at the specified intervals. Once it is refreshed it is also reflected in the Web Service Proxy configuration. Although the service configuration is updated automatically, it is important to know what changes are being made to the WSDL and how they can affect the service. For example, if new binding is added to a WSDL and the service is updated to reflect it, you will still need to configure a new endpoint configuration for it.

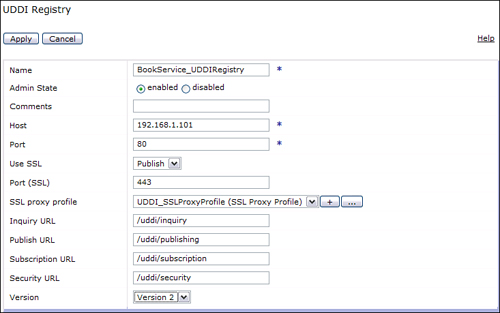
### Add a note hereUDDI

Add a note hereBecause many Web services are available to the general public or external business partners, there is a challenge that is presented. We know that a WSDL describes the Web service providing all of the required information to the consumer, but how does the service provider publish this WSDL so people who wish to consume the service can find this Web service and its operations? The open industry initiative called Universal Description Discovery and Integration (UDDI) was created to solve this specific challenge. UDDI is an XML-based registry specification that describes how Web service providers can publish a WSDL and how consumers can find it. This registry can be made available over the Internet or private networks. This can be thought of as a *Yellow Pages* for Web services. Consumers of the services can be configured to dynamically look up the WSDL for the services of interest and know exactly how and where to call them.

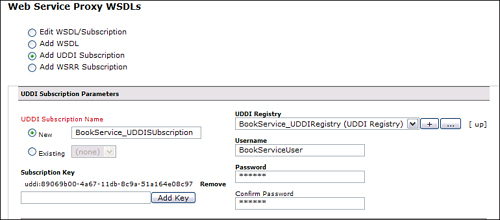
Add a note hereWe have seen when configuring a Web Service Proxy that a WSDL must be referenced to gather the required service information. It is possible to have the DataPower service automatically obtain the WSDL from a UDDI registry or to browse the registry for one when configuring the service. To have DataPower automatically obtain a WSDL from the UDDI registry you must add a UDDI subscription within your service. From the WSDL configuration tab within the service configuration, you will see a radio button labeled Add UDDI Subscription. Selecting this radio button presents the screen for configuring a new UDDI subscription.

Add a note hereA UDDI subscription object provides the information that enables the DataPower service to retrieve the WSDL and also to be notified of any updates to it. The UDDI subscription object defines all of the parameters for subscribing to a particular WSDL; however, it does not define the UDDI registry connection details. This is defined within a UDDI Registry object, which is then referenced by the UDDI subscription object.

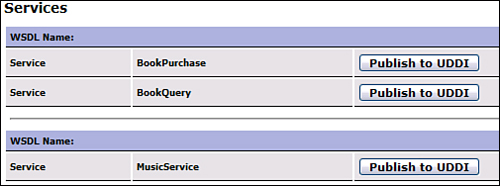
Add a note hereAs with most objects within DataPower, the UDDI Registry object can be configured from the Objects section in the left navigation menu, or we can configure it by clicking the plus + sign within the UDDI subscription configuration. In either case, the UDDI Registry object is configured via the configuration screen shown in Figure 10-23. When configuring this object, you must supply a name for the object as well as the host and port of the UDDI registry. In addition to these required fields, you can also specify if the connection to the UDDI registry should use SSL for publishing only or for all lookups. In either case a SSL proxy profile must be defined along with the SSL port for the registry. You can also define the URLs used by the registry for inquiries, publishing, and subscribing. This is left as the default in many cases. The last field defines the version of UDDI that will be used.

[](javascript:PopImage('IMG_237','http://images.books24x7.com/bookimages/id_30903/10fig23_alt.jpg','733','459'))  
Add a note hereFigure 10-23: UDDI Registry configuration.

Add a note hereAfter the UDDI Registry object is configured, you can then continue to configure the UDDI subscription object. The remainder of this configuration requires only a few remaining details. As usual, the object requires a name. A username and password are required for connecting to the registry and finally, the Subscription Key(s) are required for locating the WSDL within the registry. This key can be found in the UDDI registry for the WSDL document you are subscribing to. To add a key to this subscription, you must enter the key and click the Add button. Many times there will be more than one key required for locating the particular service. Figure 10-24 shows a completed UDDI subscription object. Once this object is configured within the WSP, it will appear along with the WSDLs within the service requiring the same endpoint configuration as when a WSDL was added locally.

[](javascript:PopImage('IMG_238','http://images.books24x7.com/bookimages/id_30903/10fig24_alt.jpg','770','339'))  
Add a note hereFigure 10-24: Completed UDDI Subscription object.

Add a note hereIn addition to DataPower using a WSDL for its service configuration, it can also publish a WSDL to a UDDI registry. When publishing WSDLs to an external registry, DataPower uses the endpoint configuration specified in the Publish section of the service configuration within the WSDL tab. This Publish section can specify a new service endpoint, or it can use the endpoint that is specified in the Local section of the configuration. The Local section represents the actual endpoint that the consumers connect to within the DataPower device, that is, the virtualized endpoints that the service creates consisting of the hostname, listening port from the FSH, and the URI. This will be embedded in the appropriate location in the original WSDL and published by DataPower, so subscribers can consume the Web service(s) proxied by the Web Service Proxy. This can be accomplished from the Services tab in the Web Service Proxy configuration screen as shown in Figure 10-25. As you can see, each binding within each WSDL can be published separately. Clicking the Publish to UDDI button for a particular binding will then prompt you for a UDDI Registry object that must be configured on the device, and subsequently a username and password for authenticating to the registry.

[](javascript:PopImage('IMG_239','http://images.books24x7.com/bookimages/id_30903/10fig25_alt.jpg','505','188'))  
Add a note hereFigure 10-25: Services tab in a WSP Configuration.

Add a note hereAfter a service is published to the UDDI registry, subscribers to that service can retrieve the WSDL, which will contain the DataPower Web Service Proxy as the endpoint instead of the original service endpoints. This truly reflects the power of the Web Service Proxy to virtualize the Web service endpoints. This virtualizes and hides the actual backend endpoints forcing all the consumers to go through the DataPower Web Service Proxy.

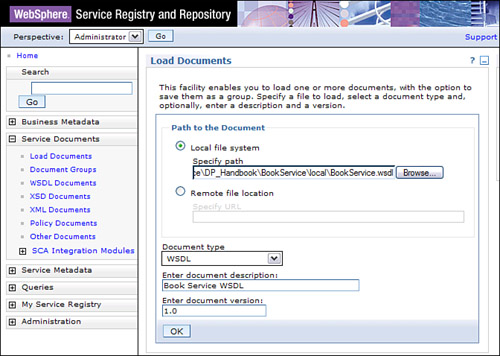
### Add a note hereWSRR

Add a note hereAlthough UDDI provides a solution to Web Service publishing and subscribing, it is a rudimentary approach that may not suit your needs. For more robust requirements, DataPower also supports the IBM WebSphere Service Registry and Repository (WSRR), from which a WSDL can be referenced. WSRR enables you to quickly and easily publish, find, enrich, manage, and govern services and policies within your SOA. We do not discuss all the additional value of WSRR in this book as you can find that information in your WSRR documentation; however we focus on the integration with the product from within a DataPower Web Service Proxy.

