WebSphere. software

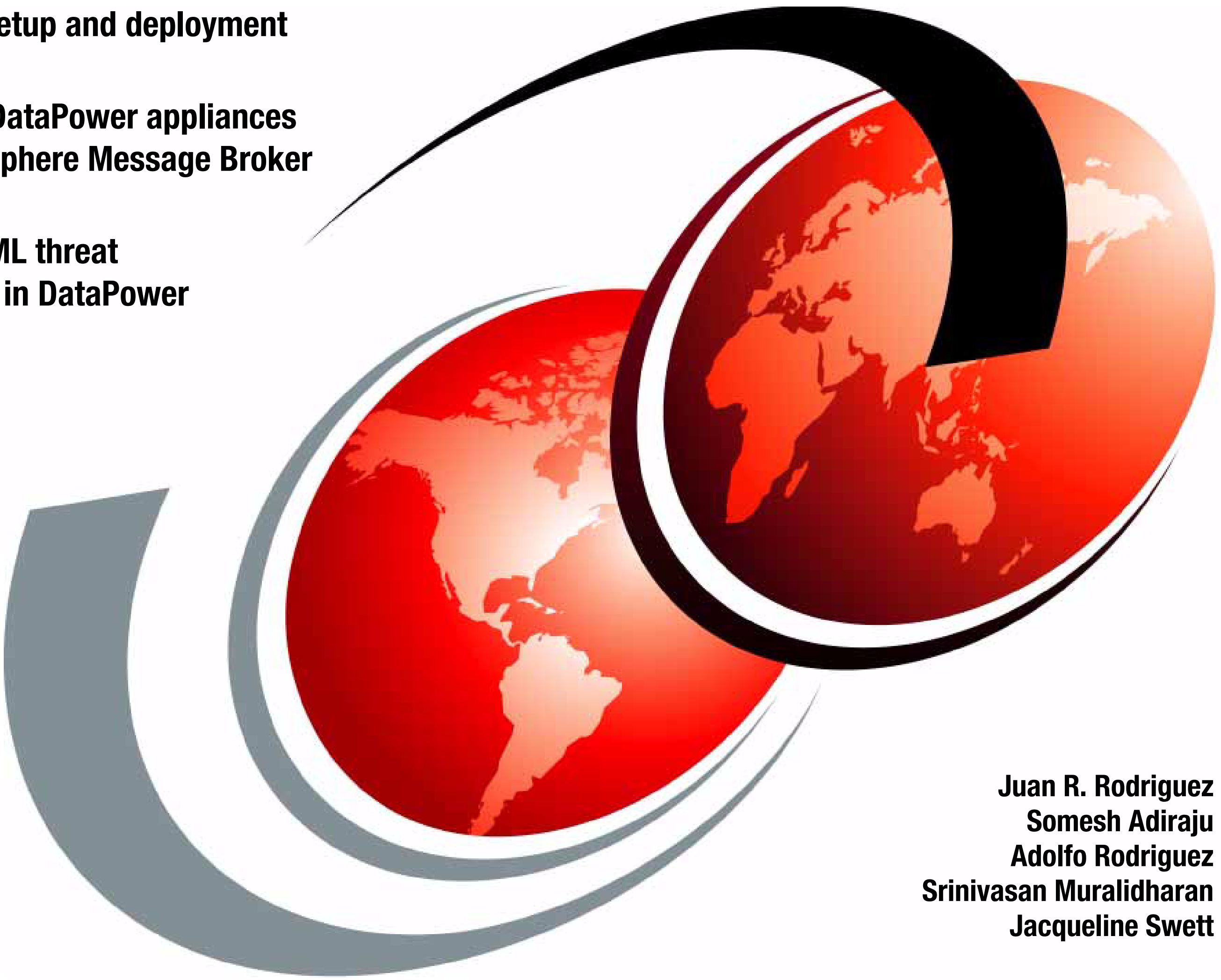
IBM

IBM WebSphere DataPower SOA Appliances
Part III: XML Security Guide

Secure and improve your XML and Web Services setup and deployment

Integrate DataPower appliances with WebSphere Message Broker

Provide XML threat protection in DataPower



Juan R. Rodriguez
Somesh Adiraju
Adolfo Rodriguez
Srinivasan Muralidharan
Jacqueline Swett

Redpaper

ibm.com/redbooks

International Technical Support Organization

IBM WebSphere DataPower SOA Appliances: Part III: XML Security Guide

April 2008

REDP-4365-00

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (April 2008)

This edition applies to Version 3, Release 6, Modification 0 of IBM WebSphere DataPower Integration Appliance.

Contents

Notices

Trademarks

Preface

IBM® WebSphere® DataPower® SOA Appliances represent an important element in the holistic approach of IBM to service-oriented architecture (SOA). IBM SOA appliances are purpose-built, easy-to-deploy network devices that simplify, help secure, and accelerate your XML and Web services deployments while extending your SOA infrastructure. These appliances offer an innovative, pragmatic approach to harness the power of SOA. By using them, you can simultaneously use the value of your existing application, security, and networking infrastructure investments.

This series of IBM Redpaper publications is written for architects and administrators who need to understand the implemented architecture in WebSphere DataPower appliances to successfully deploy it as a secure and efficient enterprise service bus (ESB) product. These papers give a broad understanding of the new architecture and traditional deployment scenarios. They cover details about the implementation to help you identify the circumstances under which you should deploy DataPower appliances. They also provide a sample implementation and architectural best practices for an SOA message-oriented architecture in an existing production ESB environment.

Part three of the series, this paper, describes how to use the DataPower appliance to secure incoming Web Services within an SOA environment, how to integrate your DataPower appliance with WebSphere Message Broker, and how to provide protection against security attacks by implementing the XML Denial of Service (XDoS) provided by DataPower appliances. The entire IBM WebSphere DataPower SOA Appliances series includes the following papers:

- *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started,* REDP-4327
- *IBM WebSphere DataPower SOA Appliances Part II: Authentication and Authorization,* REDP-4364
- *IBM WebSphere DataPower SOA Appliances Part III: XML Security Guide*, REDP-4365
- *IBM WebSphere DataPower SOA Appliances Part IV: Management and Governance,* REDP-4366

The team that wrote this paper

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Juan R. Rodriguez is a Consulting IT Professional and Project Leader at the IBM ITSO Center, Raleigh. He has an M.S. degree in Computer Science from Iowa State University. He writes extensively and teaches IBM classes worldwide on Web technologies and information security. Before joining the IBM ITSO, he worked at the IBM laboratory in Research Triangle Park, North Carolina, as a designer and developer of networking products.

Somesh Adiraju is currently working as an Integration Architect with Ultramatics Inc., in Florida. He has nine years of experience in Enterprise Application Integration space in areas of banking, finance, and telecommunications. His interests include the design, architecture, and development of enterprise scale applications specializing in the use of WebSphere MQ, Message Broker, and IBM Tivoli® Monitoring. Somesh holds a Bachelors degree in Technology from Andhra University, India.

Adolfo Rodriguez is a Software Architect within the IBM WebSphere Technology Institute (WSTI) and an Assistant Adjunct Professor of Computer Science at Duke University, where he teaches networking courses. At IBM, he leads a team of engineers focusing on emerging technologies in WebSphere products and, more recently, DataPower SOA appliances. Dr. Rodriguez's recent contributions include projects in the areas of Web Services enablement, XML processing, SOA management, and application-aware networking. His primary interests are networking and distributed systems, application middleware, overlays, and J2EE™ architecture. He has written 12 books and numerous research articles. Dr. Rodriguez holds four degrees from Duke University: BS in Computer Science, BA in Mathematics, MS in Computer Science, and a Ph.D. in Computer Science (Systems).

Srinivasan Muralidharan is an Advisory Engineer with 15 years of industrial experience and nine years with IBM. He is currently working on DataPower-related projects at the IBM WebSphere Technology Institute. He is widely experienced in SOA-related technologies in all tiers of the software development stack. He has studied SOA performance with DataPower appliances and has investigated integrating DataPower with other mid-tier and back-end traditional components, such as WebSphere Application Server, MQ, CICS®,and IMS™ in the SOA context of reusing existing systems and enterprise modernization.

Jacqueline Swett is a Senior WebSphere IT Specialist working for the Federal Software Sales Group. She has twenty-five years of experience working in the IT Industry, especially in the application integration and Middleware areas. After she received her degree in Computer Science in 1982, she worked for a major insurance company for fifteen years as a developer, database administrator, and system programmer. She joined a software development firm in 1997 to develop a system monitor for MQSeries®. Since joining IBM in 2000, she has worked as a solution architect for the Software Group, as a lead designer and developer for a major US Government modernization project, and as a lead WebSphere IT specialist for the Federal Government accounts since 2004. She is a member of the WebSphere Application Integration Technical Competencies Leadership team.

Thanks to the following people for their contributions to this project:

Joel Gauci
IBM France

Davin Holmes
IBM Australia

Christian Ramírez-Mora
GBM Corporation, Costa Rica

Joel Gauci
IBM Paris, France

Marcel Kinard, Robert Callaway, John Graham
IBM Research Triangle Park, North Carolina, USA

Peter Crocker, Ben Thompson, Dominic Storey
IBM United Kingdom

Andy Grohman
IBM Charlotte, North Carolina, USA

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks® publications in one of the following ways:

- Use the online **Contact us** review IBM Redbooks publications form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Web Services security improvements

In this chapter, we explain how you can use DataPower appliances to secure incoming Web Services within a services-oriented architecture (SOA) environment.

When exposing service-oriented systems to Internet access, you must consider various security issues. Most businesses are used to dealing with the potential for attack against Web-based sites, but the new and emerging challenge that we face is exposing the enterprise business logic in the form of Web Services. While this capability provides greater flexibility and integration with partners, it also exposes businesses to greater risks, because it necessitates opening the firewall through standard ports.

As a logical evolution, the core sample application that we described in *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started*, REDP-4327, will be fortified by securing incoming SOAP requests against some of these attacks.

1.1 Extensible Markup Language threat protection

As the backbone of SOA, the popularity of Web Services has invited a whole new class of attacks. At a high level in the context of Web Services, we define these as XML threats. That is, a client system sends an Extensible Markup Language (XML) request (maybe as a SOAP request) to your systems that in some way intends to do harm.

If a system can be accessed by outsiders (for example, via the Internet), an intruder can choose to send messages to your system in order to damage it or simply to consume resources. Depending on the system, it is even possible that these intruders are authorized to use your system, but are trying to exploit that authorization in some inappropriate way. Also, you must not discount the possibility of attacks (or other actions that can affect system response) from behind your own firewalls.

There are four broad classifications of XML threats:

- XML Denial of Service (XDoS): Slowing down or disabling a Web Service so that valid service requests are hampered or denied. We discuss XDoS in Chapter 3, “XML threat protection in DataPower” on page 97.
- Unauthorized Access: Gaining unauthorized access to a Web Service or its data
- Data Integrity/Confidentiality: Attacks that strike at the data integrity of Web Service responses, requests, or underlying databases
- System Compromise: Corrupting the Web Service itself or the servers that host it

These threats are not addressed by traditional firewalls; although, they offer protection at the transport levels and allow Web Services to effectively tunnel through this layer via standard HTTP(S) and expose the enterprise applications.

These threats are not addressed either when using traditional Web service gateways that do not look for most XML threats or limit the traffic.

Note: DataPower can act as an application level gateway to protect your enterprise applications against these attacks.

Do not understate the seriousness of these new threats. The following IBM developerWorks® article clearly defines the breadth and seriousness of attacks that are possible to any service exposed using XML:

http://www-128.ibm.com/developerworks/websphere/techjournal/0603_col_hines/0603_col_hines.html

The article concludes with the following comments:

“To truly harden a system using Web Services, you need to perform several important security steps (recommended by leading consultants), including:

- Inspect messages for well-formedness.
- Validate schema.
- Verify digital signatures.
- Sign messages.
- Implement service virtualization to mask internal resources via XML transformation and routing.
- Encrypt data at the field level.”

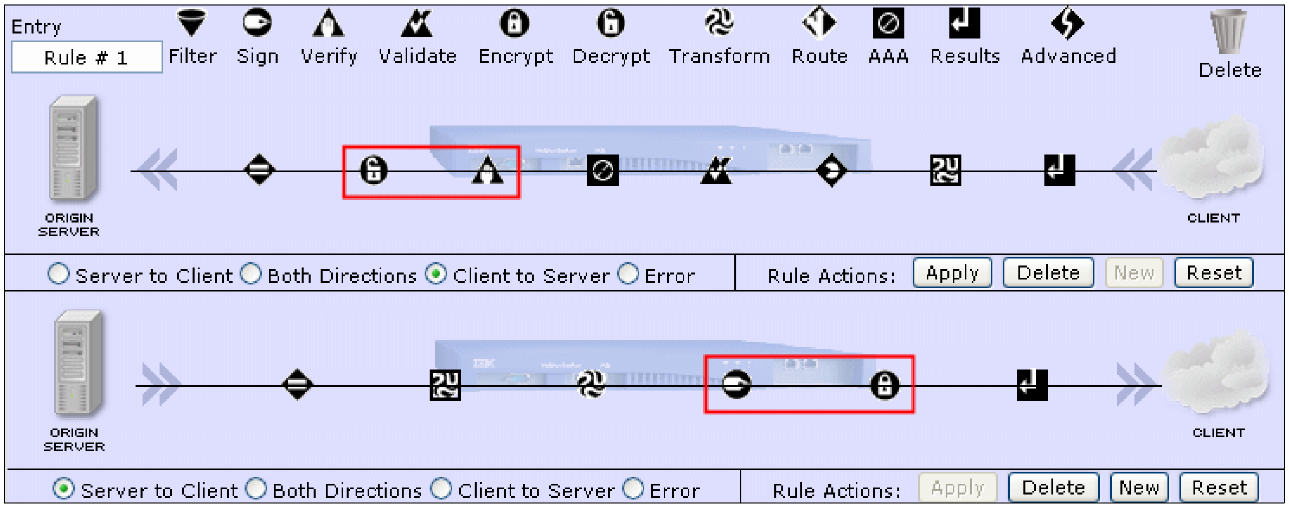
For systems hosting Web Services, particularly public Internet-facing ones, you must seriously consider the case for hardened gateway devices acting as XML firewalls to protect your systems from XML threats.

In this chapter, we illustrate how you can use the previously mentioned security options with the DataPower appliance.

1.2 Using Multi-Protocol Gateway to address security concerns

The graphic shown in Figure 1-1 illustrates the end-to-end scenario, but we only discuss the highlighted actions in this section.

Note: Well-formedness checking and validate schema are discussed in *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started*, REDP-4327. Authentication, Authorization, and Auditing (AAA) are discussed in *IBM WebSphere DataPower SOA Appliances Part II: Authentication and Authorization*, REDP-4364.



Server to Client Both Directions [✓ Client to Server] Error

Server to Client [✓ Server to Client] Both Directions Client to Server Error

Figure 1-1 Shows both Rules of ITSO_MPGW Multi-Protocol Gateway

In this section, we use a Multi-Protocol Gateway, which was created in *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started*, REDP-4327, as the core transformation engine. We enhance this Multi-Protocol Gateway to process WS-Sec messages:

- Request decryption
- Request verification
- Response encryption
- Response signing

In addition, we also show you how to enable transport level Secure Sockets Layer (SSL).

Note: Before performing the following steps, create a Multi-Protocol Gateway object and except for the Request-side HTTP port and configure it identically to the Multi-Protocol Gateway that is described in *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started*, REDP-4327.

1.2.1 Request processing

We will start with Request Rule #1, which accepts an encrypted message from the client, but it does not handle the messages sent by the server to the client. This is enforced by selecting the client-to-server radio button at the bottom of the rule. A Match action examines the inbound message and determines whether the rule needs to be run against the message. In this case, the match action accepts all input messages (as implemented by the asterisk character (*) criteria in the Match action).

The following subsections detail how the remainder of the security actions are implemented in this policy.

Decrypt action

A decrypt action performs full or field-level document decryption. Implementation of a decrypt action requires certain parameters that are supplied during Multi-Protocol Gateway configuration. Use the following procedure to add a decrypt action to a document processing rule:

1. Drag and drop the decrypt action icon before the “Extract using XPath” action in the Request processing rule.
2. Double-click the decrypt action icon to display the basic panel in Figure 1-2.

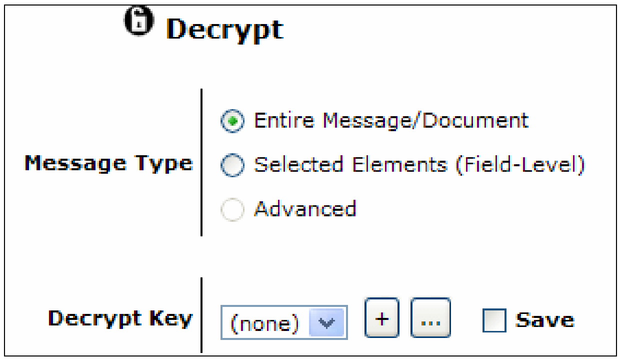
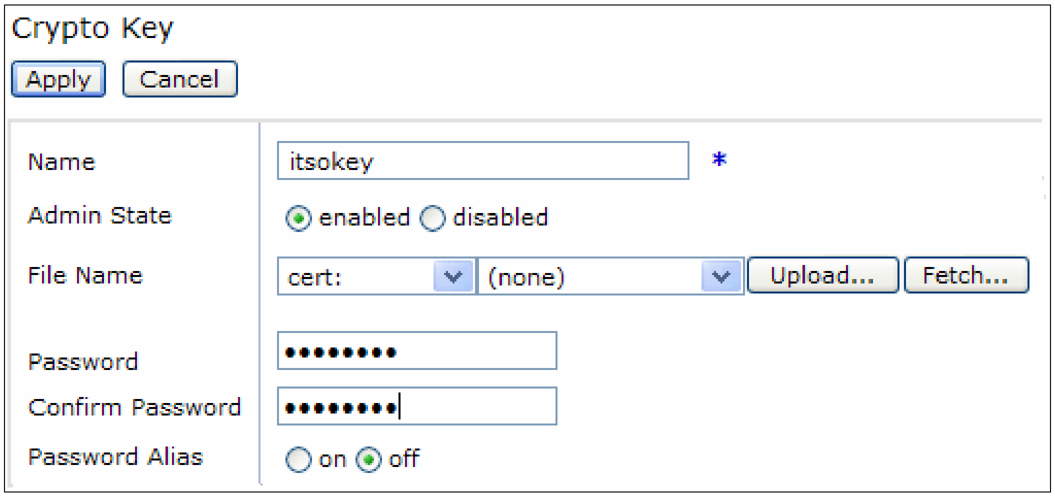


Figure 1-2 Decrypt action (basic) panel

3. In the sample input file, the whole message is encrypted. Therefore, choose the option **Message Type** → **Entire Message/Document**.
4. Use the Decrypt Key drop-down list to select a Cryptographic Key to use for decrypting the content. Click **+** to create a new key. This brings up the Crypto Key panel as shown in Figure 1-3 on page 5.


Crypto Key

Name: itsokey *

Admin State: [✓ enabled] disabled

File Name: cert: (none)

Password: ●●●●●●●●●

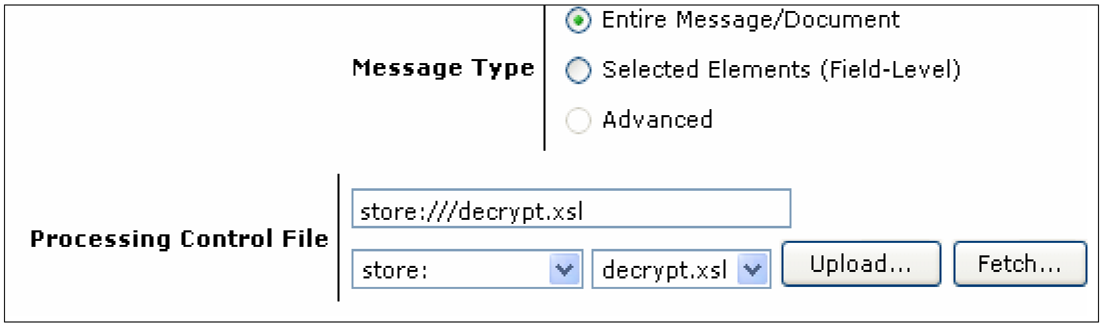
Confirm Password: ●●●●●●●●●

Password Alias: on [✓ off]

Figure 1-3 Crypto Key object creation

5. In this sample scenario, specify `itsokey` in the name field. You will be associating this object with the `itsodp-privkey.pem` key provided in the associated materials. Upload this file into the appliance by clicking **Upload**. To complete the configuration, specify a password, for example, `itsopass` for the password and confirm password fields.