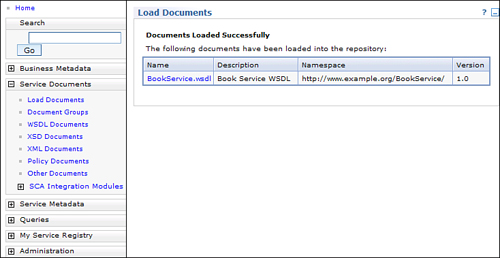
#### Adding a Document to WSRR

Add a note hereLet’s start by taking a look the WSRR administration console. WSRR is actually a Java EE application that runs within a WebSphere Application Server (WAS) and can be administered through a Web browser. If you are familiar with the WAS administration console, it may look somewhat familiar because it uses the Integrated Solutions Console that provides a similar look and feel. Although many different types of documents and metadata can be stored within WSRR, we only talk about WSDLs in this chapter.

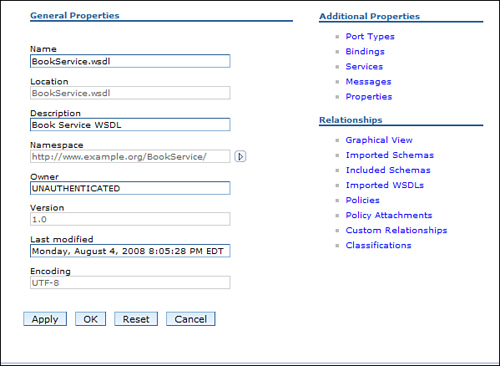
Add a note hereBefore you can do anything with the product, you must first install all the supporting products and WSRR itself. WSRR requires a database (DB2, Oracle, or Cloudscape®) and a WebSphere Application Server to be installed. For complete installation instructions, please refer to the WSRR documentation. Once all of the products are installed and deployed, you can access the administration console through a browser. Once you have logged into WSRR, adding a WSDL is a simple process. From the administration console, you would navigate from the left menu to Service Documents→Load Documents as shown in Figure 10-26.

[](javascript:PopImage('IMG_240','http://images.books24x7.com/bookimages/id_30903/10fig26_alt.jpg','672','478'))  
Add a note hereFigure 10-26: Loading a document in WSRR.

Add a note hereAs you can see, there are not many fields to be entered on the screen presented. By clicking the Browse button, you can navigate to the WSDL on your local computer. You can also reference a remote location for retrieving the WSDL by providing a URL. You can then optionally add a description and a document version. Notice in Figure 10-26 that we are loading the Book Service.wsdl file and entering a version of 1.0 (version is optional). Clicking OK and Finish loads the WSDL in the repository, and a message is displayed that the document has been loaded successfully as shown in Figure 10-27.

[](javascript:PopImage('IMG_241','http://images.books24x7.com/bookimages/id_30903/10fig27_alt.jpg','734','379'))  
Add a note hereFigure 10-27: WSDL successfully loaded to the repository.

Add a note hereSo we said that the WSDL was loaded into the repository but what does that mean to us? Let’s take a look at the WSDL within WSRR. Again, through the WSRR administration console, we can navigate to Service Documents→WSDL Documents from the left navigation menu. Or, if you are still on the success screen after loading the document, you can click the filename. In our example we can click the BookService.wsdl shown in Figure 10-27. This displays the details of the document loaded. Figure 10-28 shows the BookService.wsdl details within the WSRR administration console.

[](javascript:PopImage('IMG_242','http://images.books24x7.com/bookimages/id_30903/10fig28_alt.jpg','599','438'))  
Add a note hereFigure 10-28: BookService.wsdl within WSRR.

Add a note hereThere are only a handful of details shown here but there are many links that can display additional details if they are available. Of particular interest to us are the Name and the Namespace on this screen, as this is how we will reference the file from our DataPower service. There is also a Last modified date and time as well as the version number that we had given it when we loaded it.

Add a note hereSome other links from the WSDL document screen shown in Figure 10-28 are under the Additional Properties headings. Most of these links should look somewhat familiar to you as they are the different parts of a WSDL that we discussed earlier in this chapter. These can be selected to view the applicable information for the WSDL document. For example, Figure 10-29 shows the Bindings within our BookService.wsdl file. We see the two bindings that we created for this service. Each binding can also be selected to view information about it. Notice that each entry also has a Name and a Namespace associated with it. This is because individual sections of the WSDL can be retrieved independently from WSRR.

[](javascript:PopImage('IMG_243','http://images.books24x7.com/bookimages/id_30903/10fig29_alt.jpg','950','270'))  
Add a note hereFigure 10-29: Bindings within the BookService.wsdl.

Add a note hereAlthough there is much more that you can do with WSRR, simply adding a WSDL as shown in this section is all that is required for a DataPower service to subscribe to it.

#### Subscribing to WSRR from a DataPower Service

Add a note hereNow that we have a WSDL loaded into WSRR we can talk about referencing it from a DataPower service. Much like subscribing to a UDDI registry from a DataPower service, a WSRR subscription must also be created. This object references a WSRR Server object that provides all the connection details for connecting to the repository, and then you will need to configure a subscription describing what you want to retrieve from it.

Add a note hereYou can begin to configure these required objects from within the WSDL tab of the service configuration screen because you are essentially adding a WSDL to the service. Selecting the Add WSRR Subscription displays the screen for configuring the objects.

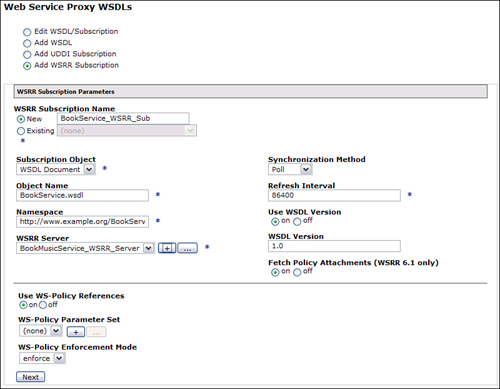
Add a note hereOn this screen there are few parameters here that need to be supplied. To start, you need to provide a name for this Object. You will then be required to add a WSRR Server object; an existing one can be referenced or edited, or a new object can be created. The configuration screen for this object is shown in Figure 10-30. The only two required fields for this object are a name and a SOAP URL where the WSRR server can be contacted. If you are contacting the WSRR server over HTTPS, you will also need to specify a SSL Proxy Profile object. If your WSRR repository requires authentication, you will need to provide the username and password here as well.