   One interesting point to note here is by clicking Advanced on Figure 1-4, you see under processing control file `store:///decrypt.xsl` that DataPower has its own stylesheets for all actions. You can use this stylesheet as a default or create your own Extensible Stylesheet Language (XSL) stylesheet for custom processing as shown in Figure 1-4.


Message Type: [✓ Entire Message/Document] Selected Elements (Field-Level) Advanced

Processing Control File: store:///decrypt.xsl

store: decrypt.xsl

Figure 1-4 Decrypt Action (Advanced) panel

6. Click **Done** to complete the decrypt action.
7. Click **Apply** to add the decrypt action to the Document Processing Rule.

Verify action

The decrypted SOAP message contains the signer certificate, which can be verified to make sure that this is an acceptable message. Implementation of the verify action requires certain parameters that are supplied during the Multi-Protocol Gateway configuration, in this scenario, `ITSO-MPGW`.

Use the following procedure to add a verify action to a Document Processing Rule:

1. Drag and drop the Verify action icon after Decrypt action.
2. Double-click the verify action icon to display the verify action panel in Figure 1-5 on page 6.

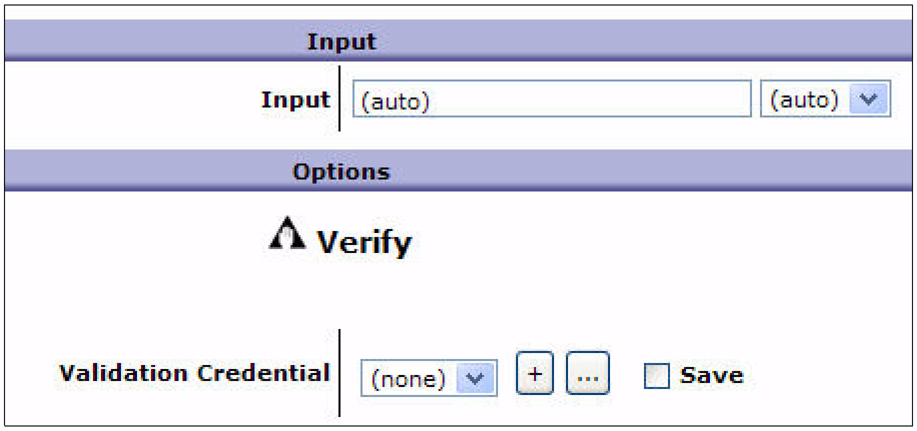


Figure 1-5 Verify action panel

3. You first configure the Validation Credential. Click **+** to create a Validation Credential object. This action will bring up the Crypto Validation Credentials panel as shown in Figure 1-6.

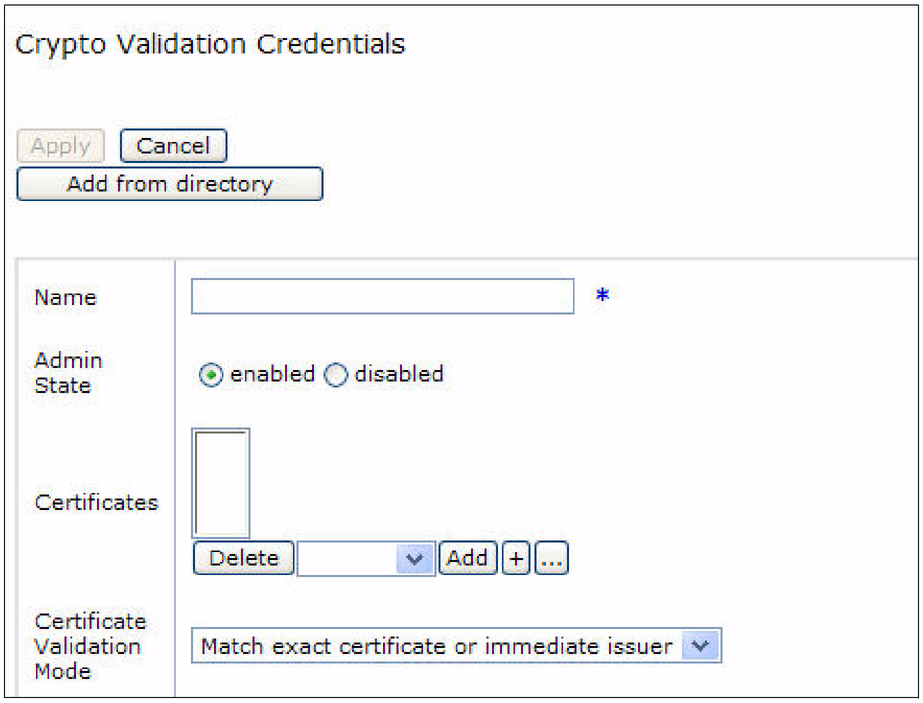
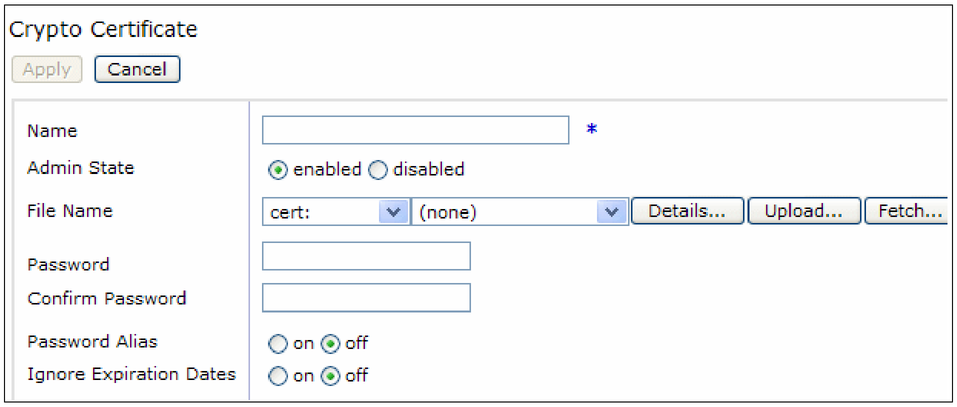

Crypto Validation Credentials
Apply
Cancel
Add from directory
Name
*
Admin State
enabled
disabled
Certificates
Delete
Add
+
...
Certificate Validation Mode
Match exact certificate or immediate issuer

Figure 1-6 Creating validation credentials

4. In this scenario, type `itsovalid` in the name field. Click **+** in the certificates section. This will open up the Crypto Certificate panel to configure the crypto certificate as shown in Figure 1-7 on page 7.

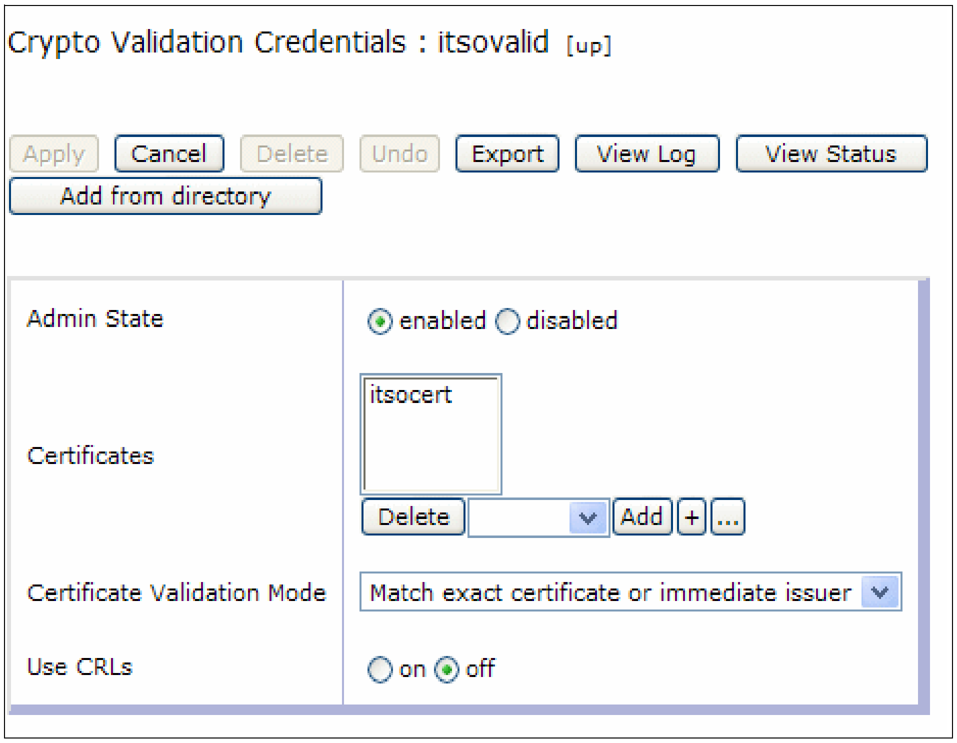


Crypto Certificate

Name	*
Admin State	[✓ enabled] disabled
File Name	cert: (none)
Password	
Confirm Password	
Password Alias	on [✓ off]
Ignore Expiration Dates	on [✓ off]

Figure 1-7 Configure crypto certificate

5. For example in this scenario, name the certificate object `itsocert`. Import the `itsodp-sscert.pem` certificate provided in the additional materials into the appliance by using **Upload**. Specify the password, for example, `itsopass`, in the password and confirm password fields.
6. Click **Apply**. This takes you back to the Crypto Validation Credentials panel shown in Figure 1-6 on page 6. To add the newly created Certificate object, click **Add**. You see the `itsocert` certificate listed in the Certificates list box as illustrated in Figure 1-8.

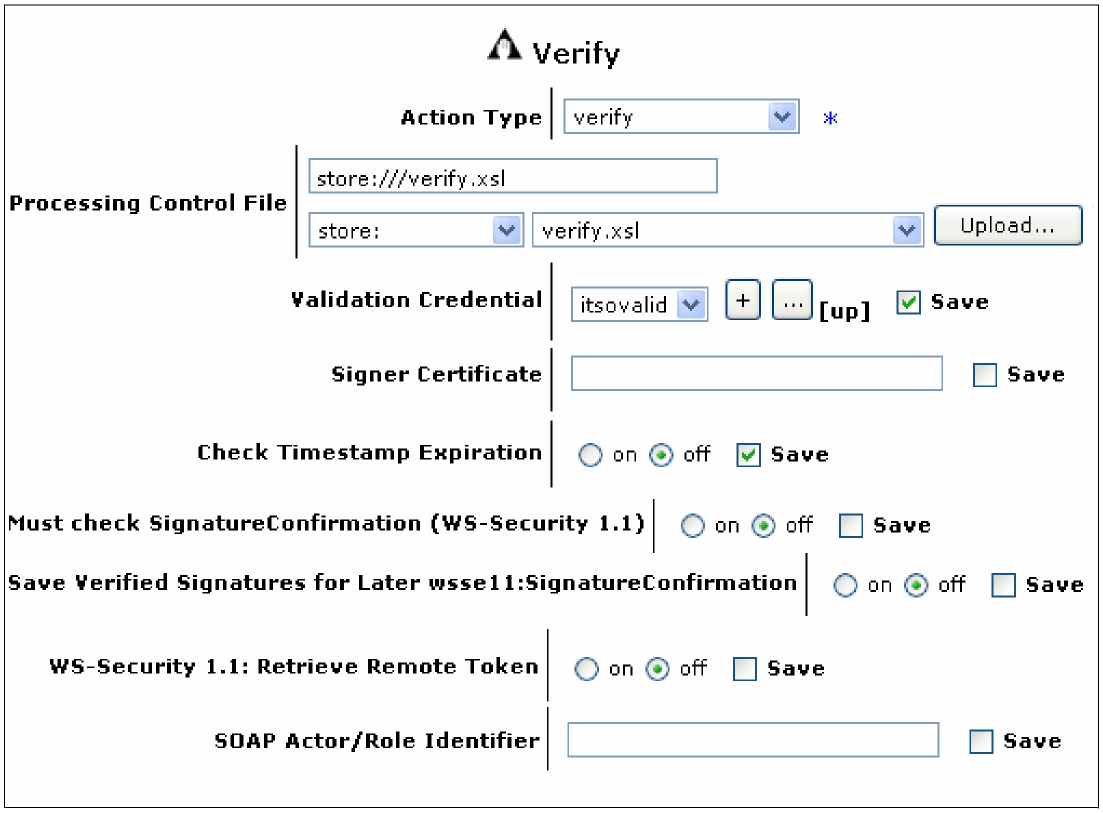


Crypto Validation Credentials : itsovalid [up]

Admin State	[✓ enabled] disabled
Certificates	itsocert
Certificate Validation Mode	Match exact certificate or immediate issuer
Use CRLs	on [✓ off]

Figure 1-8 After adding itsocert to the certificates

7. Clicking **Advanced** opens up the Verify action (advanced) panel that is shown in Figure 1-9 on page 8.



Verify

Action Type	verify	*
Processing Control File	store:///verify.xsl	
	store: verify.xsl	Upload...
Validation Credential	itsovalid + ... [up]	[✓ Save]
Signer Certificate		Save
Check Timestamp Expiration	on [✓ off]	[✓ Save]
Must check SignatureConfirmation (WS-Security 1.1)	on [✓ off]	Save
Save Verified Signatures for Later wsse11:SignatureConfirmation	on [✓ off]	Save
WS-Security 1.1: Retrieve Remote Token	on [✓ off]	Save
SOAP Actor/Role Identifier		Save

Figure 1-9 Verify action (advanced) panel

8. You can also check a WS-Security Timestamp for expiration. The verify action checks for a time stamp by default. To disable this action, change the value of the Check Timestamp Expiration property to off.
9. Optionally, use the Output dialog box to specify the destination context for the verified document.
10. Click **Done** to complete the verify action.
11. Click **Apply** to add the verify action to the Document Processing Rule.
12. The security processing aspect of the incoming WS-Sec message is complete. The decrypted message still contains WS-Sec tags. The core application processing is based on the payload only. You will need to strip out the WS-Sec sections from the message so that the resulting message will be identical to the input message. To strip out the WS-Sec sections from the message, drag and drop a **Transform** action after the Verify action of the previous step. Specify the `stripwssec.xsl` provided in the additional materials as the XSL executing the transform.
13. After the previous step, the message is now identical to the input message. To complete the request side processing, follow, for example, the Markup Language (ML) to COBOL transformation. For details, see *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started,* REDP-4327.

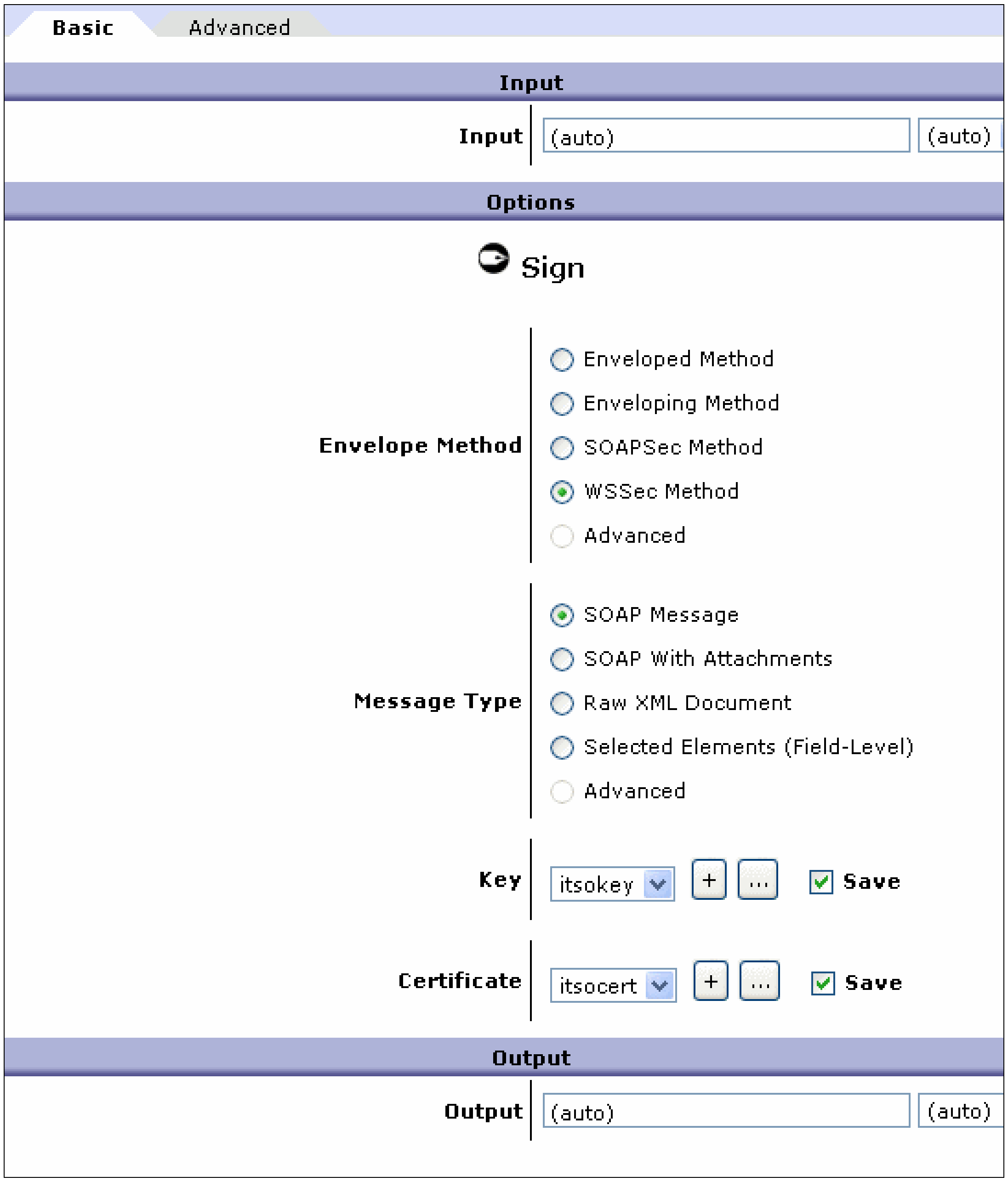
1.2.2 Response side processing

The core application transforms response side COBOL input to XML and converts it into SOAP. In this section, we will do WS-Sec processing to secure the response message.

Sign action

A *sign action* digitally signs documents. In our case, it signs the incoming SOAP message before encrypting the message and sending it back to the client:

1. Drag the Sign icon and drop it next to the action that converts the message to SOAP using the converttosoap.xsl.
2. Double-click the **Sign** icon to display the sign action panel as illustrated in Figure 1-10.



Input: (auto) (auto)

Envelope Method: Enveloped Method [ ] / Enveloping Method [ ] / SOAPSec Method [ ] / [✓ WSSec Method] / Advanced [ ]

Message Type: [✓ SOAP Message] / SOAP With Attachments [ ] / Raw XML Document [ ] / Selected Elements (Field-Level) [ ] / Advanced [ ]

Key: itsokey [✓ Save]

Certificate: itsocert [✓ Save]

Output: (auto) (auto)

Figure 1-10 Sign action panel

3. Use the Input dialog box or values-list (a list of contexts available within this Document Processing Rule) to specify the source context for the encrypt action. This is the output of the previous transform action.

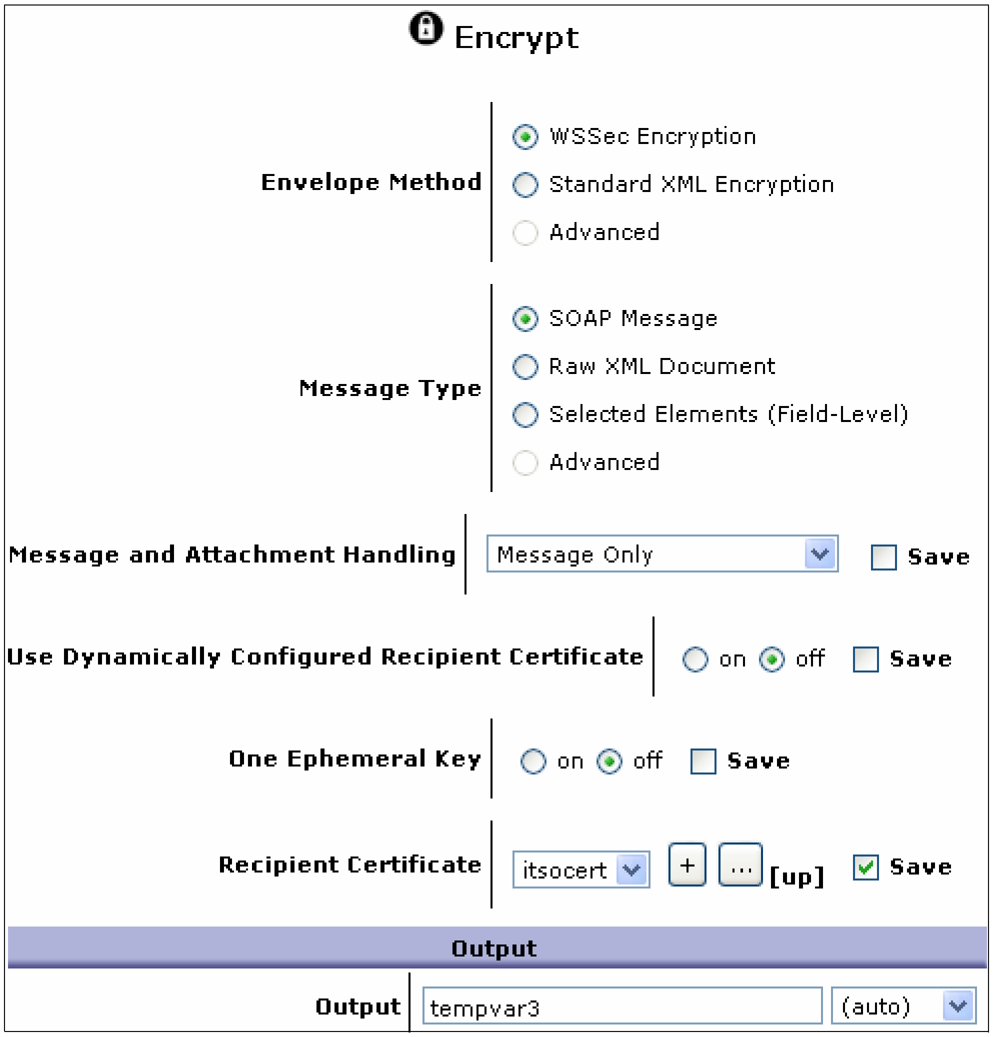
Note: When defining a sign action, you can use variables to specify input and output contexts.