[](javascript:PopImage('IMG_244','http://images.books24x7.com/bookimages/id_30903/10fig30_alt.jpg','753','455'))  
Add a note hereFigure 10-30: WSRR Server Object configuration screen.

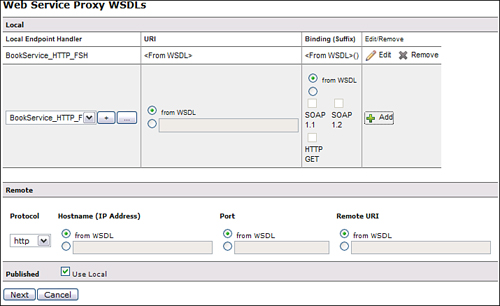
Add a note hereNow that we have a name for our WSRR Subscription object and a WSRR Server Object, we can specify the subscription details for the document that we want to retrieve. To indicate this, we must provide the name of the document, the namespace, and optionally the version. Note that the option to subscribe to a particular version of a document from DataPower is only available in firmware 3.7.1 and later.

Add a note hereIf you recall from Figure 10-27 when we loaded the WSDL into the WSRR repository, there was a name, a namespace, and a version specified. For our example document, the name was BookService.wsdl, the corresponding namespace was [http://www.example.org/BookService/,](http://www.example.org/BookService/) and the version was 1.0. This is what we enter in the corresponding fields for our WSRR Subscription object. Before entering the version, the Use WSDL Version radio option must be set to on. Also notice the field labeled Synchronization Method, which indicates how the WSDL will be synchronized with the repository. The options are Poll, where the object will poll the repository at the interval specified in the Refresh Interval field, and Manual, where you would need to manually refresh the WSDL.

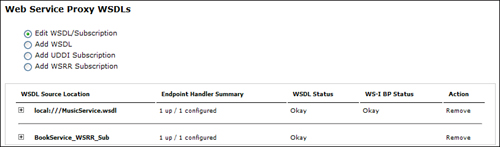
Add a note hereFigure 10-31 shows our completed WSRR Subscription for the BookService.wsdl. Another interesting option available in firmware versions 3.7.1 and later is the Fetch Policy Attachments option in the WSRR subscription. This allows for the DataPower service to retrieve any Web Service policy attachments that are stored in the repository for a given WSDL. Policy attachments are discussed further in [Chapter 19](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3524#3524).

[](javascript:PopImage('IMG_245','http://images.books24x7.com/bookimages/id_30903/10fig31_alt.jpg','800','622'))  
Add a note hereFigure 10-31: Completed WSRR Subscription.

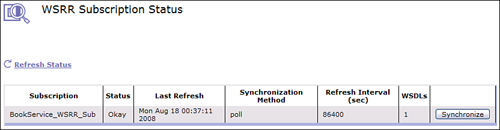
Add a note hereClicking Next from this screen completes the WSRR Subscription, but remember, this is still just a WSDL we are adding, and we will need to add an FSH object. You will notice a slight difference when configuring the Local and Remote sections for this WSDL. The fields that are usually populated with information from the WSDL are no longer populated. This is because this information is not yet known by our service as it will retrieve it from the repository. Instead you will see a radio button labeled From WSDL for the information that can be obtained from the WSDL. There is also a radio button that enables you to override this information. Figure 10-32 shows this configuration screen where we added one FSH object for the Local Endpoint handler and left the remaining fields to be obtained from the WSDL.

[](javascript:PopImage('IMG_246','http://images.books24x7.com/bookimages/id_30903/10fig32_alt.jpg','715','437'))  
Add a note hereFigure 10-32: Endpoint configuration screen.

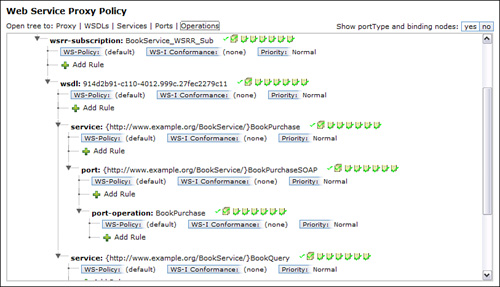
Add a note hereAfter you have created the endpoint configuration and clicked Next, you have completed the subscription. You should now see the name of the WSRR Subscription object where you would normally see the WSDL filename and a status of Okay if everything is configured properly. Figure 10-33 shows our BookService\_WSP with the MusicService.wsdl being retrieved from the local file system and the BookService.wsdl being retrieved from WSRR via the BookService\_WSRR\_Sub object.

[](javascript:PopImage('IMG_247','http://images.books24x7.com/bookimages/id_30903/10fig33_alt.jpg','815','240'))  
Add a note hereFigure 10-33: WSDLs added to the service.

Add a note hereAs an added checkpoint, the status of the WSRR subscription can be checked by navigating to Status→Web Service→WSRR Subscription Status in the left-side menu. Here the status of each WSRR Subscription created within the working domain can be viewed. The information provided on this screen for each subscription is the status, last refresh date and time, synchronization method, refresh interval, and number of WSDLs. There is also a button labeled Synchronize that enables the manual synchronization, or refresh, of the WSDL(s). Figure 10-34 shows this screen with our newly created WSRR subscription.

[](javascript:PopImage('IMG_248','http://images.books24x7.com/bookimages/id_30903/10fig34_alt.jpg','832','217'))  
Add a note hereFigure 10-34: WSRR Subscription status.

Add a note hereBecause we can’t see any details about the BookService.wsdl file within the WSDLs tab of our service, let’s take a look at the Policy tab to see what it looks like after adding this new subscription. Figure 10-35 shows our policy after adding the WSRR Subscription for the BookService.wsdl. Notice that all the bindings and operations are shown where rules can be configured, but there is an additional level at the top of this hierarchy. This is the subscription level.

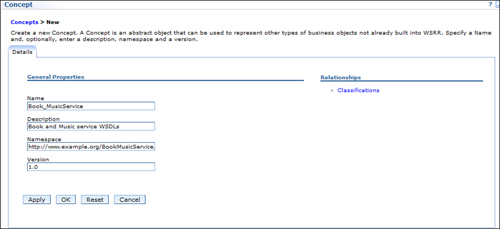
[](javascript:PopImage('IMG_249','http://images.books24x7.com/bookimages/id_30903/10fig35_alt.jpg','813','466'))  
Add a note hereFigure 10-35: Web Service Proxy policy showing a WSRR subscription.

Add a note hereNow that we have a service that is subscribing to a WSRR repository for a WSDL, any time the WSDL changes in the repository, the changes will be reflected in the service at the next scheduled polling interval for the WSRR Subscription. This can be powerful when subscribing to a common service, as you will not need to be notified of the changes. This is only one of the many benefits of using WSRR for document management and governance. For more information on the WebSphere Registry and Repository product, please visit the WebSphere products section of the IBM Web site.