4. Select the Envelope Method radio buttons to select the signature type.
5. Select the SOAP Message radio button to select the message type.
6. In this scenario, the same Crypto Key object created for the Decrypt action (see Figure 1-3 on page 5) will be used for signing the message. Select **itsokey** from the drop-down list in Key.
7. The same Crypto Certificate object created for the validation action (see Figure 1-7 on page 7) will be used.
8. Use the Enter Output Context dialog box to specify the destination context for the signed XML.
9. Click **Done** to complete the sign action.
10. Click **Apply** to add the sign action to the Document Processing Rule.

Encrypt action

An encrypt action performs full or field-level document encryption. Implementation of an encrypt action requires certain parameters that are supplied during service configuration. Use the following procedure to add an Encrypt action to Document Processing Rule.

1. Drag the Encrypt icon and drop it next to the Sign action.
2. Double-click the Encrypt Icon to display Figure 1-11 on page 11.
3. Use the Input dialog box or values-list (a list of contexts available within this Document Processing Rule) to specify the source context for the encrypt action. This is the output of the previous transform action.


Encrypt

Envelope Method: [✓ WSSec Encryption] Standard XML Encryption Advanced

Message Type: [✓ SOAP Message] Raw XML Document Selected Elements (Field-Level) Advanced

Message and Attachment Handling: Message Only ☐ Save

Use Dynamically Configured Recipient Certificate: on [✓ off] ☐ Save

One Ephemeral Key: on [✓ off] ☐ Save

Recipient Certificate: itsocert [up] ✓ Save

Output

Output: tempvar3 (auto)

Figure 1-11 Encrypt action (basic) panel

4. Select **WSSec Encryption** in the Envelope Method.
5. Select **SOAP Message** in the Message Type.
6. Select **Message Only** from the Message and Attachment Handling values-list.
7. Select the **itsocert** object from the Recipient Certificate values-list. In this scenario, we use the same certificate as in the request message. However, you can use different certificates in a real deployment. If you want to create a new certificate object, follow the same procedure as outlined in Figure 1-7 on page 7.

   Note: Additional certificate and encryption settings are available on the Advanced tab.

8. Use the Output dialog box to specify the destination context for the decrypted document. You can create a new one to be used in the results action.
9. Click **Done** to complete the encrypt action.
10. Click **Apply** to add the encrypt action to the Document Processing Rule.

Results action

The Results action completes the response rule and the configuration of the policy. For example, perform the following steps:

1. Drag and drop the Results icon after the Encrypt action.
2. Double-click the Results icon and choose the Input context that was the Output context of the Encrypt action.
3. Click **Done** and click **Apply** to add the Results action to the rule.

1.2.3 Summary of the processing in this chapter

Here is the quick examination of the `ITSO_MPGW_Policy` that we created in the previous section. You can either create this `ITSO_MPGW_Policy` by following steps in the previous section or import the configuration file provided in the additional materials of this Redpaper as described in Appendix C, “Additional material” on page 147. There are two rules in this policy, and they are Request Rule #1 and Response Rule #2.

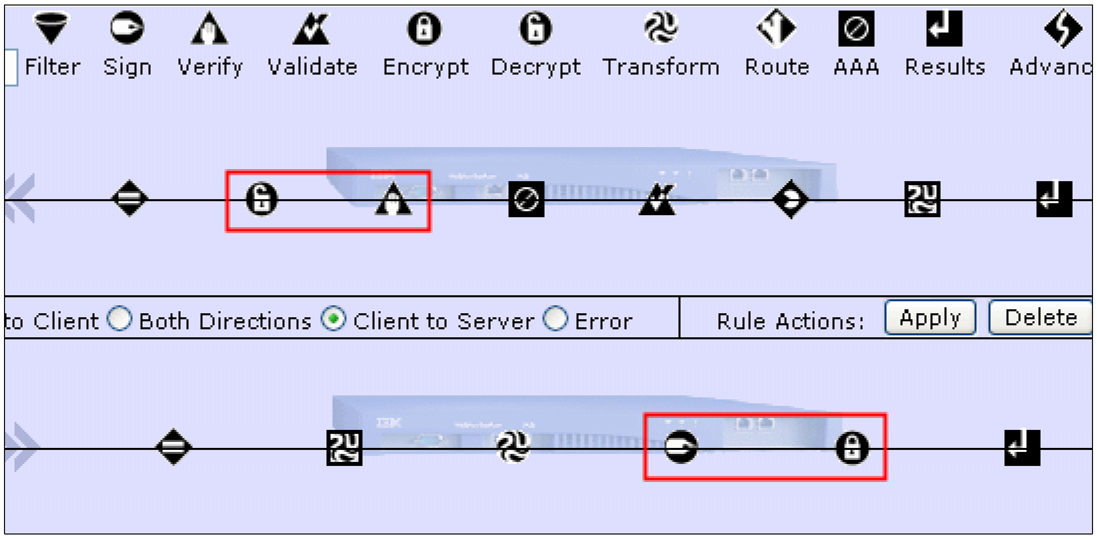


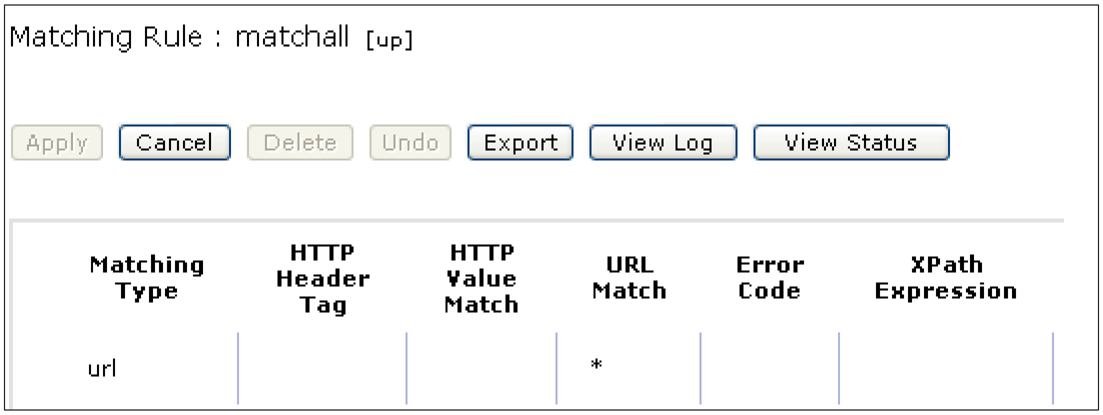
Figure 1-12 End-to-end scenario

Request Rule #1

Here is an examination of the components of Request Rule #1. Note that this is a Request rule; it only handles messages that are sent to the service by the client and does not handle messages sent by the server back to the client. This rule handles encrypted requests.

Match rule

A *Match rule* examines the inbound message and determines whether or not the rule is run against the message. In this case, the match rule examines the URL used by the client to determine a match. In our scenario, we have match *, which means that the gateway accepts all the incoming messages. See Figure 1-13 on page 13. However, in a typical scenario, you can have, for example, `*/SomeBank/someService`.

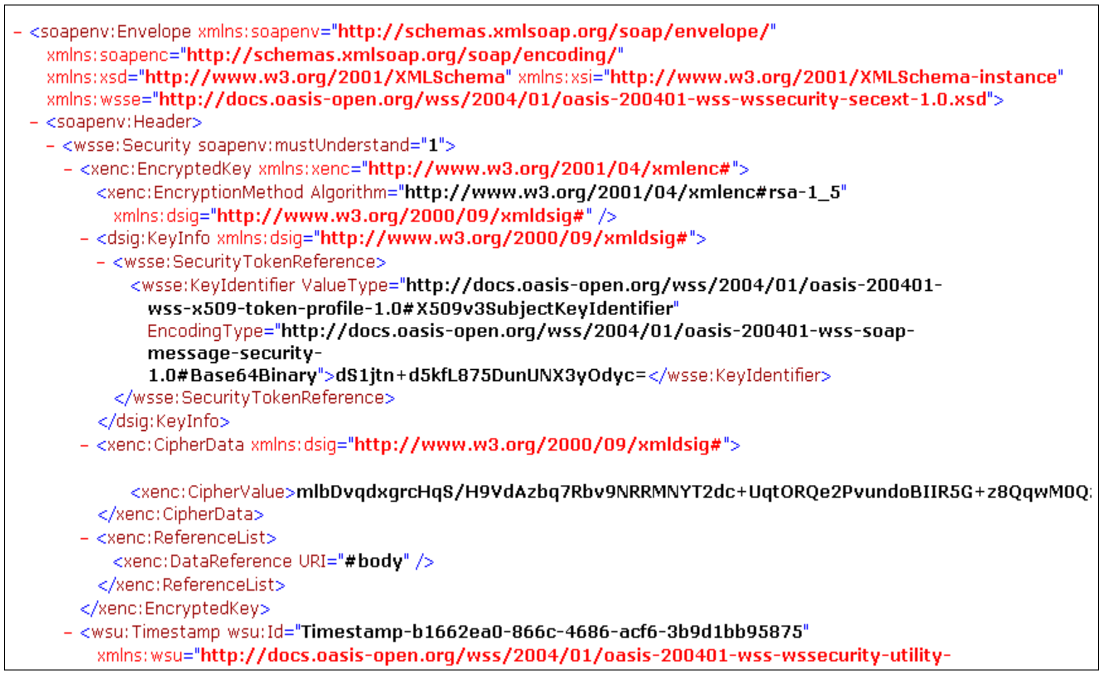

Matching Rule : matchall [up]

Matching Type	HTTP Header Tag	HTTP Value Match	URL Match	Error Code	XPath Expression
url			*		

Figure 1-13 Matching Rule

Action one: Decrypt

A Decrypt action decrypts encrypted messages. In this sample scenario, the entire message has been encrypted. The Decrypt action gets this encrypted message and decrypts it based on the private key. Figure 1-14 shows an example of an encrypted message.



```
- <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
  - <soapenv:Header>
    - <wsse:Security soapenv:mustUnderstand="1">
      - <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
          <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
            xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" />
        - <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
          - <wsse:SecurityTokenReference>
              <wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
                wss-x509-token-profile-1.0#X509v3SubjectKeyIdentifier"
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
                message-security-
                1.0#Base64Binary">dS1jtn+d5kfL875DunUNX3yOdyc=</wsse:KeyIdentifier>
            </wsse:SecurityTokenReference>
          </dsig:KeyInfo>
        - <xenc:CipherData xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">

              <xenc:CipherValue>mlbDvqdxgrcHqS/H9VdAzbq7Rbv9NRRMNYT2dc+UqtORQe2PvundoBIIR5G+z8QqwM0Q:
          </xenc:CipherData>
        - <xenc:ReferenceList>
            <xenc:DataReference URI="#body" />
          </xenc:ReferenceList>
        </xenc:EncryptedKey>
      - <wsu:Timestamp wsu:Id="Timestamp-b1662ea0-866c-4686-acf6-3b9d1bb95875"
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
```

Figure 1-14 Part of the input encrypted message

Action two: Verify

A Verify action verifies the signature contained in the message. If the signature cannot be verified, the message is rejected. If the signature is valid, the rule passes on to the next Action in the processing rule. Because we have a value in the "validation credential", it points to a Crypto Validation Credential object (`itsovalid`). Refer to Figure 1-5 on page 6. That object has to include the certificate included in the message to validate that certificate. Additional trust chain checking can also be performed by a Validation Credential object.

Action three: Authentication, Authorization, and Auditing (AAA)

This action is discussed in detail in *IBM WebSphere DataPower SOA Appliances Part II: Authentication and Authorization*, REDP-4364. After the message passes through AAA, it passes on to the Transform Action, which is the next action in the data processing rule.

Action four: Validate

This action is discussed in detail in *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started,* REDP-4327. After the message passes through Validate, it passes on to the Extract using Xpath Action, which is the next action in the data processing rule.

Action five: Extract using Xpath

This action is discussed in detail in *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started*, REDP-4327. After the message passes through Extract using Xpath action, it passes on to the Transform binary, which is the next action in the data processing rule.

Action six: Transform Binary

This action is discussed in detail in *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started*, REDP-4327. This transform binary uses a WTX Map file for transformation, which is then sent to a back-end existing application for processing.

Response Rule #2

This section examines the components of Response Rule #2.

Match rule

A match rule examines the inbound message and determines whether the rule is run against the message. In this case, the match rule examines the URL used by the client to determine a match. In our scenario, we have match *, which means the gateway accepts all the incoming messages. However, in an actual implementation, you might indicate something similar to `*/SomeUrl`.

Action one: Transform #1

This transform action converts the incoming COBOL message to a SOAP XML message before sending it to the next action on the data processing rule.

Action two: Transform #2

This transform action converts the XML to SOAP using converttosoap.xsl.

Action three: Sign

The message is signed with the private key `itsokey` and the cert `itsocert`.

Action four: Encrypt

This action encrypts the entire body of the message using the WS-Security encryption standard.

1.2.4 Running the service

This section describes the process to run the service. Follow these steps:

1. The WS-Sec form of the input is the inp-encry-final.xml in the additional materials of this Redpaper as described in Appendix C, “Additional material” on page 147.

2. When using Curl, run the following command using the DataPower’s IP address and the front-side HTTP handler’s port:

   ```
   curl -X POST --data-binary @inp-encry-final.xml http://<datapower’s
   ipaddress>:<port>
   ```

2

Integration with WebSphere Message Broker

In this chapter, we explain how to use the IBM WebSphere DataPower SOA Appliances (DataPower) as a front-end XML and Web services security gateway to off-load the security processing from the IBM WebSphere Message Broker (WMB). This solution frees the application developers from the detailed implementation and coding of XML and Web services security and also reduces the impact of the security processing from WebSphere Message Broker and the subsystems that support it.

The Redpaper *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started,* REDP-4327, shows how DataPower extends its capabilities for complex transformation, using the WebSphere Transformation Extender, to act as a Multi-Protocol Gateway and an Enterprise Service Bus (ESB). The ESB provides the protocol switching, routing, and complex transformation capabilities. Together, DataPower and WebSphere Transformation Extender provide the solution to integrate existing applications into a service-oriented architecture (SOA) environment.

Note: In this chapter, we used the information provided by three articles published in developerWorks. Thanks are due to Peter Crocker, Ben Thompson, and Dominic Storey for the following technical articles in DeveloperWorks:

- “Integrating WebSphere DataPower SOA Appliances with WebSphere MQ”:

  http://www.ibm.com/developerworks/websphere/library/techarticles/0703_crocker/0703_crocker.html

- “Integrating WebSphere DataPower XML Security Gateway XS40 with WebSphere Message Broker”:

  http://www.ibm.com/developerworks/websphere/library/techarticles/0710_crocker/0710_crocker.html

- “Integrating DataPower with WebSphere Message Broker using the Broker Explorer”:

  http://www.ibm.com/developerworks/websphere/library/techarticles/0707_storey/0707_storey.html

2.1 Introduction

The objective of this chapter is to show another option for application integration using DataPower and WebSphere Message Broker to provide the same integration capabilities. Unlike the previous option where the DataPower box executes the WebSphere Transformation Extender code, with this second option, the WebSphere Message Broker runs in its own runtime environment.

This second option offers a valuable solution to clients who have a message broker and whose security functions are performed by DataPower. DataPower is the best component designed to optimize security capabilities and is used to provide the first level of security protection. It prevents unauthorized access and filters malicious XML messages before allowing them to flow to the back-end systems. DataPower also allows security functions to be centralized, administered, managed, and controlled in a DataPower appliance from a central point of control. The message broker is dedicated to doing what it is best known for: its advanced and rich application integration capabilities.

We discuss the following major topics in this chapter:

- Scenarios description
- Configuration of DataPower and WebSphere Message Broker for scenario one
- Configuration of DataPower and WebSphere Message Broker for scenario two
- Testing the scenarios

Note: See Appendix B, “Building message flows in WebSphere Message Broker” on page 119 for information about how to build the WebSphere Message Broker artifacts.

2.2 Sample scenarios description

We will use two major scenarios to illustrate the options for integrating an SOA application with an existing application using the DataPower appliance and WebSphere Message Broker. In the first scenario, WebSphere Message Broker is used to provide the interface to the existing application as a Web service. In the second scenario, WebSphere Message Broker uses the classic way to provide the interface to the existing application via MQ. These scenarios use different protocols (HTTP and MQ) to demonstrate the options for integration between WebSphere Message Broker and DataPower. We use the following components of DataPower for front-end processing:

- XML firewall and Web service gateway
- Multi-protocol gateway

Both scenarios show an example of a message request and reply processing pattern where the information, such as the message identifier (MQ), the ReplyToQ, and the ReplyToQmgr or the HTTP/HTTPS Identifier of the original request, must be persisted on the inbound. This information is retrieved on the outbound and is used by the message broker to build the reply message header with routing and correlation information.

The two scenarios use the same data structures, transformation logic, and connectivity to the back-end CICS application via MQ. The key differences between the two scenarios are the protocols that they use for communication between DataPower and WebSphere Message Broker and the DataPower components that are used to provide the front-end services.

In both scenarios, there is a clear separation of concerns where the DataPower performs the security functions and the message broker acts as an ESB, performing the data transformation and connectivity to the back-end system. We will describe these scenarios in detail.

Scenario one: Integrating DataPower with WebSphere Message Broker

The first scenario uses HTTP/SOAP and HTTPS/SOAP protocols for connectivity between DataPower and WebSphere Message Broker. WebSphere Message Broker provides the Web services interface to the existing application. In this scenario, the message traffic goes through the XML Firewall Gateway within DataPower for Web services security processing.

This scenario has the following characteristics:

- HTTP or HTTPS protocol is used to communicate between DataPower and WebSphere Message Broker.
- WebSphere Message Broker switches from HTTP/HTTPS to MQ protocol to connect to the CICS application.
- Request and reply SOAP messages are encrypted, decrypted, and optionally signed.
- DataPower provides the front-end XML/Web services security gateway and message decryption and signature verification (inbound) and message encryption and digital signature (outbound) using the XML Firewall Gateway within DataPower.
- WebSphere Message Broker performs the transformation from SOAP to fixed format/COBOL (inbound) and from fixed format/COBOL to SOAP (outbound). The WebSphere Message Broker switches to the MQ protocol to connect to the back-end (CICS) application.
- This scenario uses the Request/Reply processing pattern.

Figure 2-1 shows the end-to-end and round-trip processing sequence for the inbound and outbound traffic, including the requesting application, DataPower, WebSphere Message Broker, and the CICS application. Figure 2-1 is also used to configure DataPower and WebSphere Message Broker to support scenario one.