### Add a note hereWSRR Concepts

Add a note hereWe have seen how to subscribe to a single WSDL within WSRR in the previous example. It was the BookService.wsdl that was retrieved from the repository. Our MusicService.wsdl file was still being stored locally. We would most likely want to add both WSDLs to the repository where we can manage both of them together. It is possible to create a second subscription to retrieve this WSDL, but there is a more convenient and efficient way to do this. This is by utilizing a WSRR concept.

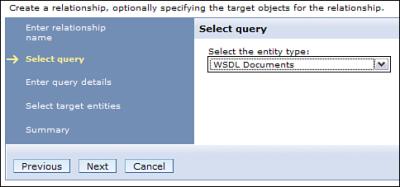
Add a note hereA WSRR concept is an association of related files within WSRR. This association is configured in WSRR and can be subscribed to instead of subscribing to each individual resource. Creating a concept is simple. From within the WSRR administration console, navigate to Service Metadata→Concepts to display all the configured concepts in the repository. There is also a button labeled New that allows you to create a new concept. Much like adding a WSDL file, the configuration screen asks for a name, a description, a namespace, and a version as shown in Figure 10-36.

[](javascript:PopImage('IMG_250','http://images.books24x7.com/bookimages/id_30903/10fig36_alt.jpg','938','430'))  
Add a note hereFigure 10-36: Creating a new concept in WSRR.

Add a note hereAfter the concept is created, the relationships can be added to it; that is, the files that you will want to reference as one entity. From the list of concepts, selecting the concept just created and then clicking the Add Relationship will present the first of five screens for creating these relationships. This simply asks for a relationship name. Figure 10-37 shows that we named our relationship Book\_Music\_WSDLs.

[](javascript:PopImage('IMG_251','http://images.books24x7.com/bookimages/id_30903/10fig37.jpg','493','218'))  
Add a note hereFigure 10-37: Specify a relationship name.

Add a note hereThe next screen is the Select query where an entity type is selected. Because many different types of entities can be stored in WSRR, this list is quite long. For our example, we are referencing WSDL Documents, so we choose that option, as shown in Figure 10-38. You may find it interesting to browse through this list to get an idea of what can be stored and referenced in a relationship.

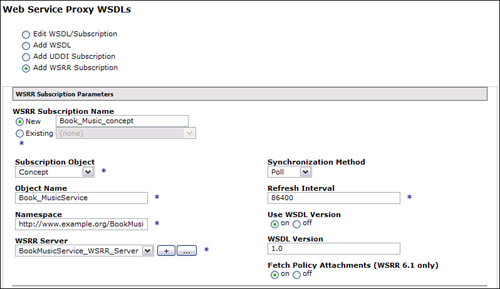
[](javascript:PopImage('IMG_252','http://images.books24x7.com/bookimages/id_30903/10fig38.jpg','484','227'))  
Add a note hereFigure 10-38: Selecting an entity type.

Add a note hereNext the query details can be entered. This is where a query can be created to narrow down, or filter, the list of documents to be selected for the relationship. If there were several hundred or even thousands of documents, this query would come in handy at this point. In our example, we have uploaded only two WSDL documents. This is assuming that the second WSDL has been uploaded. Because we have only two WSDL documents in our repository, we can enter a \* in the Name for the query. This returns the list of all WSDL documents in the repository, as shown in Figure 10-39. Here, you can see the two WSDL documents that we are interested in.

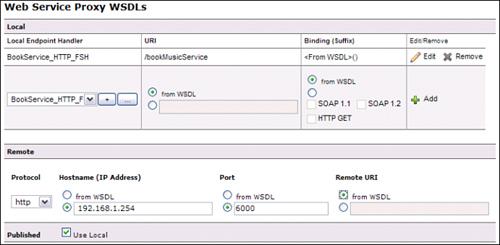
[](javascript:PopImage('IMG_253','http://images.books24x7.com/bookimages/id_30903/10fig39_alt.jpg','922','266'))  
Add a note hereFigure 10-39: WSDL Documents in the repository.

Add a note hereSelecting the two WSDLs displayed and clicking Next completes the configuration of our concept. We now have a concept in the WSRR repository that contains both the BookService.wsdl and the MusicService.wsdl documents.

Add a note hereBack in our DataPower service, we can now create a subscription to this new concept. This is identical to creating a WSRR subscription that retrieves a single WSDL; however, instead of selecting WSDL as the Subscription Object, we would select Concept. Figure 10-40 shows our WSRR subscription for the newly created concept.

[](javascript:PopImage('IMG_254','http://images.books24x7.com/bookimages/id_30903/10fig40_alt.jpg','798','462'))  
Add a note hereFigure 10-40: Creating a WSRR Subscription for a concept.

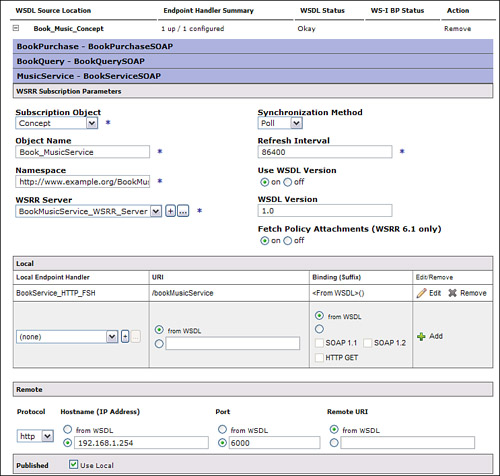
Add a note hereAfter creating this WSRR subscription, an FSH must be configured for it. In this case, we are working with more than one WSDL, so we might want to create a new, common URI to add to our Local configuration, as shown in Figure 10-41. Also notice that we chose to configure one Remote endpoint for this configuration. We could have chosen to select the endpoint from the WSDL as well.

[](javascript:PopImage('IMG_255','http://images.books24x7.com/bookimages/id_30903/10fig41_alt.jpg','691','339'))  
Add a note hereFigure 10-41: Endpoint configuration for the WSRR concept.

Add a note hereWe have now created one WSRR subscription for our service that subscribes to a WSRR concept that contains two WSDLs. We no longer need to maintain two WSDLs and endpoint configurations within the WSP as the WSDLs are retrieved from and maintained within the repository. If we wanted to confirm that we are indeed retrieving two WSDL documents from this concept, we can again view the WSDL subscription status. You can see in Figure 10-42 that our new subscription contains two WSDLs as expected.

[Image from book](javascript:PopImage('IMG_256','http://images.books24x7.com/bookimages/id_30903/10fig42_alt.jpg','831','65'))  
Add a note hereFigure 10-42: WSRR subscription status for a concept.

Add a note hereGoing back to our WSP, we can now view the new WSRR subscription in our service within the WSDL tab. As shown in Figure 10-43, the subscription shows the three bindings that are identified within the two WSDL documents as well as our endpoint configuration.