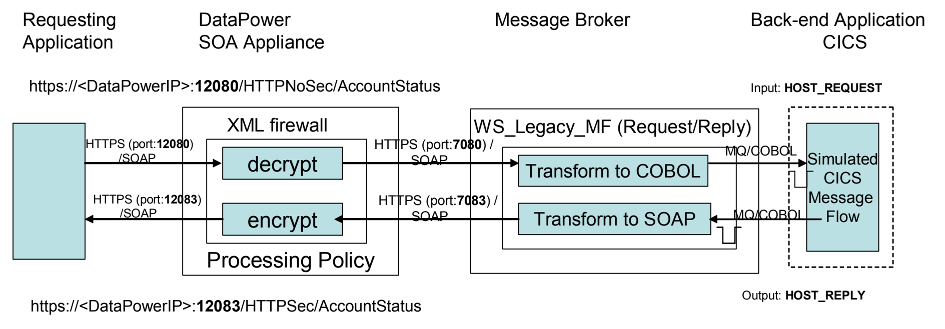


Figure 2-1 End-to-end processing flow for scenario one

Scenario two: Integrating DataPower Appliance with WMB via MQ

The second scenario uses the MQ protocol for connectivity to DataPower and the message broker. In this scenario, the message traffic goes through the DataPower Multi-Protocol Gateway for Web services security processing. This scenario has the following characteristics:

- MQ protocol used for inbound/outbound communication between DataPower and the message broker.
- MQ is used to connect to the back-end application.
- The request and reply SOAP message is encrypted.
- DataPower provides the front-end Multi-Protocol Gateway and message decryption (inbound) and encryption (outbound).
- WebSphere Message Broker performs the transformation from SOAP to fixed format/COBOL (inbound) and from fixed format/COBOL to SOAP (outbound).
- This scenario uses the Request/Reply processing pattern.

Figure 2-2 shows the end-to-end and round-trip processing sequence for the inbound and outbound traffic, including the requesting application, DataPower, WebSphere Message Broker, and the CICS application. This figure is also used to configure DataPower and WebSphere Message Broker to support scenario two.

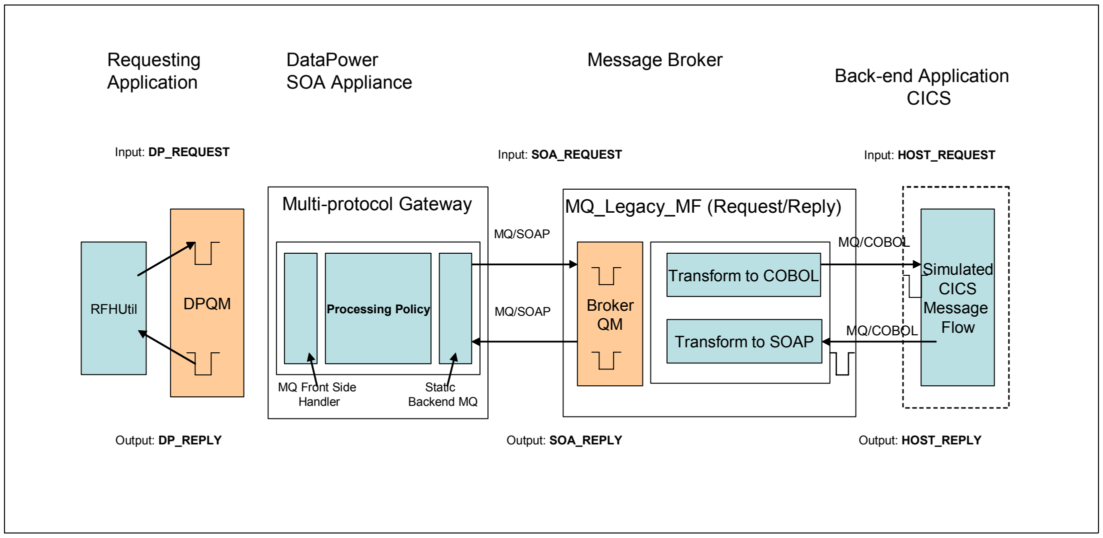


Figure 2-2 End-to-end processing flow for scenario two

2.3 Configuration of DataPower and WMB for Scenario one

This section explains the message flows and how to configure DataPower and WebSphere Message Broker to test the prototype, including simple problem determination. We assume that you have basic knowledge of the WMB installation, configuration, and development. The installation and basic configuration of the WMB and DataPower are not included in this Redpaper.

For your convenience, we provided the completed and working message set and message flow projects that you can import into your WebSphere Message Brokers Toolkit. You can build the WebSphere Message Broker artifacts yourself by following the instructions documented in Appendix B, “Building message flows in WebSphere Message Broker” on page 119. You can download the WebSphere Message Broker prebuilt artifacts and scripts to set up MQ and WebSphere Message Broker from the additional materials link.

The prototypes use the following software and hardware:

- IBM WebSphere DataPower SOA Appliances XI50 (licensed for WMQ and TAM)
- IBM WebSphere Message Broker V6.0.0.5
- IBM WebSphere MQ V6.0.2
- IS02 support pack: WebSphere Message Broker Explorer plug-in for WebSphere MQ Explorer
- IBM DB2® V8.0

WebSphere Message Broker and its dependent software run on Windows® in our sample scenario environment.

Integrating the DataPower appliance with WMB via HTTP(S)

The main objective of this section is to explain how to configure the XML firewall in DataPower to provide Web service security and to configure the message broker to integrate with DataPower using HTTP and HTTPS. WebSphere Message Broker V6.0.0.5 does not support WS-Security. For that reason, we use DataPower to act as a Web Service security gateway to allow WS-Security-enabled applications to interface with WebSphere Message Broker.

First, you will learn the message flow logic, which provides the processing sequence for transformation and connectivity to the front-end and back-end subsystems. We will explain how to build the message broker artifacts in Appendix B, “Building message flows in WebSphere Message Broker” on page 119.

Next, you will learn to perform the following tasks:

- Configure the WebSphere Message Broker.
- Configure DataPower XML Firewall as a Web services security gateway: Use the DataPower security wizard, which is provided as part of the WebSphere Message Broker Explorer plug-in support pack IS02, to configure DataPower from WebSphere MQ Explorer.
- Configure DataPower to add a digital signature to the processing policy.
- Test the configuration using the message flows.

2.3.1 Message flow logic

WebSphere Message Broker is used to provide the integration services; it is important that you understand the processing logic of the message flows. It requires the following two message flows to execute scenario one.

Host_Simulation_MF

This flow simulates the CICS COBOL application. It retrieves the message from the HOST_REQUEST queue, fills in the customer information and status via the Set_Customer_Info_Status compute node, and sends a reply message to the ReplyToQ specified in the MQ header MQMD via the MQReply node. This message flow logic is illustrated in Figure 2-3 on page 22.

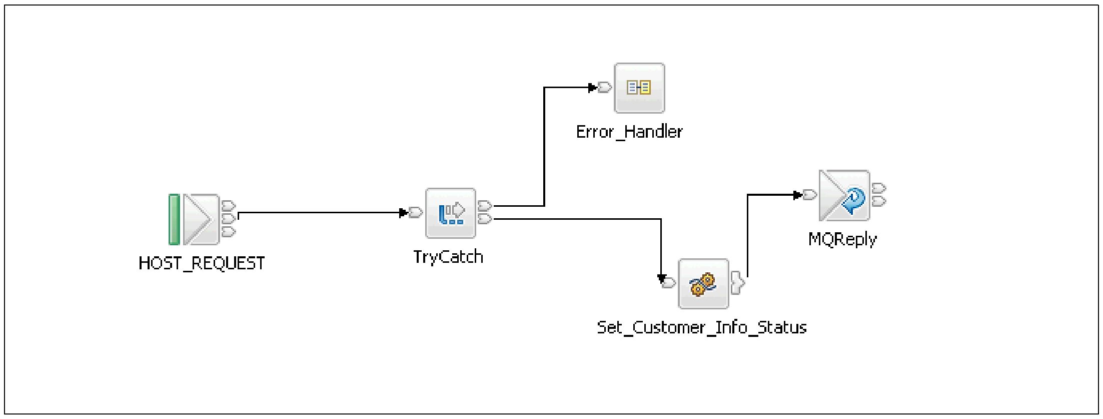


Figure 2-3 Host_Simulation_MF message flow

WS_Legacy_MF

This is the main flow that provides connectivity to DataPower and the CICS application. It also performs the transformation from XML/SOAP to COBOL on the inbound and COBOL to XML on the outbound. This message flow has two paths and supports both HTTP and HTTPS requests. The two paths are:

- **Inbound path:**
  a. Receives the SOAP message via the HTTP/HTTPS node. The input message is a request for customer information (name, address, phone, and so forth) and status (P = Premium, G = Gold, and S = Silver) based on an account number. The SOAP message body contains only one field, which is the account number.
  b. Transforms the SOAP message from XML into a fixed format via the XML_To_COBOL compute node.
  c. Puts the transformed message to the HOST_REQUEST queue for the CICS Simulation flow to pick it up. This message flow was created to simulate the CICS application. It reads the request message, builds the output message with customer information and status, and sends the output message to the ReplytoQ specified in the MQMD header. The customer information is hard-coded for simplicity.
  d. Builds the second XML message to store the HTTP Identifier/context via the Set_HTTP_Id compute node. This HTTP Identifier is correlated to the message that was sent to the existing application.
  e. This state message is put to the HTTP_STATEQ queue for later retrieval (see Outbound path).
- **Outbound path:**
  a. An MQ reply message is put to the HOST_REPLY queue by the CICS Simulation flow.
  b. The fixed format message from CICS is transformed into a SOAP message body.
  c. The HTTP Identifier is retrieved from the HTTP_STATEQ queue.
  d. The reply SOAP message is sent to the original requester via the HTTP Reply node.

The message flow WS_Legacy_MF processing logic is illustrated in Figure 2-4 on page 23.

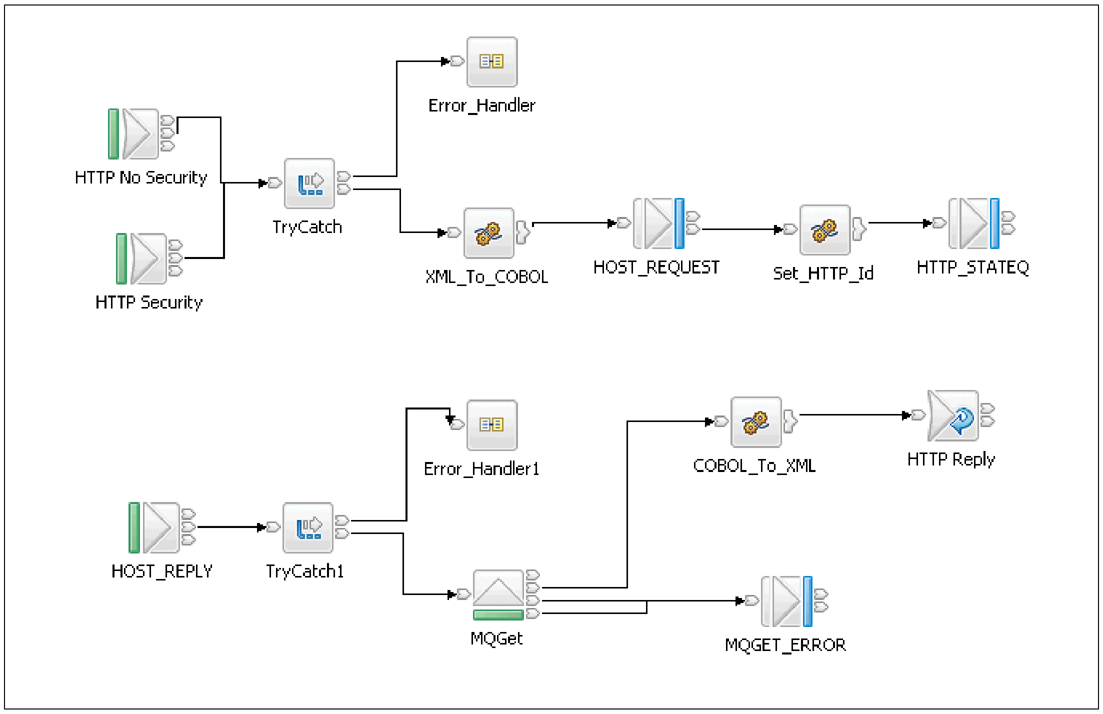


Figure 2-4 WS_to_Legacy_MF message flow

Both the WS_To_Legacy_MF and the Host_Simulation_MF use the following common subflow Error_Handler to catch all the exceptions. It provides the following processing sequences:

- Parses the exception list to get the last error number and exception text to build the error message. This processing logic is done in the ExceptionList compute node.
- Puts the error message in the CATCH_ALL_ERRORS queue.
- Throws the exception to back out the transaction.

Figure 2-5 shows the processing sequence of the Error_Handler subflow.

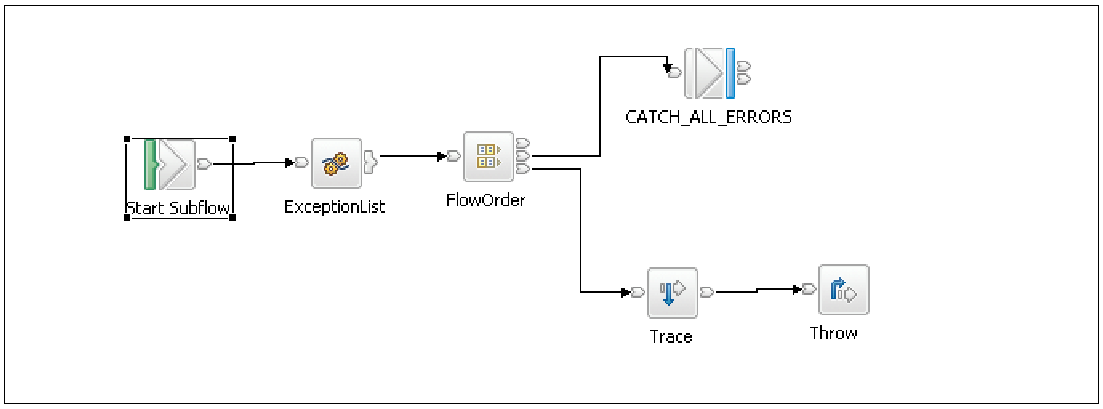


Figure 2-5 Error_Handler subflow

2.3.2 Configure WebSphere Message Broker

We need to set up the environment for WebSphere Message Broker to run the prototype. WebSphere Message Broker uses many queues that must be defined in the queue manager.

Also, it uses the HTTP over SSL (HTTPS) protocol, which requires the configuration of the HTTPS connection to the message broker.

The following steps are required to configure the message broker to run the first scenario:

1. Download the artifacts from the Additional Materials link.
2. Define the queues.
3. Deploy the message flows and message sets to a message broker.
4. Configure the HTTP and HTTPS input nodes.
5. Configure the HTTPS connection to the message broker.
6. Install the WebSphere MQ Explorer support pack IS02.

The message broker environment consists of the following components:

- Queue manager: WBRK6_QUEUE_MANAGER
- Message Broker: WBRK6_DEFAULT_BROKER
- Configuration manager: WBRK6_DEFAULT_CONFIGURATION_MANAGER

Step 1: Download the artifacts from the Additional Materials link

We provide all of the necessary artifacts to test both scenarios. You do not have to build message broker artifacts. However, we provide a section explaining how to build WebSphere Message Broker artifacts if you want to build them yourself. You can download the artifacts for this Redpaper from the Additional Materials link:

`http://www.redbooks.ibm.com/Redbooks.nsf/pages/addmats`

Note: See Appendix C, “Additional material” on page 147 to download the artifacts that are used to configure WMB before you can test the scenarios.

Table 2-1 lists the artifacts.

Table 2-1 Artifacts

File names	Purposes
WMBIntegrationDomain.zip	WMBIntegration application domain within DataPower
DPWMBIntegration.zip	Project Interchange that contains message sets, message flows, and SOA2Legacy.bar file
HTTPS_Conf	Commands to configure WMB for HTTPS listener
MQ_Conf	Commands to define queues for WMB
Test_GenEncryptSOAP.cmd	Command to generate the encrypted SOAP message for testing the XML Loopback firewall after you have completed its configuration for message encryption/decryption. You can skip this step if you choose to use the prebuilt message provided for your testing.
Test_GenEncSignSOAP.cmd	Command to generate an encrypted and signed SOAP message after you add a digital signature to the processing policy of the Loopback XML firewall. You can skip this step if you choose to use the prebuilt message provided for your testing.
Test_RegularSOAP_MB_HTTP.cmd	Command to test the regular SOAP message with WMB/HTTP

File names	Purposes
Test_RegularSOAP_MB_HTTPS.cmd	Command to test the regular SOAP message with WMB/HTTPS
Test_EncSignSOAP_DP_HTTP.cmd	Command to test encrypted and signed SOAP messages with DataPower/HTTP
Test_EncSignSOAP_DP_HTTPS.cmd	Command to test encrypted and signed SOAP messages with DataPower/HTTPS
Test_EncryptSOAP_DP_HTTP.cmd	Command to test encrypted SOAP messages with DataPower/HTTP
Test_EncryptSOAP_DP_HTTPS.cmd	Command to test encrypted SOAP messages with DataPower /HTTPS
TestEncryptSOAP_IN.xml	Data for testing encrypted SOAP messages. This file is provided for your testing. We used the Test_GenEncSignSOAP.cmd to generate this message. You can also generate the encrypted test message for your environment using the Loopback firewall that you created.
TestEncSignSOAP_IN.xml	Data for testing encrypted and signed SOAP messages. This file is generated by the Test_GenEncSignSOAP.cmd. You can use the message provided for your testing or generate the encrypted and signed test message yourself using the Loopback firewall that you created.
TestRegularSOAP_IN.xml	Data for testing regular SOAP messages
TestInvalidSOAP_IN.xml	Data for testing invalid SOAP messages

Step 2: Define queues

The message flows use many queues that are defined using the MQ_Conf.bat script. This script defines the queues for both scenarios. Table 2-2 lists the names of the queues and their associated functions.

Table 2-2 Queues used by the WS_To_Legacy_MF and MQ_To_Legacy_MF message flows

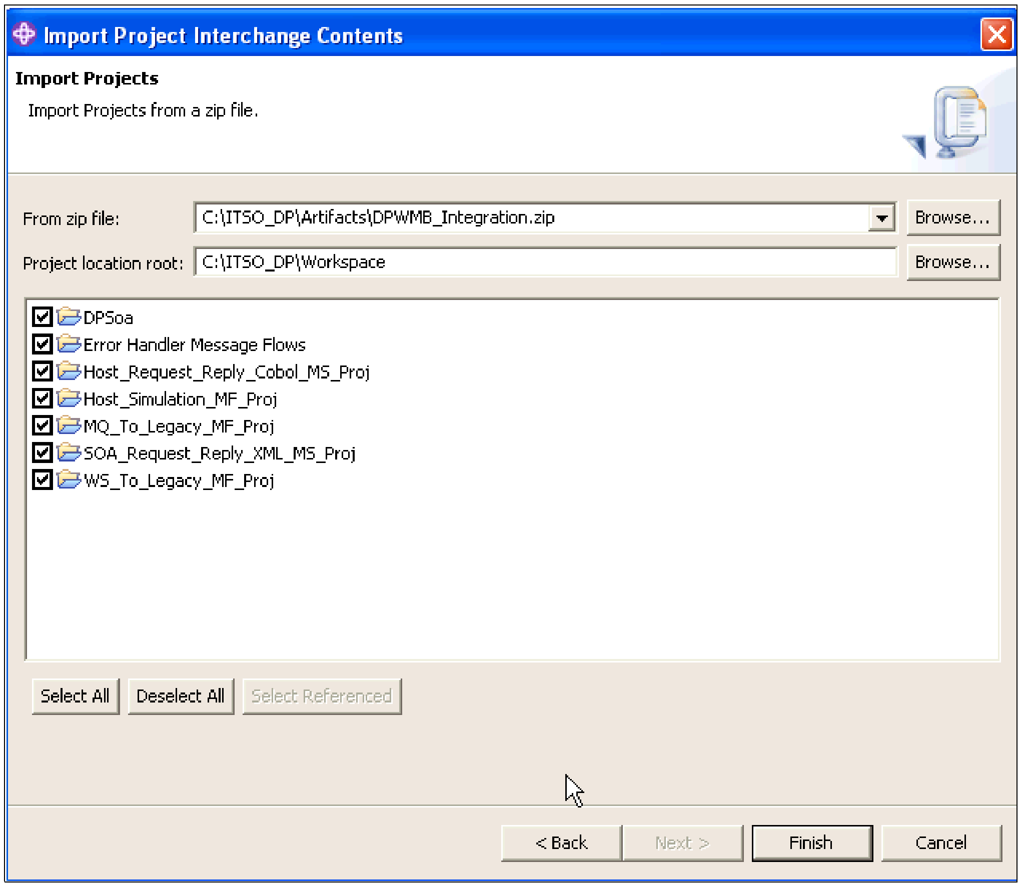
Queue name	Function
CATCH_ALL_ERRORS	Used to store the error message for any errors encountered in any node in the message flow where the failure terminal is not wired
DP_GET	Queue used by DataPower to retrieve a request message from an MQ client application
DP_PUT	Queue used by DataPower to send a response message to an MQ client application
CATCH_ALL_ERRORS	Queue to catch all errors from message flows
HOST_REQUEST	Request queue for CICS application
HOST_REPLY	Reply queue for CICS application if a request comes from WS_To_Legacy_MF message flow
HOST_REPLY2	Reply queue for CICS application if a request comes from MQ_To_Legacy message flow

Queue name	Function
HTTP_STATEQ	Queue used to store the HTTP context, such as the HTTP identifier
SOA_REQUEST	Request queue to start the MQ_To_Legacy flow
SOA_REPLY	Reply queue used by WMB to put the response message to DataPower
MQGET_ERROR	Used to store the error message for the MQGET when no message returned after the wait interval has timed out or when the MQGET gets a warning return code
MQMD_STOREQ	Queue used to store the message/correlation identifier

Step 3: Deploy the message sets and the message flows

We provide the prebuilt project interchange that contains the message set and message flow projects. It also includes the broker archive (bar) file that contains the compiled code to be deployed into the message broker runtime. You can import the project interchanges into WebSphere Message Brokers toolkit and deploy them to your message broker. The following steps show you how to do it:

1. Import the project interchange into the message broker:
   a. Click **Start** → **Program File** → **IBM WebSphere Message Brokers 6.0** → **Command Console**.
   b. Click **File** → **Import** → **Project Interchange** → **Next**.
   c. Click **Browse** to select C:\ITSO_DP\Artifacts\DPWMBIntegration.zip, click **Select All,** and click **Finish** as shown in Figure 2-6 on page 27.