[](javascript:PopImage('IMG_257','http://images.books24x7.com/bookimages/id_30903/10fig43_alt.jpg','743','707'))  
Add a note hereFigure 10-43: Refreshed Book\_Music\_Concept WSRR subscription.

Add a note hereThroughout the examples in this chapter, we have been combining two different WSDLs. We mentioned that DataPower can publish these WSDLs with the new virtualized endpoints. When multiple WSDLs are added to a service and the WSDLs are published, they are done so as one combined WSDL.

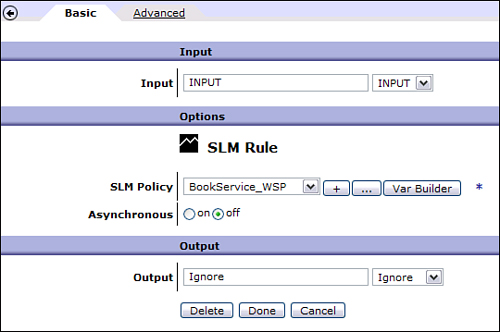
Add a note hereTo view how a WSDL will be published, it is possible to retrieve the WSDL through a browser using the endpoint of the service (the virtualized DataPower endpoint) with a ?WSDL appended to it. This will then return the WSDL as DataPower would publish it. Because this is performing an HTTP GET when this request is made, the GET method must be enabled within the FSH object for this request to be accepted. After this is enabled in our FSH object, we can submit the request. For our BookService\_WSP, we would enter the URL [http://192.168.1.254:6880/bookMusicService?WSDL,](http://192.168.1.254:6880/bookMusicService?WSDL) which would return one WSDL that contains the combined WSDLs within our service.

## Service Level Monitoring (SLM)

Add a note hereWe have now configured a WSP, including processing policies and WSDL management. After this service is up and running, you may be interested in monitoring and controlling the traffic to the service. This is made possible by the SLM action that is added to your service by default.

Add a note hereAs we showed when we created our first WSP service back in [Figure 10-7](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=1623#1623), a request and a response rule are automatically created for you at the Proxy level. The response rule is just an empty rule with a Match Rule and a Results action, but if you look at the request rule; the request rule contains an SLM action that looks like a simulated line graph and will enable the monitoring of the service.

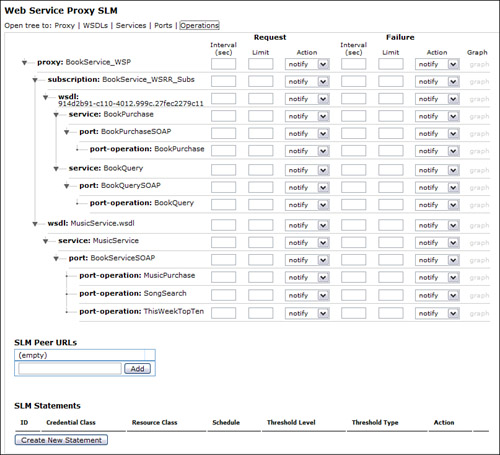
Add a note hereThe nice thing about the default SLM action is that there is really no configuration required for it. The only configuration required for the SLM action is an SLM Policy object. This object is created by default and added to the default SLM action upon WSP creation; it can be configured via the SLM tab in the policy configuration. We will see how this is configured shortly. Figure 10-44 shows the default SLM action configuration for our BookService\_WSP service.

[](javascript:PopImage('IMG_258','http://images.books24x7.com/bookimages/id_30903/10fig44_alt.jpg','540','359'))  
Add a note hereFigure 10-44: Default SLM action configuration.

Add a note hereOne very unique feature of this action that was created by default is that it is propagated to all other request rules within the policy behind the scenes. You will not see an SLM action in the request rules that you might configure at the different levels of the hierarchy, but it is always invoked as long as it is not removed from the proxy level. Because of this, you do not have to configure an SLM action for any other rules to have the SLM policy triggered.

Add a note hereThe SLM action that was added to the service by default allows for the traffic to the service to be monitored and controlled. Even though the name of the action is Service Level Monitoring, it allows for more than just monitoring the traffic. You can set a limit on how many transactions should be allowed to hit a certain operation, port, WSDL, or service in a specified amount of time. After this maximum threshold is reached, you can then decide if the service should reject subsequent traffic (throttle), queue the traffic for later submission (shape), or generate a log message (notify). When the shape option is chosen, there is a maximum of 2,500 transactions that will be queued. After that limit is reached, all subsequent transactions are dropped. Let’s take a look at how this all can be configured.

Add a note hereWithin the Web Service Proxy Configuration screen, there is a SLM tab located to the right of the WSDL tab. This tab contains all the SLM information for the service. Figure 10-45 shows the SLM tab for a WSP.

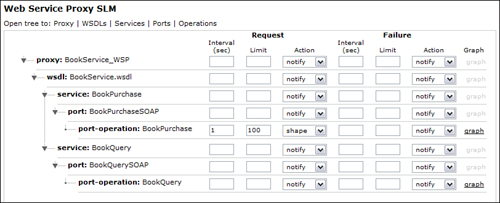
[](javascript:PopImage('IMG_259','http://images.books24x7.com/bookimages/id_30903/10fig45_alt.jpg','807','734'))  
Add a note hereFigure 10-45: SLM tab.

Add a note hereMuch like the Policy tab, this screen contains a hierarchy of all the WSDLs within the service down to the operations. Here it is possible to specify the transaction limits and intervals for requests to the service at a granular level. The interval indicates the number of seconds that the transactions will be counted, and the Limit indicates the maximum number of transactions for the number of seconds specified in the Interval field. So if you wanted to limit the number of transactions for a given operation to 5,000 transactions in a three-second span, you would specify 5,000 as the limit and 3 as the interval.

Add a note hereThe action can then be set to indicate whether the traffic should be shaped or throttled, or to write a message to the log. Each of these options can be very useful for serving different purposes. The notify action, which will write a message to the log might be used for keeping track of peaks in traffic for future capacity planning. The shape action can be used to protect your backend server from bursts of traffic over a longer period of time. And the throttle action might be used to enforce a maximum number of transactions allowed in a given period of time from a specific consumer. In addition to being able to monitor and control the request traffic to the service, you can also do the same for transactions that fail.

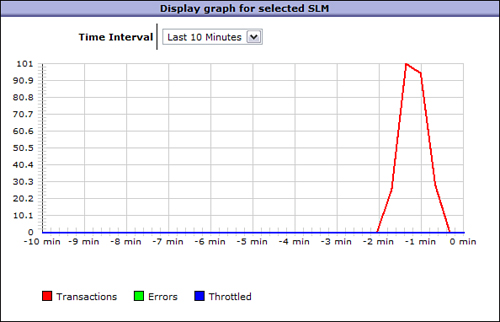
Add a note hereTo see how this all works, let’s suppose we performed a load test on our Book service and determined that the BookPurchase operation is resource-intensive, and it was determined that the saturation or break point of the operation was reached when it received 100 transactions per second. With this information, we can decide how we will configure our Service Level Monitor. Because we know that it is only the BookPurchase operation that causes the Web service to buckle, we will set our limit for this operation. We must now decide what should happen to the transactions after this limit is reached. Because we know that the holiday season usually brings spikes of traffic to our service, it might be possible to reach this level. Being that this is the purchase operation, we certainly would not want to lose any sales by rejecting transactions. This is where we could set the action to shape. Now when the transaction limit is reached, the subsequent requests will be queued and finally released to the backend at a rate that will not exceed our defined limit. This allows for a spike in traffic to be handled that would otherwise bring down the backend service or server. This seems like a much better alternative.