Import Project Interchange Contents
Import Projects
Import Projects from a zip file.

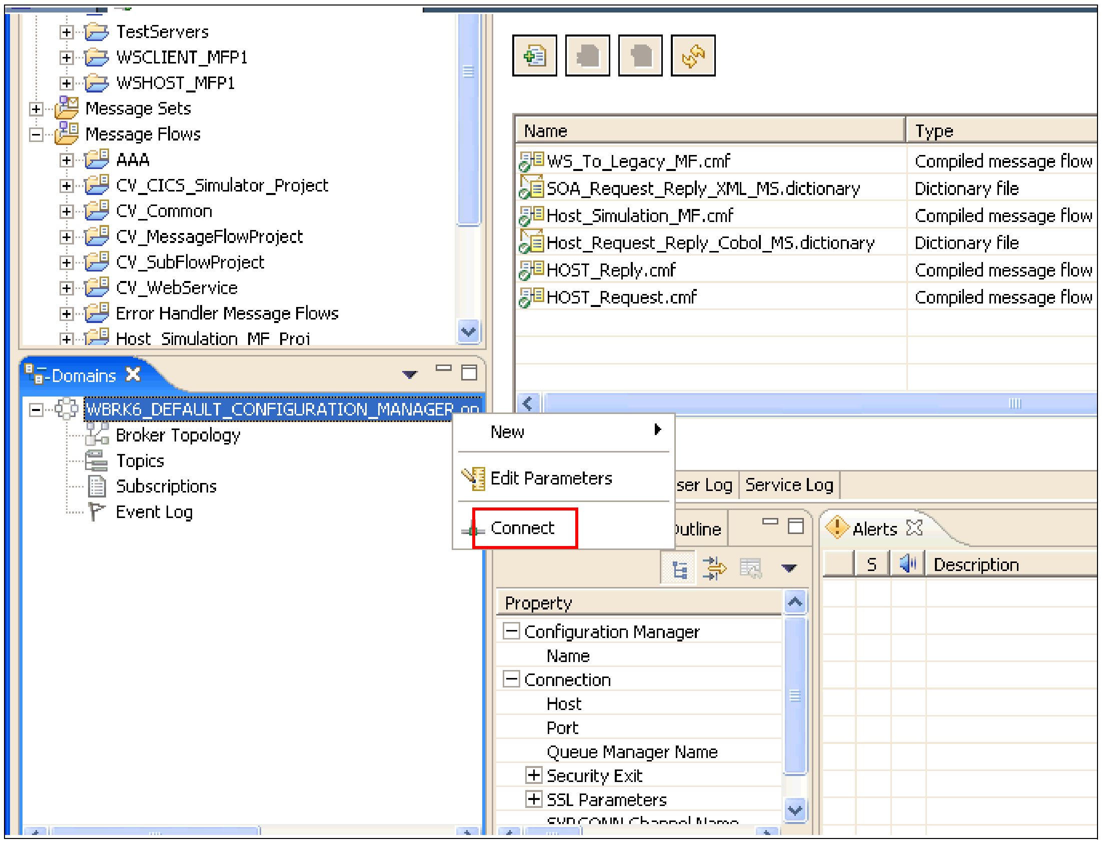
From zip file: C:\ITSO_DP\Artifacts\DPWMB_Integration.zip

Project location root: C:\ITSO_DP\Workspace

- [✓] DPSoa
- [✓] Error Handler Message Flows
- [✓] Host_Request_Reply_Cobol_MS_Proj
- [✓] Host_Simulation_MF_Proj
- [✓] MQ_To_Legacy_MF_Proj
- [✓] SOA_Request_Reply_XML_MS_Proj
- [✓] WS_To_Legacy_MF_Proj

Figure 2-6 Import Project Interchange

2. Deploy the SOA2Legacy.bar file to the message broker:
   a. Create a new Execution group called `DPSoa`.
   b. Connect the Configuration Manager to the message broker as shown in Figure 2-7 on page 28.


TestServers
WSCLIENT_MFP1
WSHOST_MFP1
Message Sets
Message Flows
AAA
CV_CICS_Simulator_Project
CV_Common
CV_MessageFlowProject
CV_SubFlowProject
CV_WebService
Error Handler Message Flows
Host_Simulation_MF_Proj

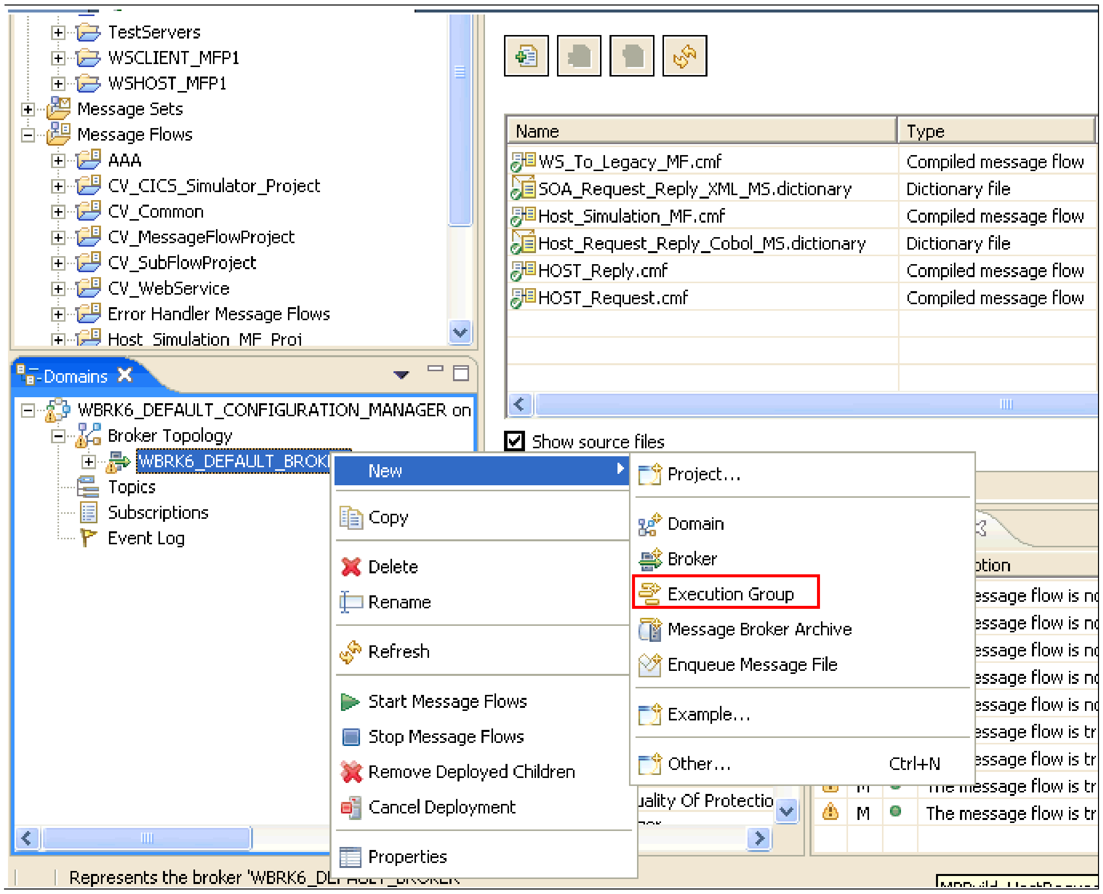
Name	Type
WS_To_Legacy_MF.cmf	Compiled message flow
SOA_Request_Reply_XML_MS.dictionary	Dictionary file
Host_Simulation_MF.cmf	Compiled message flow
Host_Request_Reply_Cobol_MS.dictionary	Dictionary file
HOST_Reply.cmf	Compiled message flow
HOST_Request.cmf	Compiled message flow

Domains
WBRK6_DEFAULT_CONFIGURATION_MANAGER_on
Broker Topology
Topics
Subscriptions
Event Log

Property
Configuration Manager
Name
Connection
Host
Port
Queue Manager Name
Security Exit
SSL Parameters
SVRCONN Channel Name

Figure 2-7 Connect configuration manager to broker

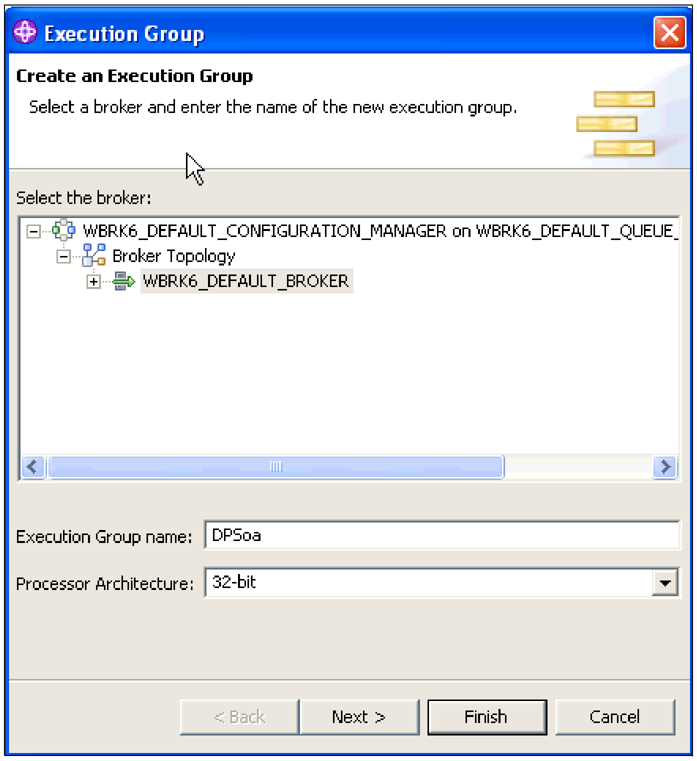
c. Right-click the message broker, click **New** and click **Execution Group**, as shown in Figure 2-8 on page 29.



Name	Type
WS_To_Legacy_MF.cmf	Compiled message flow
SOA_Request_Reply_XML_MS.dictionary	Dictionary file
Host_Simulation_MF.cmf	Compiled message flow
Host_Request_Reply_Cobol_MS.dictionary	Dictionary file
HOST_Reply.cmf	Compiled message flow
HOST_Request.cmf	Compiled message flow

Figure 2-8 Create a new Execution Group

d. Set the Execution Group Name to `DPSoa`, as shown in Figure 2-9 on page 30.


Execution Group

Create an Execution Group

Select a broker and enter the name of the new execution group.

Select the broker:

WBRK6_DEFAULT_CONFIGURATION_MANAGER on WBRK6_DEFAULT_QUEUE_

Broker Topology

WBRK6_DEFAULT_BROKER

Execution Group name: DPSoa

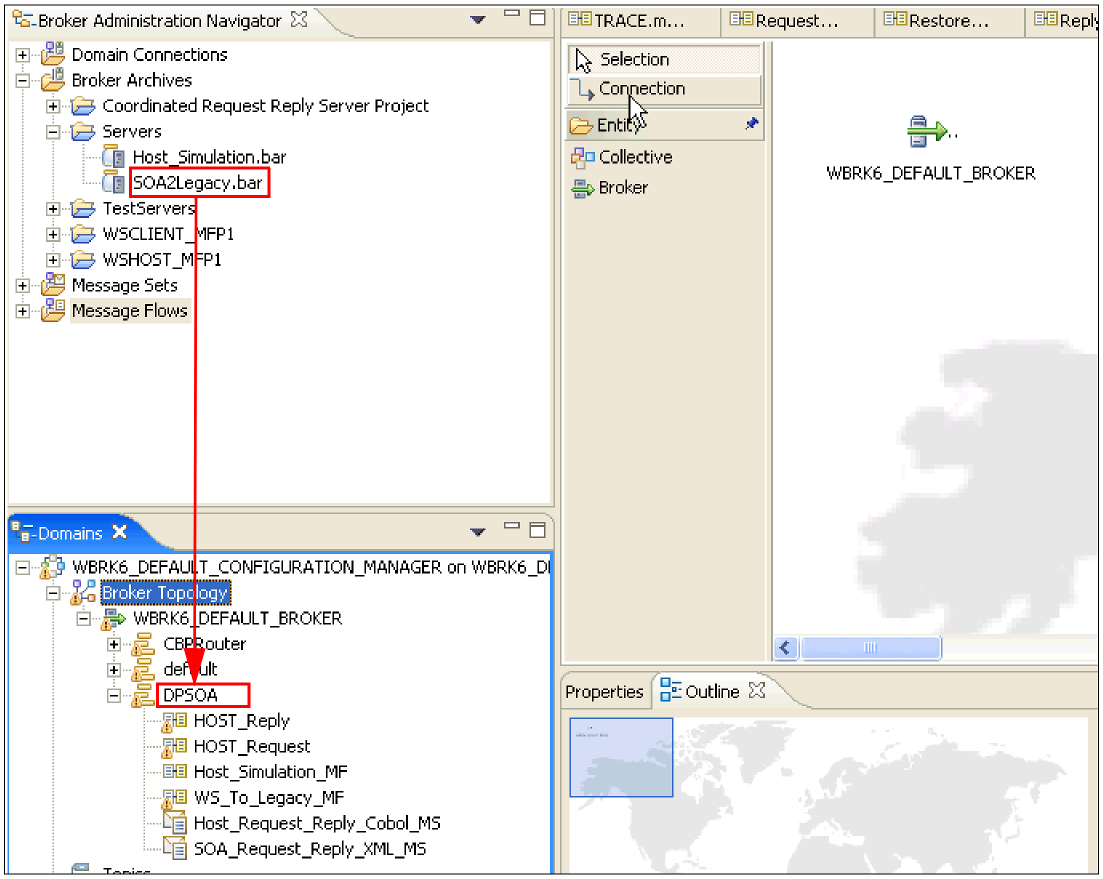
Processor Architecture: 32-bit

Figure 2-9   Create a new Execution Group

e. Deploy the Soa2Legacy bar file to DPSoa Execution Group:

   i. Drag and drop Soa2Legacy.bar into the DPSoa Execution Group as shown in Figure 2-10 on page 31.

   ii. Verify that the bar has been successfully deployed with the following artifacts displayed under DPSoa Execution Group: Host_Reply, Host_Request, Host_Simulation_MF, WS_To_Legacy_MF, Host_Request_Reply_Cobol_MS, and Host_Request_Reply_XML_MS.


Broker Administration Navigator

- Domain Connections
- Broker Archives
  - Coordinated Request Reply Server Project
  - Servers
    - Host_Simulation.bar
    - SOA2Legacy.bar
  - TestServers
  - WSCLIENT_MFP1
  - WSHOST_MFP1
- Message Sets
- Message Flows

Domains

- WBRK6_DEFAULT_CONFIGURATION_MANAGER on WBRK6_DI
  - Broker Topology
    - WBRK6_DEFAULT_BROKER
      - CBPRouter
      - default
      - DPSOA
        - HOST_Reply
        - HOST_Request
        - Host_Simulation_MF
        - WS_To_Legacy_MF
        - Host_Request_Reply_Cobol_MS
        - SOA_Request_Reply_XML_MS

Figure 2-10   Deploy SOA2Legacy.bar

Step 4: Configure the HTTP and HTTPS input nodes

The message flows used by this sample have two HTTP nodes: one node is configured for the HTTP protocol and the other node is configured for the HTTP over Secure Sockets Layer (SSL) protocol. Each of these nodes must be configured to listen on one of the following URLs:

- HTTP No Security input node: /HTTPNoSec/AccountStatus URL
- HTTP Security input node: /HTTPSec/AccountStatus URL

Figure 2-11 shows the HTTP and HTTPS input nodes.

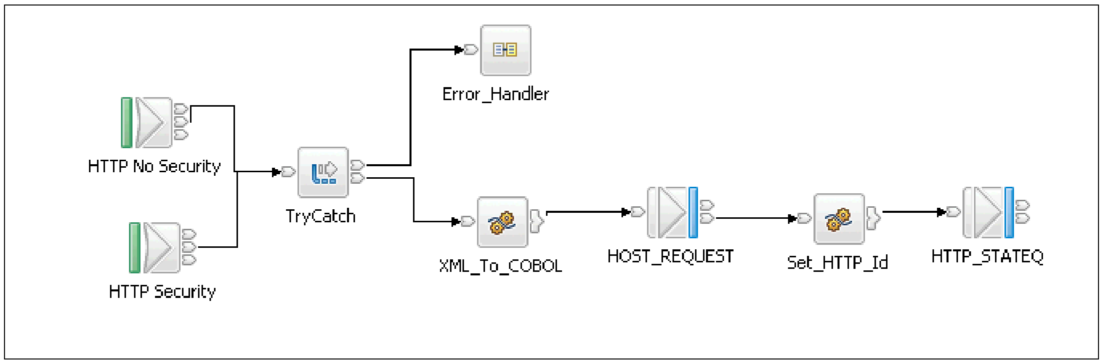
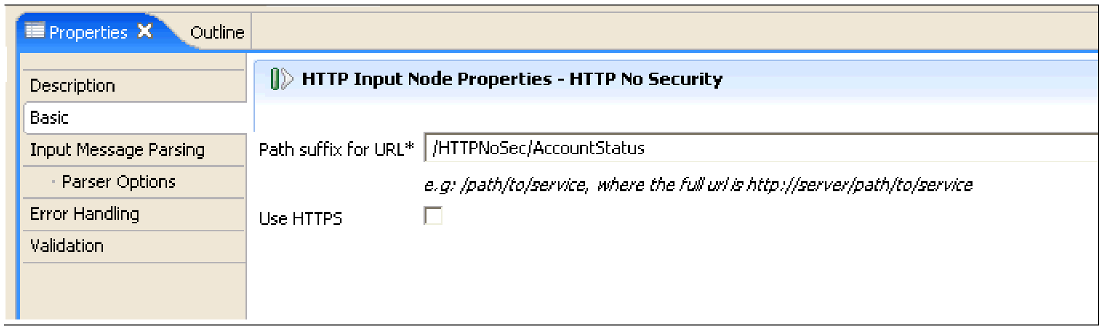


Figure 2-11   HTTP and HTTPS input nodes

To configure the HTTP input nodes:

1. Configure the HTTP No Security Input node as shown in Figure 2-12 by following these steps:
   a. Right-click the HTTP No Security Input node graphic and select the **Properties** tab at the bottom.
   b. Click the **Basic** tab and set the name of the Path suffix for URL to `/HTTPNoSec/AccountStatus`.
   c. Save the change by pressing the Ctrl + S keys.



HTTP Input Node Properties - HTTP No Security

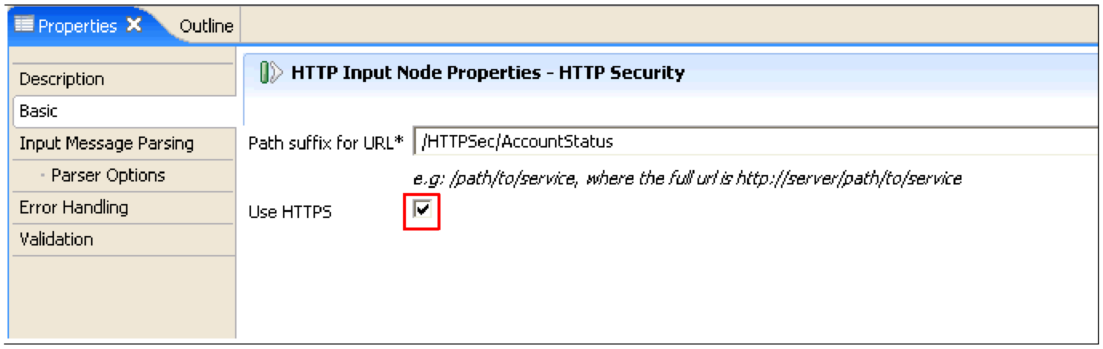
Path suffix for URL* /HTTPNoSec/AccountStatus

e.g: /path/to/service, where the full url is http://server/path/to/service

Use HTTPS ☐

Figure 2-12 HTTP No Security node configuration

2. Configure the HTTP Security input node as shown in "HTTP Security input node configuration" on page 32 by following these steps:
   a. Right-click the HTTP Security node graphic and select the **Properties** tab.
   b. Click the **Basic** tab and set the name of the Path suffix for URL to `/HTTPSec/AccountStatus`.
   c. Select the check box **Use HTTPS**.
   d. Save the change by pressing the Ctrl + S keys.