Add a note hereFigure 10-46 shows how the SLM within our service would be configured in this scenario. As you can see, we set the interval to 1 second and the limit to 100 with an action of shape. This indicates that when the requests to the BookPurchase operation reaches 100 transactions within a one second span, the subsequent requests will begin to be queued for later submission until the spike subsides.

[](javascript:PopImage('IMG_260','http://images.books24x7.com/bookimages/id_30903/10fig46_alt.jpg','812','330'))  
Add a note hereFigure 10-46: Setting a SLM for the BookPurchase operation.

Add a note hereNow that we have set our SLM Policy to shape all requests to the BookPurchase over 100 per second, let’s see what happens when this limit is reached. To do this we must send multiple concurrent requests to this operation. This can be accomplished using a load generating tool such as Apachebench. Apachebench is a free tool that allows you send multiple concurrent HTTP requests to a single destination.

Add a note hereAfter using this tool to submit a total of 5,000 requests to our BookQuery operation, we can take a look at how our SLM affected the requests. Clicking the graph link next to the Book Purchase operation in the SLM tab displays a graph representing the transactions for this operation. Three different types of graphs are available here. The transactions can be viewed by rate, latency, or count. The default graph shown is latency. Figure 10-47 shows this graph after we submitted the 5,000 requests from our Apachebench tool. Notice that all 5,000 requests were sent, but the number of requests within a one-second interval was capped at 100. This was because we set our SLM Policy for this operation to shape the traffic at 100 requests in a one-second time span.

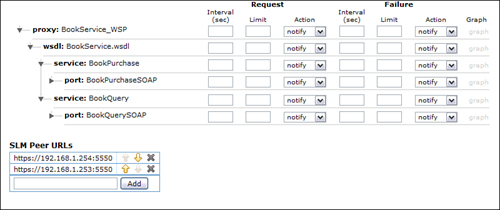
[](javascript:PopImage('IMG_261','http://images.books24x7.com/bookimages/id_30903/10fig47_alt.jpg','594','382'))  
Add a note hereFigure 10-47: Latency graph for BookPurchase transactions.

Add a note hereOur previous SLM example demonstrates that the traffic to the backend Web service can be shaped to accommodate any peaks in traffic without bringing down the backend service or infrastructure. This seems to work fine, and it would if there was only one DataPower device forwarding requests to the backend service. But what if there were multiple devices sitting behind a load balancer, all servicing requests for the same Web service? In our example we knew that we could not send more than 100 transactions to our service per second. If there were multiple devices sending requests to this backend service, then each device would have to know how many requests were being sent by the others in order to start shaping the traffic.

Add a note hereThis problem is solved by configuring an SLM Peer Group. An SLM Peer Group allows each SLM policy on these different devices to communicate with each other, providing the number of transactions sent to the backend service. This allows for the aggregation of these transactions, which can then be used to enforce the limit and threshold for the SLM policy. The SLM Peer group can be configured from within the SLM tab of the WSP. Each device must have identical SLM policies with the same SLM Peer group configured. Each peer group on each device must list all the device members, including itself.

Add a note hereA SLM Peer Group can be configured within the SLM tab for a WSP service; however, other service types must first create a Peer Group Object to facilitate this. This object can be created by navigating to Objects→Peer Group from the left navigation menu. Because we are working with WSP services here, we will show how this is accomplished within the SLM tab of the WSP configuration.

Add a note hereFigure 10-48 shows our BookService\_WSP SLM tab now with two URLs added to the SLM Peer URLs. The IP address and port specified must be the interface and port that the XML Management Interface is configured to listen on for the DataPower device being added. The default HTTPS port is 5550 and can be changed in the XML Management Interface settings. The XML Management Interface must also be enabled with the SLM Endpoint option selected. One of these URLs in this list must be the local DataPower IP:Port that this service is configured on. All members of the peer group must have identical SLM policies and SLM Peer URLs.

[](javascript:PopImage('IMG_262','http://images.books24x7.com/bookimages/id_30903/10fig48_alt.jpg','800','336'))  
Add a note hereFigure 10-48: SLM Peer group configuration for the BookService\_WSP.

Add a note hereWith the SLM Peer URLs setup, as shown in Figure 10-48, we can load balance traffic across two DataPower devices for our BookService\_WSP service. Now the SLM policy configured for the BookPurchase operation will take into account the requests being sent to the backend service by both devices.

### Add a note hereCustom Service Level Monitors

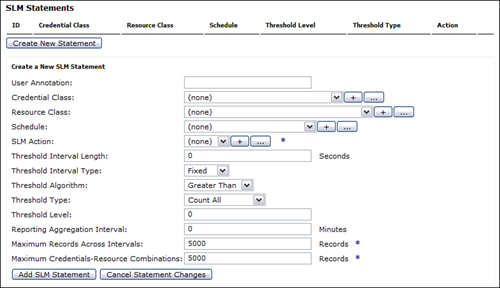
Add a note hereWe have demonstrated the flexibility and level of granularity of the SLM configuration within a WSP and, for the most part, this should suffice for your monitoring needs. There may be times, however, when you actually would like to configure an SLM Policy at a more granular level than what is offered here. This could be anything: a specific time of day, a specific user or group, or a specific header within the message itself. There are many possible scenarios that you might come across which is why DataPower offers the option of creating a custom SLM statement.

Add a note hereAn SLM statement is a user-defined set of conditions and actions that specify when an SLM Policy will be executed, under what conditions, the action to take, and the thresholds to set. The options within this configuration allow for customization at every level of the policy. The possibilities are endless, so we will walk you through the configuration screen and a sample scenario to give you a feel for what can be accomplished. After that you will be equipped to configure your own custom rules.

Add a note hereLocated at the bottom of the SLM tab of the WSP configuration is a section labeled SLM Statements. This is where you will define any custom SLM statements that are to be applied to transactions flowing through the service. Clicking the Create New Statement button opens the configuration screen for the new SLM Statement. The first thing you might notice is that there are many more options and configuration parameters than just the Interval, Limit, and Action that are configured in the standard SLM configuration above it. This should give you your first indication of how specific you can make this statement.

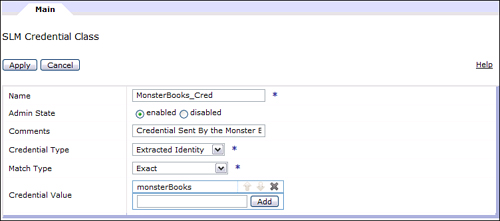
Add a note hereTo demonstrate the configuration options and how to configure this statement, let’s take another example scenario for our BookService\_WSP service. Suppose there was an external company that we know utilizes our service for purchasing books. This company purchases a significant number of books from us; however, they do not submit their purchases as the requests come in to them in real time. They store all their purchases for the week and kick off a batch process at the end of the week to submit them through the BookPurchase operation within our Web service. Our concern in that when this batch process kicks off it will impact the performance of the real time requests being made to our service. This would increase the response time to our other customers. Let’s take a look at how this can be solved by adding a SLM Statement to our SLM Policy.

Add a note hereFigure 10-49 shows the SLM Statement configuration screen presented after clicking the Create New Statement button. The first field here is the User Annotation, which is simply a place to enter a comment for this statement. The next field is the Credential class field. This refers to a credential class object that can represent a specific user credential for which this SLM Policy is enforced. An existing Credential Class can be selected or a new one can be created.

[](javascript:PopImage('IMG_263','http://images.books24x7.com/bookimages/id_30903/10fig49_alt.jpg','785','452'))  
Add a note hereFigure 10-49: SLM Statement configuration screen.

Add a note hereFigure 10-50 shows the configuration screen for the SLM Credential Class. As usual, this object requires a name, and optional comments may be entered. The type of credential to be matched is specified in the Credential Type field. The options here are

* Add a note here**Mapped Credential—** This credential is the result of the Mapped Credential phase in a AAA policy.
* Add a note here**Extracted Identity—** This credential is the result of the Extract Identity phase in a AAA policy.
* Add a note here**Client IP—** The IP of the client making the request.
* Add a note here**Custom Stylesheet—** Specify a custom XSLT stylesheet to extract the credential.
* Add a note here**IP From Header—** The IP address specified in the HTTP header.
* Add a note here**MQ Application—** The MQ application specified in the MQ message.
* Add a note here**Request Header—** Uses the value of a specified request header.

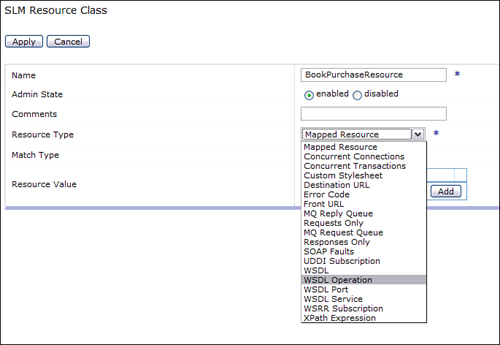
[](javascript:PopImage('IMG_264','http://images.books24x7.com/bookimages/id_30903/10fig50_alt.jpg','752','332'))  
Add a note hereFigure 10-50: SLM Credential Class configuration.

Add a note hereAs you can see, there are quite a few options to choose from when you need to execute an SLM policy based on a credential. In our example we know that the BookPurchase operation contains a AAA policy that will extract the identity of the client making the requests. Because the company making the request will always send the same credentials, we can choose the Extracted Identity for the Credentials Type field.

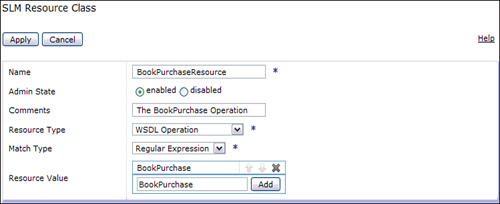
Add a note hereThe next field in this configuration is the Match Type. Here we can specify how we want to match on the credential. Here we can specify that an exact match is required, a regular expression to match with, or to allow DataPower to extract and store each unique credential for the specified type. In our example we know the credential that will be passed, so we will choose Exact for this field. We can now specify the value of the credential that is to be matched. Let’s suppose the company that submits this batch process passes a credential that is authenticated within our AAA policy with a username of monsterBooks. Figure 10-50 shows the configuration that we created to match on the Extracted Identity on a credential with the name of monsterBooks. Now when this batch process is submitted, each request will be identified by our SLM Credential Class.

Add a note hereEach of these fields is filtering the set of messages to which this SLM statement will actually apply. By adding a specific credential, we have assured that this statement will be executed *only* on calls submitted by monsterBooks. If we want it to apply to other customers, we would also have to enter their exact credentials. To define unique parameters for each user’s requests, you would need to create a new SLM statement for each customer.

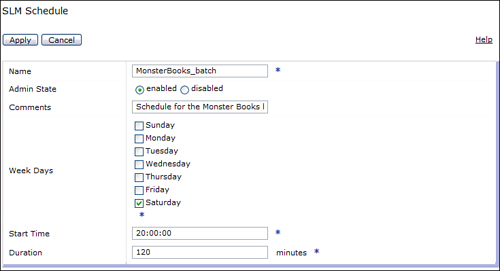
Add a note hereReturning to our SLM Statement configuration, the next field we can configure is the Resource Class. This references a Resource Class object that specifies what type of request this SLM statement will apply to. Figure 10-51 shows this object’s configuration screen with all the available Resource types. Notice that this is a very exhaustive list to choose from. Because we are concerned with our BookPurchase operation, we chose WSDL operation from this drop-down.

[](javascript:PopImage('IMG_265','http://images.books24x7.com/bookimages/id_30903/10fig51_alt.jpg','685','472'))  
Add a note hereFigure 10-51: Selecting a resource type.

Add a note hereOnce the Resource Type is specified, Match Type must be specified just as we did in the SLM Credential Class object. The actual operation name that is recognized by DataPower within each request is a fully namespace qualified name that includes the port as well as the operation. Because of this, we can simple specify that this is a regular expression match and just provide the operation name as we know it. Remember that you must click the Add button to add the value entered for the Resource Value in order for it to be added to the object’s configuration. Figure 10-52 shows the completed Resource Class configuration for our example.

[](javascript:PopImage('IMG_266','http://images.books24x7.com/bookimages/id_30903/10fig52_alt.jpg','753','307'))  
Add a note hereFigure 10-52: Completed SLM Resource Class object.

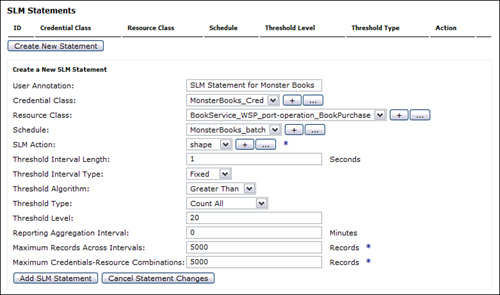
Add a note hereNow that we specified in our SLM statement that we are looking for a request carrying the monsterBooks credential for the BookPurchase operation, we have narrowed down this statement to the correct requestor and request type. We can narrow this a bit more if we would like to. We know that this company triggers this batch process once a week on a Saturday night at 8:00 PM. In an effort to not restrict any real time requests from this company during the week, we can specify that we only want this statement to apply to requests on this day and time. This can be specified in the SLM statement within the Schedule field. This field references a Schedule object that can be configured. Figure 10-53 shows how we would configure this object to restrict this SLM statement to requests made on Saturdays between 8:00 PM and 10:00 PM.