HTTP Input Node Properties - HTTP Security

Path suffix for URL* /HTTPSec/AccountStatus

e.g: /path/to/service, where the full url is http://server/path/to/service

Use HTTPS ☑

Figure 2-13 HTTP Security input node configuration

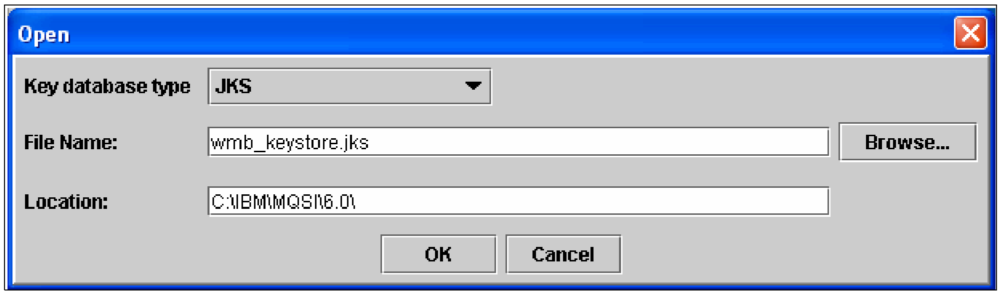
Note: If you choose to build the message flows by yourself or if you imported the completed project interchange, the configuration of the HTTP input nodes is done in Appendix B, "Building message flows in WebSphere Message Broker" on page 119.

Step 5: Configure HTTPS connection to the message broker

We explain how to set the broker properties for HTTP over SSL for the HTTPS input node to work with a message broker. You can set all of these properties by using the commands supplied by the HTTPS_Conf.bat from the download zip file.

First, you have to create a key store to store the broker's certificates before you can run the script to configure the HTTPS connection to the broker:

1. Create a key store file to store the broker's certificates as shown in Figure 2-14.
   a. **Start** → **IBM WebSphere Message Broker 6.0** → **Command Console** to open the broker command console.
   b. Type `ikeyman` on the command line.
   c. On the IBM Key Management panel, click **Key Database File** → **New**.
   d. Select the Key Database type **JKS** from the drop-down list box.
   e. Enter the File Name: `wmb_keystore.jks` and click **OK**.

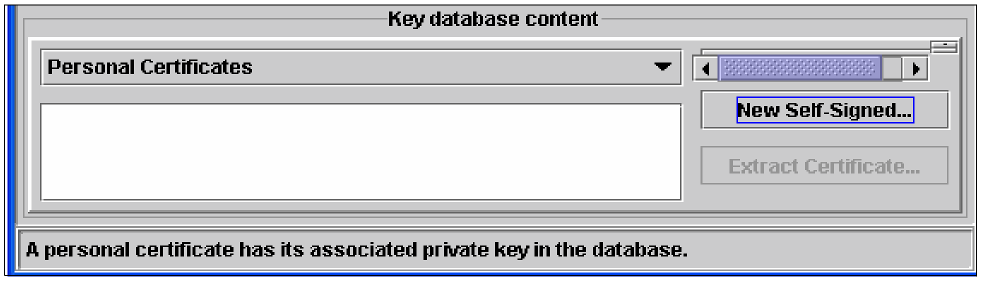


Key database type: JKS
File Name: wmb_keystore.jks
Location: C:\IBM\MQSI\6.0\

Figure 2-14 Create keystore

   f. Enter the password on the pop-up menu and click **OK**.
   g. This password is used to restrict access to the file. The keystore is now created and is ready for use by the broker.

2. Create a new self-signed personal certificate:
   a. Select **Personal Certificates** from the drop-down list under **Key database content.**
   b. Click **New Self-Signed** as shown in Figure 2-15.

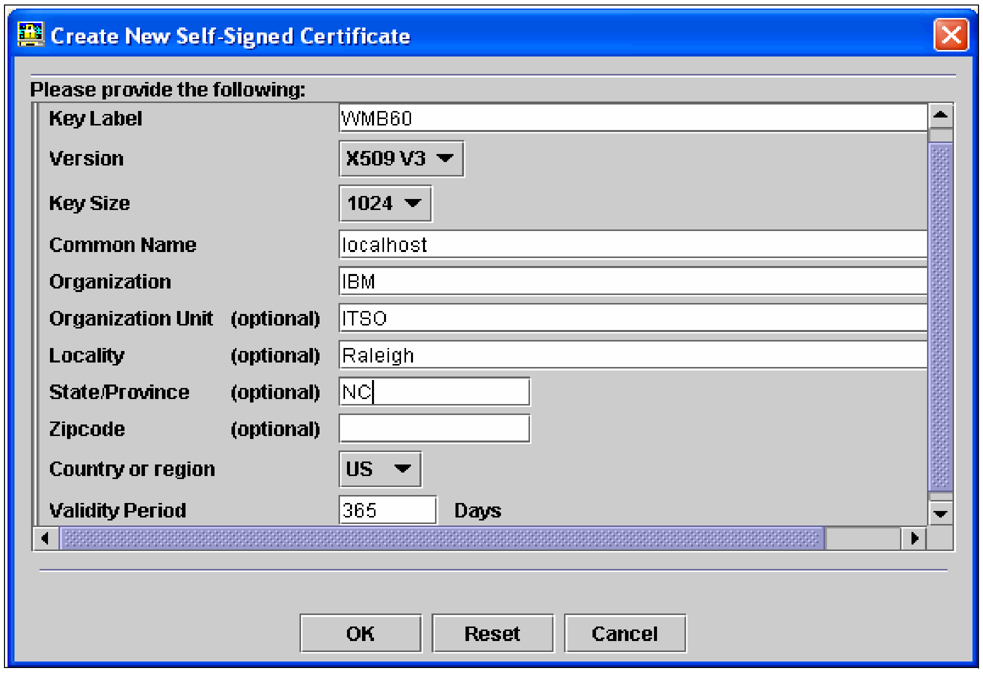


A personal certificate has its associated private key in the database.

Figure 2-15 Personal Certificates creation panel

   c. Enter the following values in the field names: Key Label: `WMB60`, Common Name: `localhost`, Organization: `IBM`, Organization Unit: `ITSO`, Locality: `Raleigh`, and State/Province: `NC` as shown in Figure 2-16 on page 34.

Note: The common name must match the host name. You can choose other values for your organization, but we recommend that the common name match your host name in a production environment. It is not that important to match the common name to a host name in a test environment.

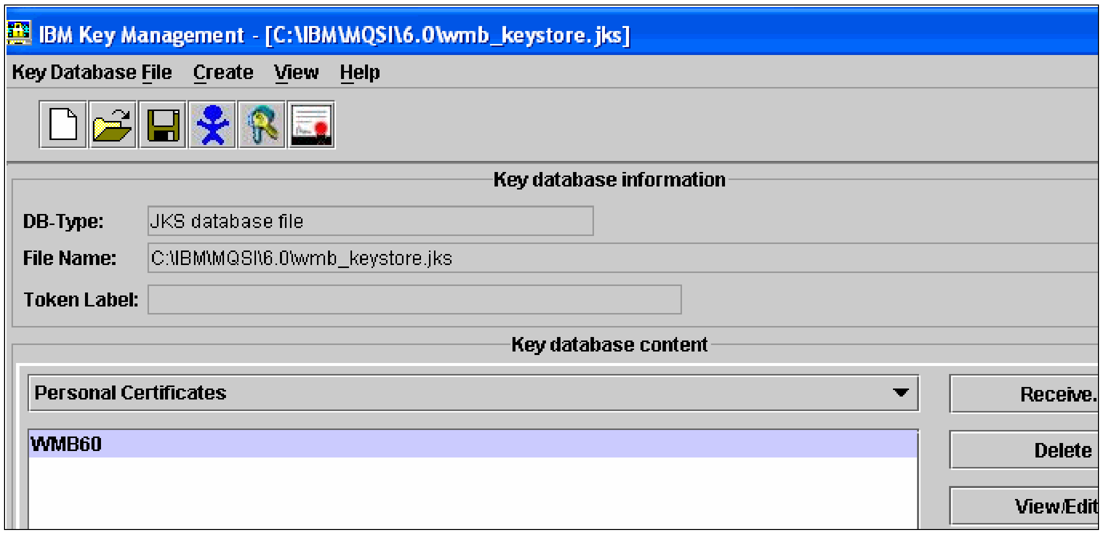

Create New Self-Signed Certificate

Please provide the following:

Key Label	WMB60
Version	X509 V3
Key Size	1024
Common Name	localhost
Organization	IBM
Organization Unit (optional)	ITSO
Locality (optional)	Raleigh
State/Province (optional)	NC
Zipcode (optional)	
Country or region	US
Validity Period	365 Days

Figure 2-16 Information used to create a personal self-signed certificate

d. Click **OK**. The personal certificate `WMB60` is now created and stored in the `wmb_keystore` as shown in Figure 2-17.


IBM Key Management - [C:\IBM\MQSI\6.0\wmb_keystore.jks]

Key database information

DB-Type:	JKS database file
File Name:	C:\IBM\MQSI\6.0\wmb_keystore.jks
Token Label:	

Key database content

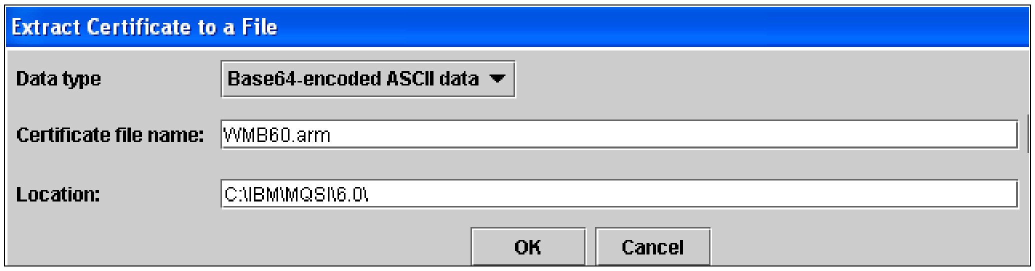
Personal Certificates

WMB60

Figure 2-17 WMB60 personal certificate

e. Click **Extract Certificate** and click **OK** as shown in Figure 2-18 on page 35.

Note: You only need to perform this step if you want to use the certificate with DataPower for a two-way authentication. You must upload this certificate into the DataPower appliance and add it to a trusted server within a client SSL profile. You can trust that DataPower appliance connects only to trusted servers with this configuration.



Data type: Base64-encoded ASCII data

Certificate file name: WMB60.arm

Location: C:\IBM\MQSI\6.0\

Figure 2-18 Extract certificate

3. Configure the broker to use SSL on a particular port.

   Many broker properties need to be set for HTTP over SSL to work with the message broker. All of these properties can be set using the following commands supplied by the HTTPS_Conf.bat script, which executes the following commands:

   a. `mqsichangeproperties WBRK6_DEFAULT_BROKER -b httplistener -o HTTPListener -n enableSSLConnector -v true`

      This command turns on SSL support in the message broker by setting the value `enableSSLConnector` to true.

   b. `mqsichangeproperties WBRK6_DEFAULT_BROKER -b httplistener -o HTTPSConnector -n keystoreFile -v "C:\IBM\MQSI\6.0\wmb_keystore.jks"`

      This command specifies the keystore to be used by setting the value `keystoreFile` to the previously created keystore: `wmb_keystore.jks`.

   c. `mqsichangeproperties WBRK6_DEFAULT_BROKER -b httplistener -o HTTPSConnector -n keystorePass -v password`

      This command specifies the password for the keystore file by setting a value for `keystorePass`. You have to replace the password in the command with the password used to create the `wmb_keystore`.

   d. `mqsichangeproperties WBRK6_DEFAULT_BROKER -b httplistener -o HTTPSConnector -n port -v 7083`

      This command specifies the port to which the message broker listens for HTTPS messages. We use port `7083` for HTTPS and port `7080` for HTTP.

   Important: You must recycle the message broker after executing these commands in order for the message broker to listen to the HTTPS messages on port `7083`. Enter the command **`netstat -an`** to verify that ports `7080` and `7083` show a LISTENING state.

4. Test the configuration of the HTTPS connection to the message broker.

   We must take a checkpoint now to make sure that the message flows and the HTTP and HTTPS connectors within the message broker are operational. You now submit the scripts to test the HTTP and HTTPS nodes. You have to follow the instruction in Step one: Test WMB configuration for HTTP and HTTPS to complete this task.

Step 6: Install the WebSphere MQ Explorer support pack IS02

This support pack allows you to configure WS-Security for DataPower using the DataPower security wizard within the WebSphere Message Broker Explorer. This wizard simplifies the configuration of the DataPower appliance security features to provide WS-Security for the message flow that uses the HTTP/HTTPS input node. You will be able to administer

WebSphere MQ V6, WebSphere Message Broker V6, and DataPower SOA appliance security from a common administrative console.

It is very easy to install this support pack. You can download the support pack and the installation instruction from the following link:

`http://www-1.ibm.com/support/docview.wss?rs=849&context=SSKM8N&dc=D400&uid=swg24012457&loc=en_US&cs=UTF-8&lang=en&rss=ct849websphere`

This support pack requires the following software:

- WebSphere Message Broker V 6.0.0.1 or later
- WebSphere MQ Explorer V6.0.2 or later
- Eclipse V3.0.2

We will use the security wizard for DataPower within the WebSphere Message Broker Explorer to configure the WS-security for message encryption and decryption features of DataPower in this section.

Important: You must set up JMS on the broker's queue manager to use the DataPower security wizard within WebSphere Message Broker Explorer. The IS02 support pack provides the instructions in "Setting up JMS for use in IS02" section.

2.3.3 Configure DataPower XML Firewall Gateway

In this section, you configure DataPower XML Firewall Gateway using the security wizard within WebSphere Message Broker Explorer. The XML firewall within DataPower is the component that processes the WS-Security for the HTTP and HTTPS protocols. WS-Security specifications were developed to address the message-level security of Web services applications. It includes authentication (user name and password validation), confidentiality (message encryption and decryption), and integrity (signed message). Without an appropriate level of security, it is not feasible for businesses to exchange messages over the open network with their clients and external business partners.

This example explains how to use the security wizard within the WebSphere Message Broker Explorer to configure the XML firewall within DataPower to provide the Web service security features. The wizard automatically retrieves information about the HTTP input and HTTPS input nodes in the message flow, creates the cryptographic profiles for SSL communications, and creates the DataPower encryption and decryption policies. The current version of the WebSphere Message Broker V6 does not support WS-Security. The DataPower appliance extends the capabilities of the message broker to meet its requirements for message encryption and decryption, digital signature, and XML firewall security to protect important corporate data.

Note: The IS02 DataPower Security wizard provides basic security configurations: XML firewall within DataPower, entire message encryption and decryption, and SSL connection. It does not allow you to configure encryption and decryption based on part of the message, digital signature, authentication, or other type of gateway, such as the Multi-Protocol Gateway within DataPower.

You must perform the initial configuration for DataPower before you can use the security wizard in the WebSphere Message Broker Explorer, which is provided by the IS02 support pack. The system administrator normally performs these tasks. However, we will explain the initial steps for setting up the DataPower appliance in case you want to do it yourself.

You will learn how to do the following tasks:

- Set up DataPower in it ally before you can use the DataPower security wizard within WebSphere Message Broker Explorer:
  - Create an application domain.
  - Create a user account and make it part of the sysadmin group.
  - Create the set of crypto key, crypto identification credential, and crypto profile to use between DataPower appliance and the message broker.
  - Create the set of crypto key, crypto identification credential, and crypto profile to use between DataPower appliance and the external clients and servers.
- Use the DataPower Security wizard within the WebSphere Message Broker Explorer to configure the Web services security for the message flows in scenario one:
  - Specify the default WS-Security policy sets.
  - Define the DataPower connection parameters.
  - Define the DataPower firewall and DataPower Policy.
  - Perform final security processing.

Currently, the security wizard within WebSphere Message Broker Explorer does not support digital signature. We will show you how to add the policy rule action to the configuration created by the security wizard to support digital signature in 2.3.4, “Configure DataPower to add support for Digital Signature” on page 57.

Initial configuration of DataPower appliance

You will learn how to perform the initial setup of DataPower before you can use the IS02 support pack to configure DataPower to provide WS-security (encryption and decryption of the entire message) for the message flows in scenario one. The initial tasks are explained in the following steps.

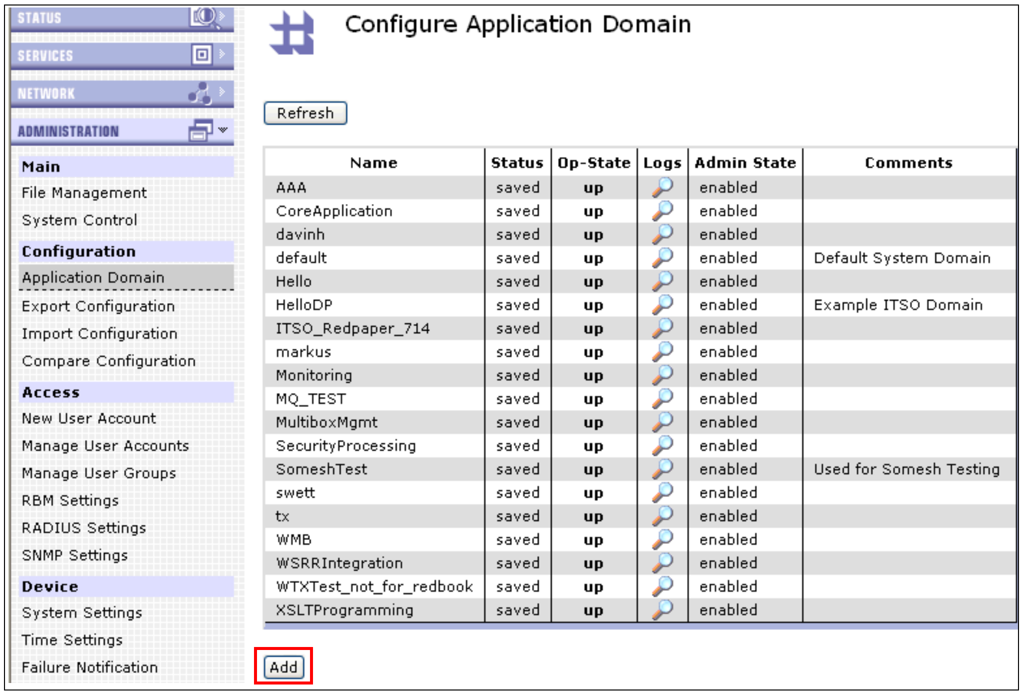
Step one: Create an application domain

DataPower allows you to create different application domains. The *default* domain is the only domain available when the system is initialized. An application domain allows you to isolate problems to your domain, restrict access to other systems, and to port the configurations among different domains.

Note: You must log on to the default domain to be able to create an application domain or a user account.

Create an application domain named `WMBIntegration`:

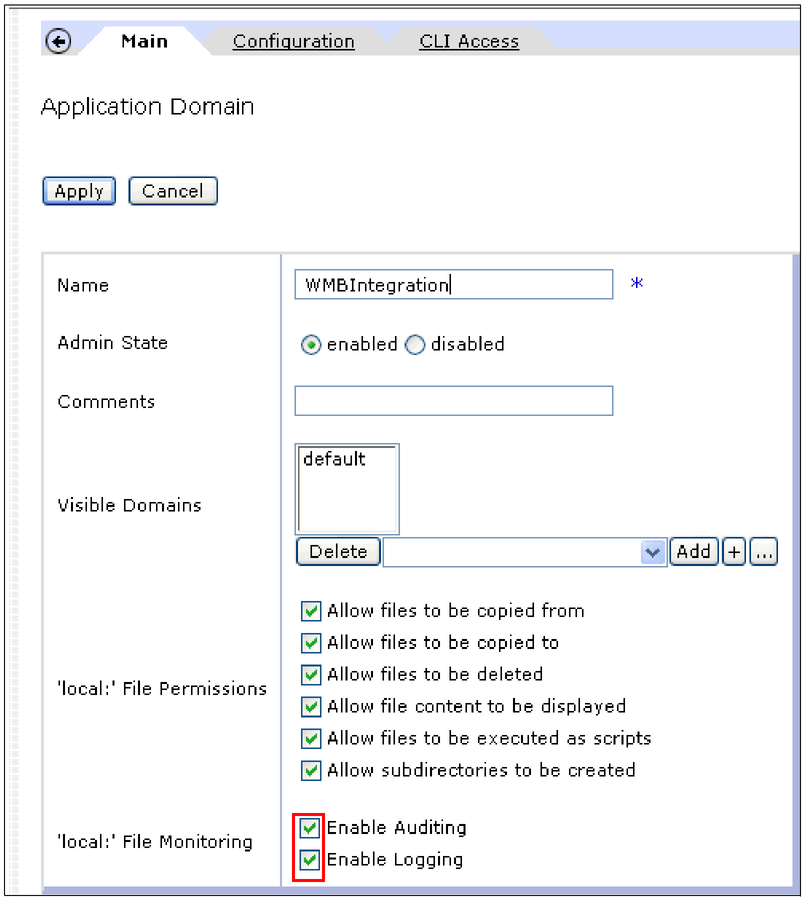
1. Log on to DataPower `default` domain.
2. Select **ADMINISTRATION** → **Configuration** → **Application Domain**. Click **Add** to display the Configure Application Domain. See Figure 2-19 on page 38.



Name	Status	Op-State	Logs	Admin State	Comments
AAA	saved	up		enabled	
CoreApplication	saved	up		enabled	
davinh	saved	up		enabled	
default	saved	up		enabled	Default System Domain
Hello	saved	up		enabled	
HelloDP	saved	up		enabled	Example ITSO Domain
ITSO_Redpaper_714	saved	up		enabled	
markus	saved	up		enabled	
Monitoring	saved	up		enabled	
MQ_TEST	saved	up		enabled	
MultiboxMgmt	saved	up		enabled	
SecurityProcessing	saved	up		enabled	
SomeshTest	saved	up		enabled	Used for Somesh Testing
swett	saved	up		enabled	
tx	saved	up		enabled	
WMB	saved	up		enabled	
WSRRIntegration	saved	up		enabled	
WTXTest_not_for_redbook	saved	up		enabled	
XSLTProgramming	saved	up		enabled	

Figure 2-19 Create a new application domain

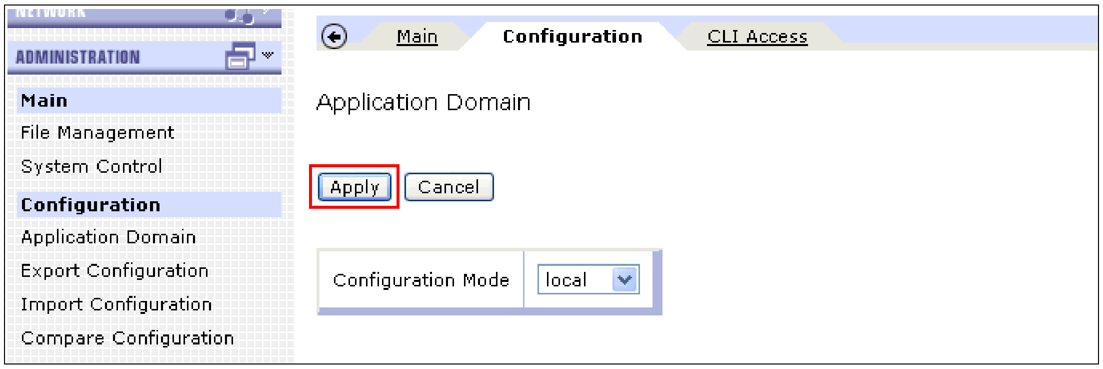
3. Enter the name of the new domain in the Name field and select the options listed in Figure 2-20 on page 39. Make sure that you select **Enable Auditing** and **Logging**. These options are not turned on by default:



Name	WMBIntegration *
Admin State	[✓ enabled] disabled
Comments	
Visible Domains	default
'local:' File Permissions	✓ Allow files to be copied from ✓ Allow files to be copied to ✓ Allow files to be deleted ✓ Allow file content to be displayed ✓ Allow files to be executed as scripts ✓ Allow subdirectories to be created
'local:' File Monitoring	✓ Enable Auditing ✓ Enable Logging

Figure 2-20 Application Domain Main tab

a. Select the **Configuration** tab. Set the Configuration Mode to **local**. See Figure 2-21.



Configuration Mode	local

Figure 2-21 Application Domain Configuration tab

b. Click **Apply** as shown in Figure 2-21 and click **Save Config** (top right) to save the new object as shown in Figure 2-22.

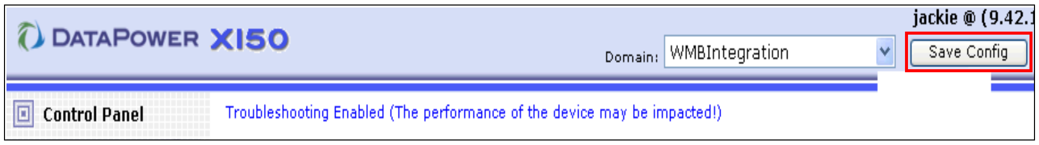
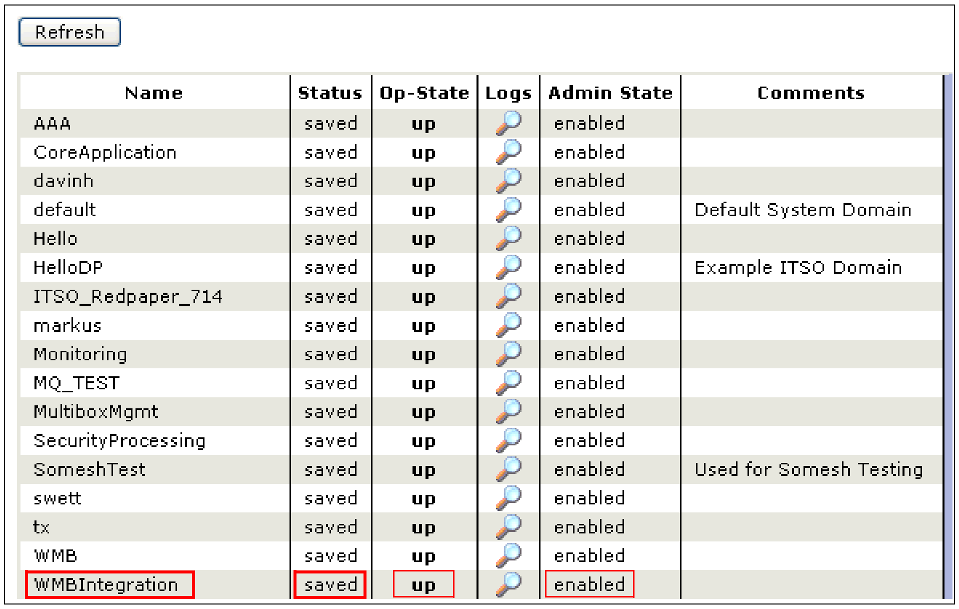


Figure 2-22 Save Config

c. Make sure that the `WMBIntegration` domain Status is `saved`, Op-State is `up`, and Admin State is `enabled` as shown in Figure 2-23.



Refresh

Name	Status	Op-State	Logs	Admin State	Comments
AAA	saved	up		enabled	
CoreApplication	saved	up		enabled	
davinh	saved	up		enabled	
default	saved	up		enabled	Default System Domain
Hello	saved	up		enabled	
HelloDP	saved	up		enabled	Example ITSO Domain
ITSO_Redpaper_714	saved	up		enabled	
markus	saved	up		enabled	
Monitoring	saved	up		enabled	
MQ_TEST	saved	up		enabled	
MultiboxMgmt	saved	up		enabled	
SecurityProcessing	saved	up		enabled	
SomeshTest	saved	up		enabled	Used for Somesh Testing
swett	saved	up		enabled	
tx	saved	up		enabled	
WMB	saved	up		enabled	
WMBIntegration	saved	up		enabled	

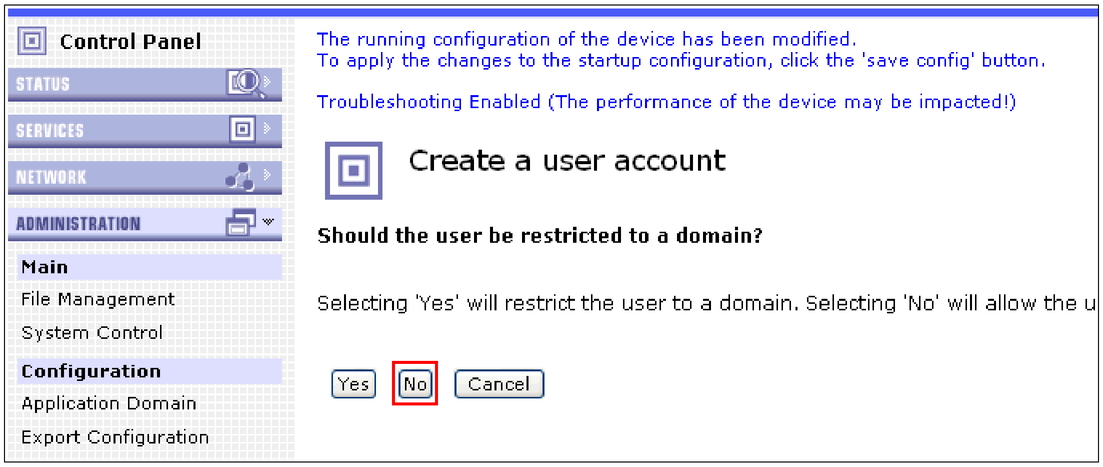
Figure 2-23 Application domain report

Step two: Create a user account

This user account will be used to administer `WMBIntegration` and communicate with WebSphere Message Broker. We are going to create a user account named `wmbadmin`. The user ID is defined in both WebSphere Message Broker and DataPower.

Follow the steps below to create a new user account:

1. Select **ADMINISTRATION** → **Access** → **New User Accounts,** select **No,** and click **Next**. See Figure 2-24.



The running configuration of the device has been modified.
To apply the changes to the startup configuration, click the 'save config' button.

Troubleshooting Enabled (The performance of the device may be impacted!)

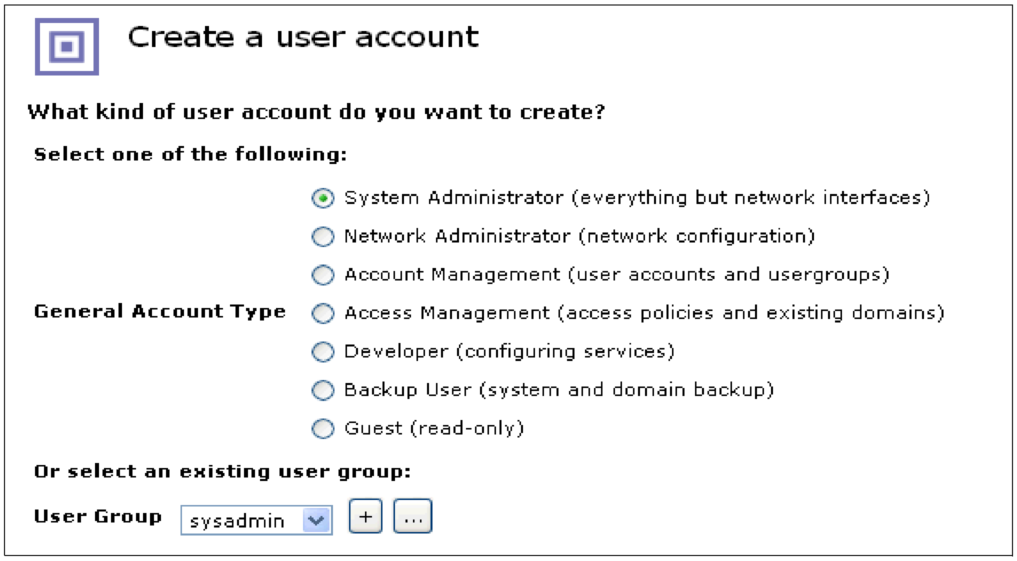
Create a user account

Should the user be restricted to a domain?

Selecting 'Yes' will restrict the user to a domain. Selecting 'No' will allow the u

Figure 2-24 Create a user account

2. Select **System Administrator**, set **User Group** to `sysadmin` as illustrated in Figure 2-25 on page 41, and click **Next**.



Create a user account

What kind of user account do you want to create?

Select one of the following:

General Account Type

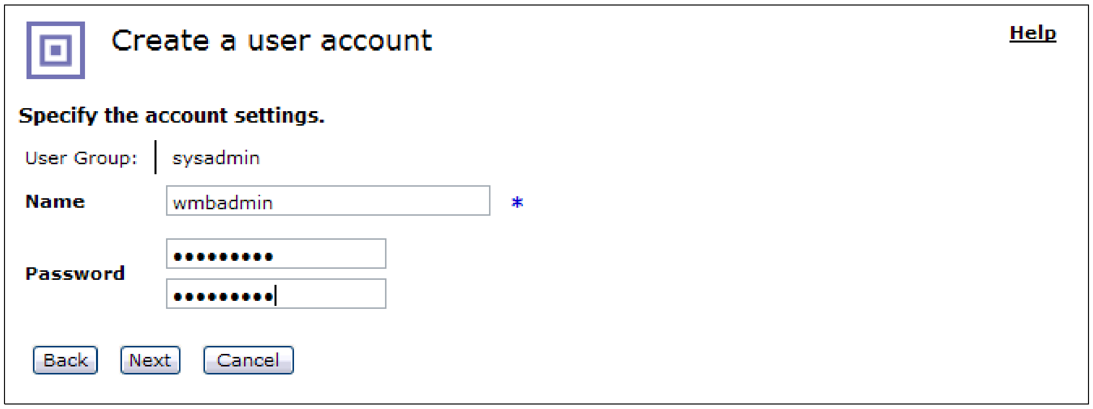
- [✓] System Administrator (everything but network interfaces)
- Network Administrator (network configuration)
- Account Management (user accounts and usergroups)
- Access Management (access policies and existing domains)
- Developer (configuring services)
- Backup User (system and domain backup)
- Guest (read-only)

Or select an existing user group:

User Group: sysadmin

Figure 2-25 Create a user account (page 1 of 2)

3. Enter the Name and password and click **Next** as shown in Figure 2-26.



Create a user account

Specify the account settings.

User Group:	sysadmin
Name	wmbadmin *
Password	•••••••••
	•••••••••

Figure 2-26 Create a user account (page 2 of 2)

4. Click **Commit**.
5. Click **Apply** and **Save Config** to save the configuration.

Note: You must switch to the `WMBIntegration` domain before performing the remaining steps.

Step three: Crypto settings for DataPower and WMB connectivity

Follow these steps:

1. Create a Crypto Key to use between DataPower and WebSphere Message Broker in the `WMBIntegration` domain.

   In the DataPower context, a *crypto key* is an object that provides an added layer of security by supplying an indirect reference (or an alias) to a file that contains a private key. The alias provided by the crypto key object is used to create a Firewall Identification Credential:

   a. Ensure that you are in the `WMBIntegration` domain.

b. Select **System Administration** → **Crypto Tools** as shown in Figure 2-27 on page 42.

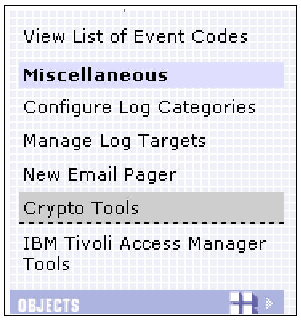
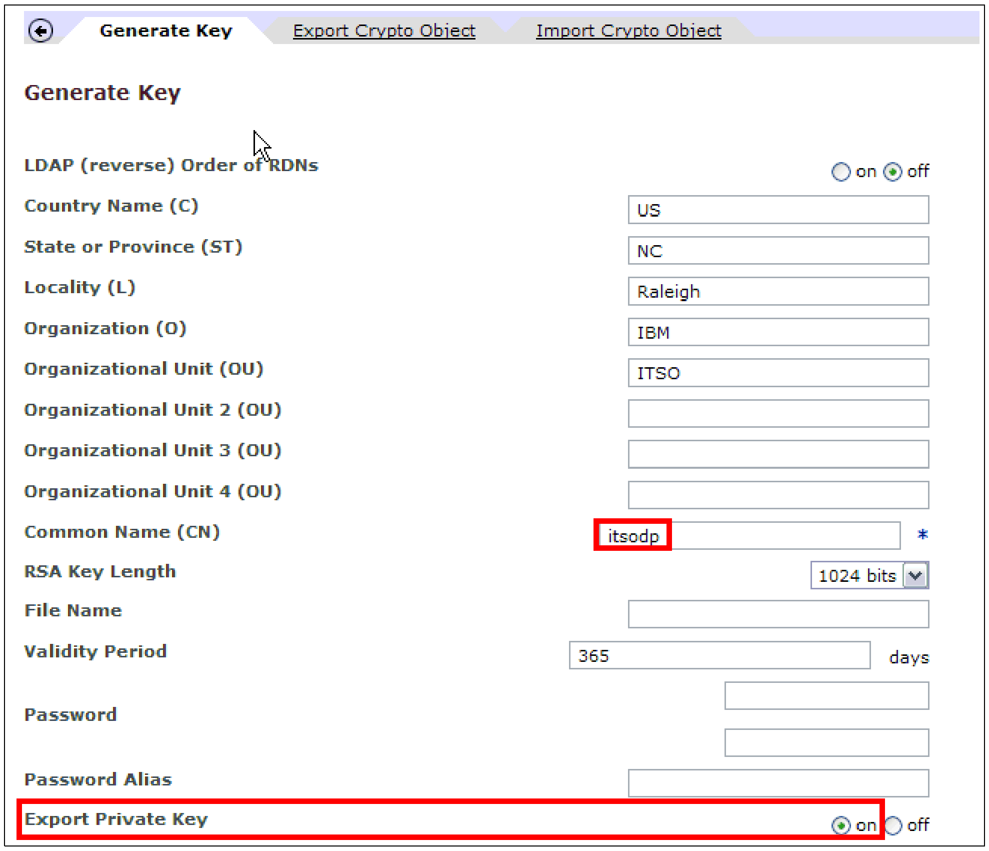


Figure 2-27 Crypto Tools

c. Enter the information: Country Name (`US`), State or Province (`NC`), Locality (`Raleigh`), Organization (`IBM`), Organizational Unit (`ITSO`), and Common Name (`itsodp`), and select **on** for Export Private Key to create the crypto key, as shown in Figure 2-28.



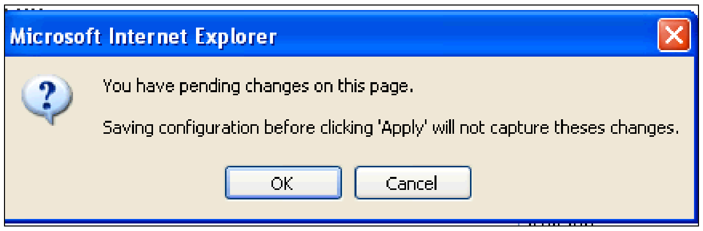
Generate Key

LDAP (reverse) Order of RDNs	on [✓ off]
Country Name (C)	US
State or Province (ST)	NC
Locality (L)	Raleigh
Organization (O)	IBM
Organizational Unit (OU)	ITSO
Organizational Unit 2 (OU)	
Organizational Unit 3 (OU)	
Organizational Unit 4 (OU)	
Common Name (CN)	itsodp *
RSA Key Length	1024 bits
File Name	
Validity Period	365 days
Password	
Password Alias	
Export Private Key	[✓ on] off

Figure 2-28 Crypto Key: itsodp

Note: By default, the certificate will expire in 365 days. You can increase or reduce the life span of the crypto key by adjusting the value of the field Validity Period.

d. Click **Generate key.** You can ignore the warning as shown in Figure 2-29 on page 43.


Microsoft Internet Explorer

You have pending changes on this page.

Saving configuration before clicking 'Apply' will not capture theses changes.

Figure 2-29 Ignore this warning

e. Click **Save Config**.

2. Create a Crypto Identification Credential.

   A crypto *Identification Credentials Set* contains a crypto key and crypto certificate. It identifies the matched public key cryptography for public and private keys used in SSL authentication. It is used to encrypt, decrypt, or sign a message:

   a. From the Control Panel, select **Keys & Certs Management**. See Figure 2-30.