[](javascript:PopImage('IMG_267','http://images.books24x7.com/bookimages/id_30903/10fig53_alt.jpg','735','399'))  
Add a note hereFigure 10-53: SLM Schedule object configuration.

Add a note hereNow that we have scoped this SLM statement down to the requestor, the operation, and the day and time, we can specify the actual SLM thresholds and actions. In the SLM statement, we can specify the same type of actions as we did in our first SLM policy. That is, what to do when a threshold is exceeded. Because we do not want to reject these batch transactions, we can simply shape this traffic as we did in our first example. Because this is a batch process, we are not as concerned with the response times so we queue these transactions with this option.

Add a note hereWe then specify the Threshold Interval Length. This again is the length in seconds that the transactions will be measured. So, if we want to restrict this process to 20 transactions per second, we would specify a 1 here to represent the 1 second. The threshold level would then be set at 20.

Add a note hereFigure 10-54 shows the completed SLM statement. Clicking the Add SLM Statement button adds this statement to the policy. We now have an SLM statement that shapes the traffic to the BookPurchase operation on Saturday nights between the hours of 8:00 PM and 10:00 PM that were submitted with a credential of monsterBooks.

[](javascript:PopImage('IMG_268','http://images.books24x7.com/bookimages/id_30903/10fig54_alt.jpg','781','461'))  
Add a note hereFigure 10-54: Completed SLM statement.

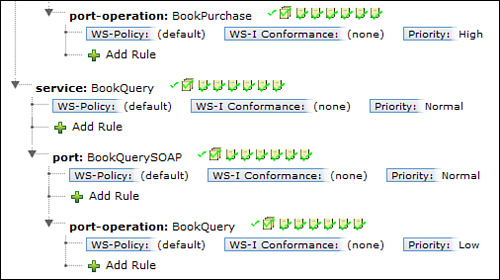
Add a note hereAll fields within the SLM statement are optional. By not providing a value for any given field implies that all possible values for that field are accepted for this SLM statement. Supplying values, however, provides additional filtering, lowering the number of potential messages to which this statement could apply.

Add a note hereAs you can see, the SLM capabilities within DataPower are extremely flexible when implementing your own SLM statements. We showed only a sample of what the possibilities are for SLM. It is advised that you consult your WebGUI guide for additional options and features within the SLM policy.

### Add a note hereService Priority

Add a note hereIt is nice to think that all things are created equal, but when we are dealing with transactions flowing through our IT systems, this is not likely the case. It is likely that different DataPower services, WSDLs, or even operations within a service will be more critical than others. In this case you might want to prioritize these so that when there is a heavy demand on the DataPower service, the more critical transactions are serviced first. For example, in our BookService\_WSP, we have a BookQuery operation and a BookPurchase operation. It is likely that you would want the transactions for purchasing books to have a higher priority to access the DataPower resources. That is, the requests for it will be handled first. This is a very simple configuration option within the service Policy that we can set.

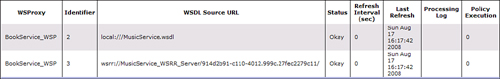
Add a note hereBecause this priority is set within the WSP Policy tab, we have the flexibility of setting it for any level of the Policy tree provided. This means that different priorities can be set at an operation, port, WSDL, or service level. At each of these levels within the Policy, there is a button labeled Priority. Clicking this button presents three options to choose from—Low, Normal, and High. As you would have guessed, a setting of Low assigns the lowest priority and a High is the highest priority. Figure 10-55 shows how we set the BookQuery operation to a priority of Low and the BookPurchase operation to High.

[](javascript:PopImage('IMG_269','http://images.books24x7.com/bookimages/id_30903/10fig55_alt.jpg','506','283'))  
Add a note hereFigure 10-55: Priority set at the operation level.

Add a note hereThis is not only convenient for prioritizing operations within a WSP, it can also be configured at the service proxy level, which will provide prioritization of all your services on the same device. Keep in mind that there are only three possible priority values that can be assigned to a given service, so it is likely that some services will be assigned the same priority on a single device.

Once the Web Service Proxies have been configured, DataPower provides some very convenient status screens for viewing the status of the parts of the WSP configuration, such as WSDLs, operations, WSRR and UDDI subscriptions, and more. We will briefly take a look at some of these status screens and demonstrate how they can be very handy.

Add a note hereAll the status screens for the Web Service Proxies must be accessed from the left navigation menu in the WebGUI and will be under Status→Web Service. The first one that we will take a look at is the Web Services WSDLs screen. This screen, shown in Figure 10-56, shows the status of each WSDL that is configured within a WSP in the current domain. Here, a full status of each WSDL can be viewed along with the WSP that it is configured in, the location of the WSDL, the refresh interval, and when the last refresh occurred.

[](javascript:PopImage('IMG_270','http://images.books24x7.com/bookimages/id_30903/10fig56_alt.jpg','1025','161'))  
Add a note hereFigure 10-56: Web Services WSDLs screen.

Add a note hereAnother handy status screen is the Web Services Operations screen. Here you can view all the operations available within all the WSDLs contained within the WSPs in the current working domain. Figure 10-57 shows this screen. Here, in one single screen, you can see the operations, WSP handling the requests to it, the IP address and port of the FSH listening for requests, the URI, SOAP action and body, the backend address, and the front-side and back-side protocols.

[Image from book](javascript:PopImage('IMG_271','http://images.books24x7.com/bookimages/id_30903/10fig57_alt.jpg','1547','116'))  
Add a note hereFigure 10-57: Web Service Operations status screen.

Add a note hereThe last status screen we discuss here is the Web Services Operation Metrics. Shown in Figure 10-58, this screen displays information about the requests and responses that have hit a particular operation. It provides the endpoint, port, and operation along with information about the requests and responses to them. Here you can see the request and response totals, successes, failures, size, and response times. This information can be valuable when identifying the load on each operation and the response times.

[Image from book](javascript:PopImage('IMG_272','http://images.books24x7.com/bookimages/id_30903/10fig58_alt.jpg','1411','140'))  
Add a note hereFigure 10-58: Web Service Operation Metrics.

## Summary

Add a note hereAs the popularity of Web services and SOA continues to grow, so do the challenges around security, governance, integration, and monitoring. We demonstrated throughout this chapter that the DataPower Web Service Proxy provides the ease, flexibility, and power to solve many of these challenges. This service has the capability to secure your Web Services against potential threats, virtualize endpoints, and schema validate requests all by simply importing a WSDL. In addition to this, the service can be configured to provide all the request and response processing that is available within other service types. The ease of use combined with the power, flexibility, and benefits provided make the Web Service Proxy a must-have when exposing Web services