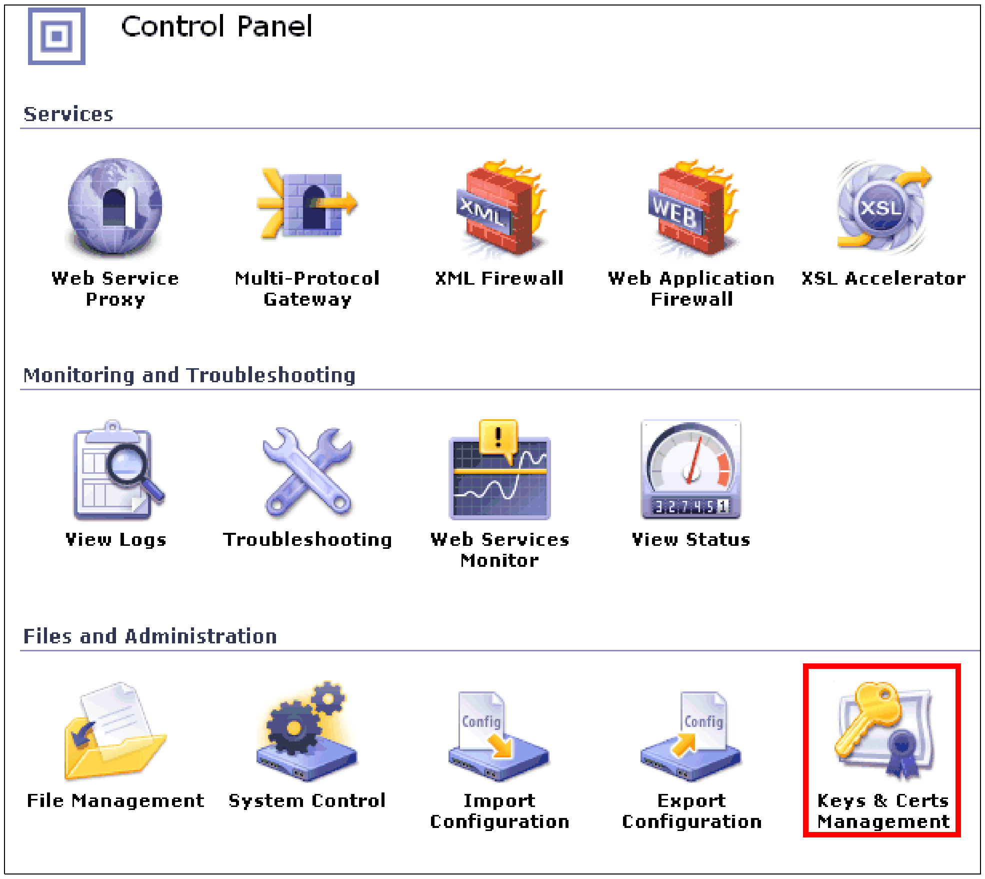
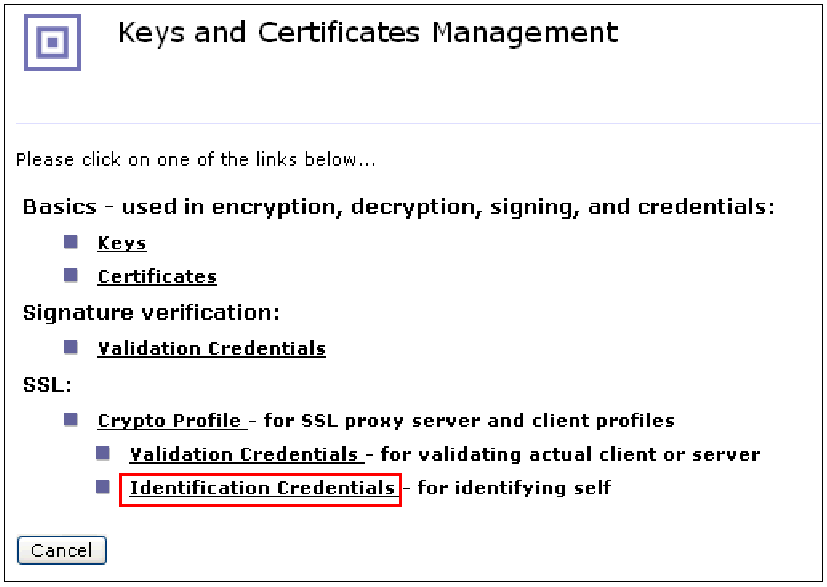


Figure 2-30 Keys & Certs Management selection

   b. Select the **Identification Credentials** link, as shown in Figure 2-31 on page 44.


Keys and Certificates Management

Please click on one of the links below...

Basics - used in encryption, decryption, signing, and credentials:

- Keys
- Certificates

Signature verification:

- Validation Credentials

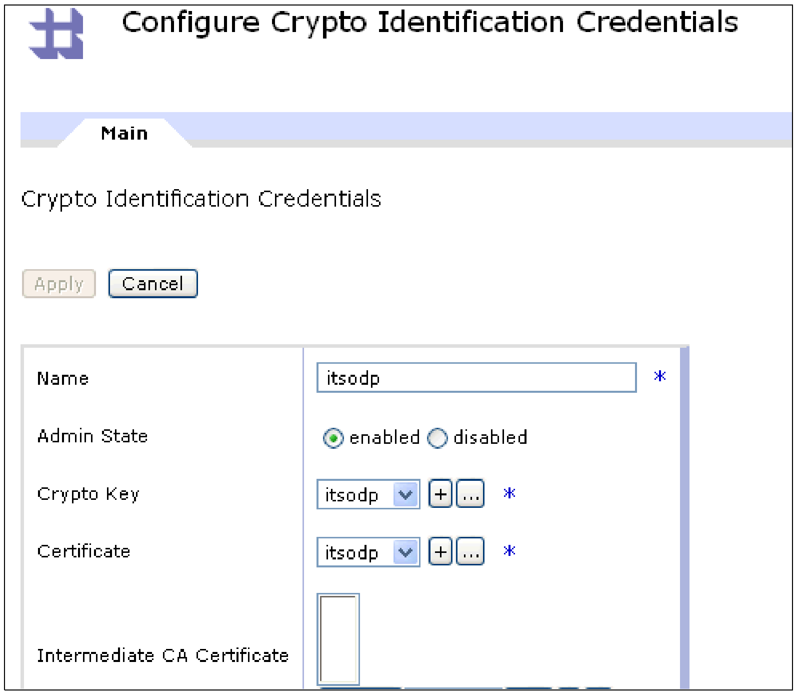
SSL:

- Crypto Profile - for SSL proxy server and client profiles
  - Validation Credentials - for validating actual client or server
  - Identification Credentials - for identifying self

Figure 2-31 Identification Credentials link

c. Click **Add.**

d. Create the identification credential using the recently created crypto key and crypto certificate. Enter the value for the fields: Name (`itsodp`), Admin State (`enabled`), Crypto key (**itsodp** - select from the drop-down list), and Certificate (**itsodp** - select from the drop-down list). See Figure 2-32.


Configure Crypto Identification Credentials

Crypto Identification Credentials

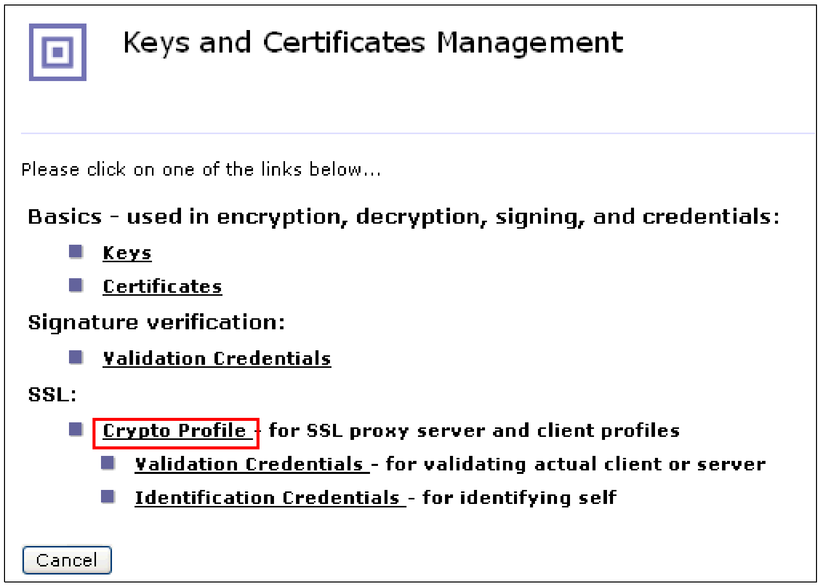
Name	itsodp *
Admin State	(● enabled) ○ disabled
Crypto Key	itsodp *
Certificate	itsodp *
Intermediate CA Certificate	

Figure 2-32 Configure Crypto Identification Credentials: itsodp

e. Click **Apply** and **Save Config** to save the configuration.

3. Create the Crypto profile called `WmbToDP` using the identification credential (`itsodp`):

a. Select the link **Crypto Profile** from the Keys and Certificates Management panel. See Figure 2-33.



Keys and Certificates Management

Please click on one of the links below...

Basics - used in encryption, decryption, signing, and credentials:

- Keys
- Certificates

Signature verification:

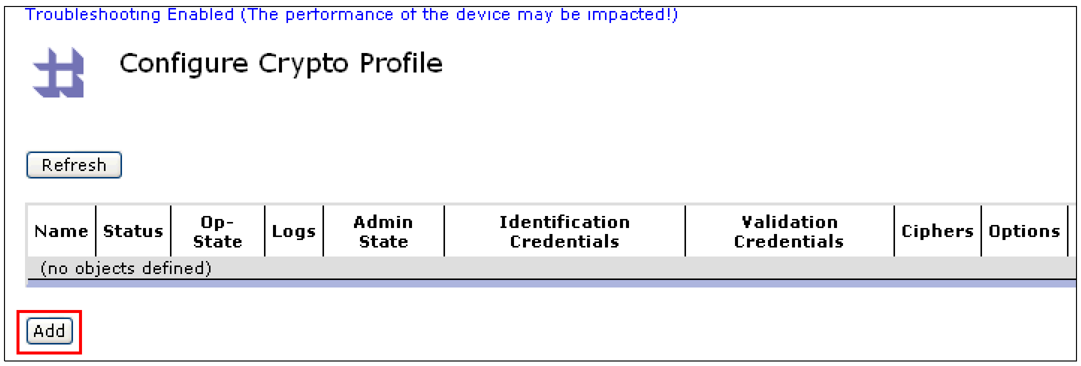
- Validation Credentials

SSL:

- Crypto Profile - for SSL proxy server and client profiles
  - Validation Credentials - for validating actual client or server
  - Identification Credentials - for identifying self

Figure 2-33 Keys and Certificates Management: Crypto Profile

b. Click **Add**, as shown in Figure 2-34.



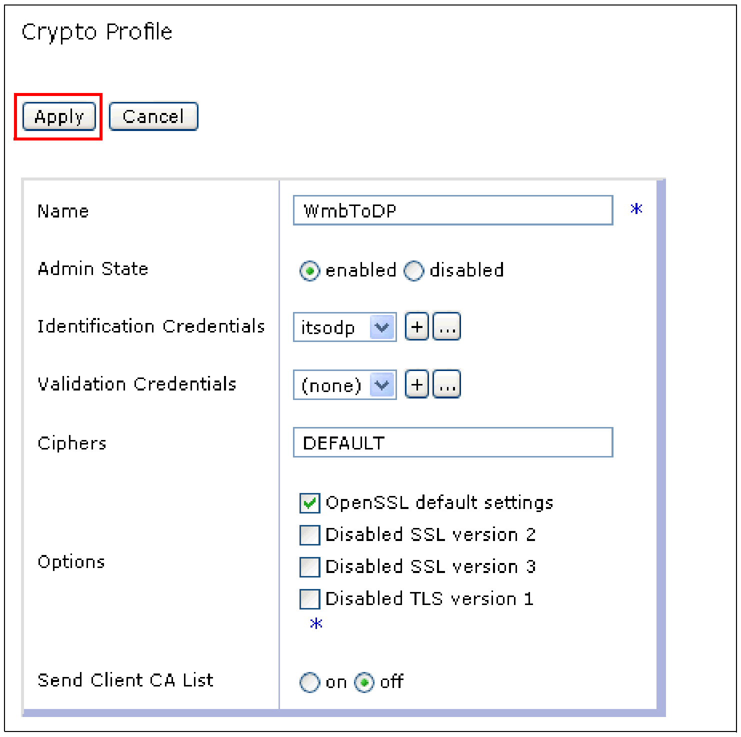
Troubleshooting Enabled (The performance of the device may be impacted!)

Configure Crypto Profile

Name	Status	Op-State	Logs	Admin State	Identification Credentials	Validation Credentials	Ciphers	Options
(no objects defined)								

Figure 2-34 Create a Crypto Profile

c. Enter the information for Name (`WmbToDP`), Admin State (click **enabled**), Identification Credentials (select **itsodp** from the drop-down list), and check **OpenSSL default settings**, as shown in Figure 2-35 on page 46.


Crypto Profile

Name	WmbToDP *
Admin State	[✓ enabled] disabled
Identification Credentials	itsodp
Validation Credentials	(none)
Ciphers	DEFAULT
Options	[✓] OpenSSL default settings [ ] Disabled SSL version 2 [ ] Disabled SSL version 3 [ ] Disabled TLS version 1 *
Send Client CA List	on [✓ off]

Figure 2-35 Crypto Profile: WmbToDP

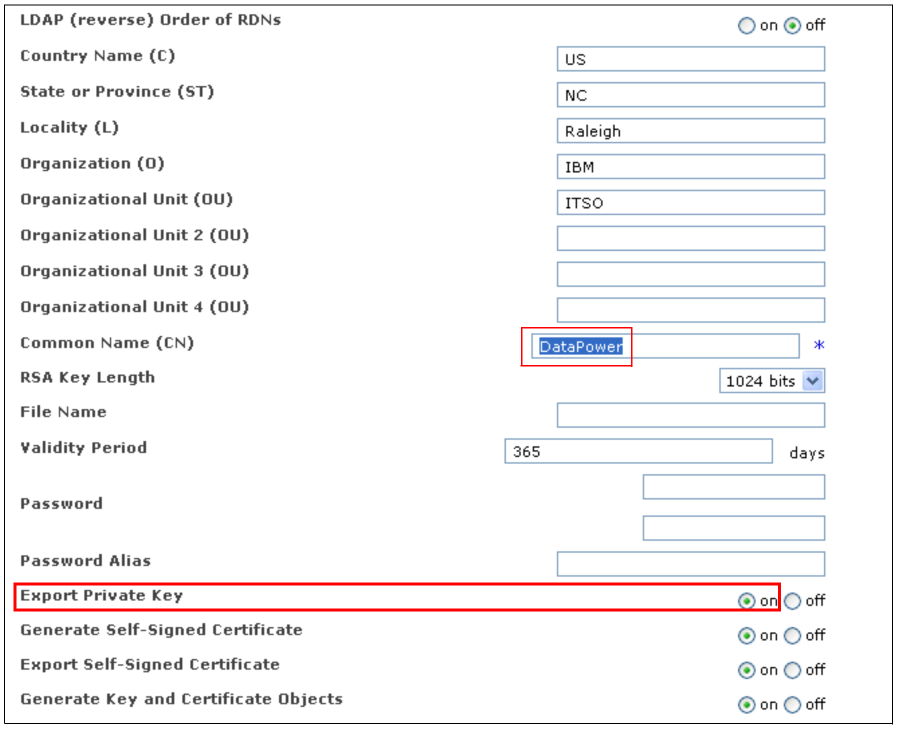
d. Click **Apply** and click **Save Config** to save the configuration.

Step four: Crypto settings for DataPower and clients connectivity

Follow these steps:

1. Repeat the previous Step 3 to create the following set of crypto key, identification credential, and security profile. The security profile is used for SSL connections from the DataPower appliance to external clients. We use the same name, `DataPower`, for the crypto key, identification credential, and crypto profile as illustrated in the next three figures:

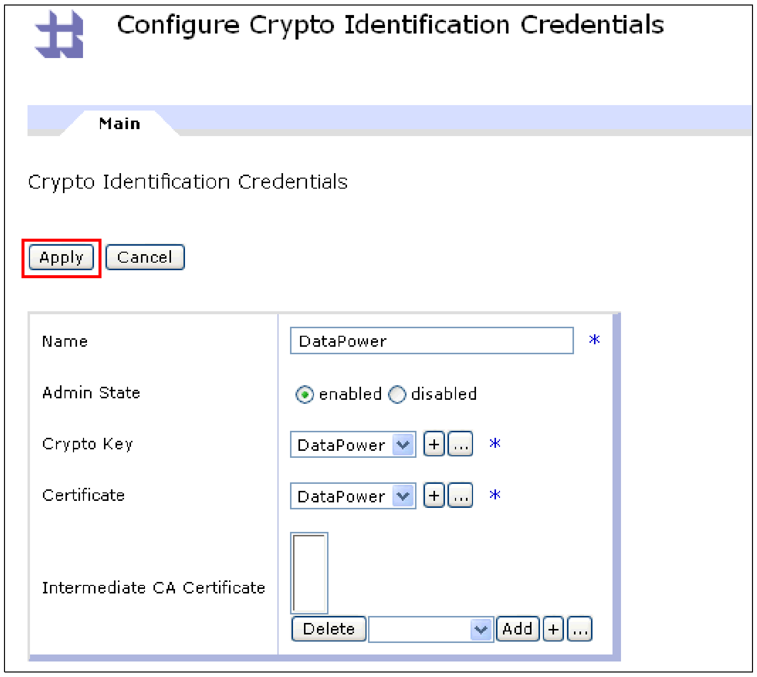
   a. Create a Crypto key with a Common Name (CN) set to `DataPower` and fill in the information as shown in Figure 2-36 on page 47.



LDAP (reverse) Order of RDNs	on [✓ off]
Country Name (C)	US
State or Province (ST)	NC
Locality (L)	Raleigh
Organization (O)	IBM
Organizational Unit (OU)	ITSO
Organizational Unit 2 (OU)	
Organizational Unit 3 (OU)	
Organizational Unit 4 (OU)	
Common Name (CN)	DataPower *
RSA Key Length	1024 bits
File Name	
Validity Period	365 days
Password	
Password Alias	
Export Private Key	[✓ on] off
Generate Self-Signed Certificate	[✓ on] off
Export Self-Signed Certificate	[✓ on] off
Generate Key and Certificate Objects	[✓ on] off

Figure 2-36 Crypto key: DataPower

b. Create an identification credential and set it to `DataPower` with the information as shown in Figure 2-37 and click **Apply**.



Configure Crypto Identification Credentials

Main

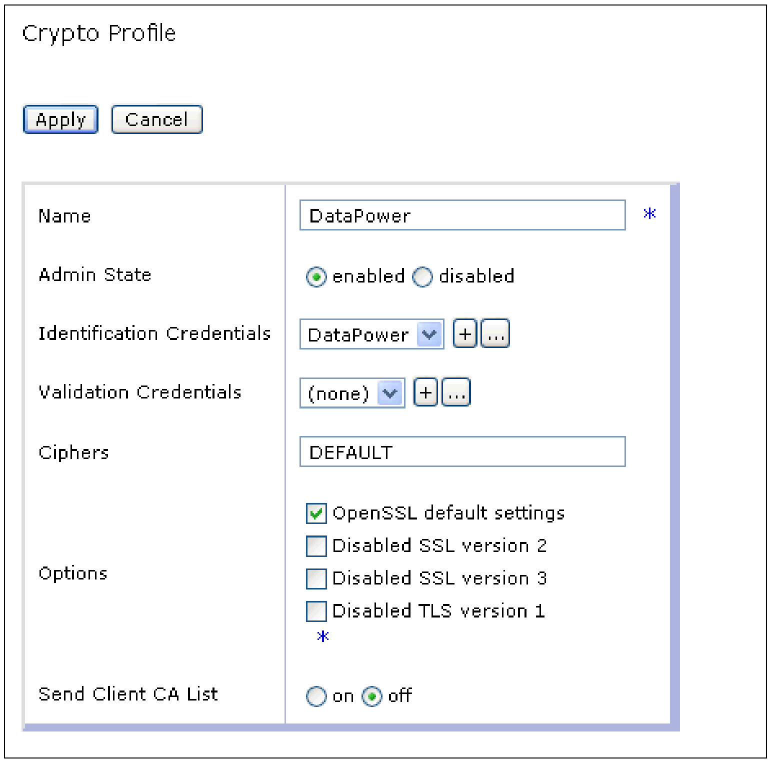
Crypto Identification Credentials

Apply Cancel

Name	DataPower *
Admin State	[✓ enabled] disabled
Crypto Key	DataPower *
Certificate	DataPower *
Intermediate CA Certificate	Delete Add

Figure 2-37 Crypto Identification Credentials: DataPower

c. Create a crypto profile and set it to `DataPower`. Fill in the information as shown in Figure 2-39 on page 48 and click **Apply** and **Save Config** to complete.



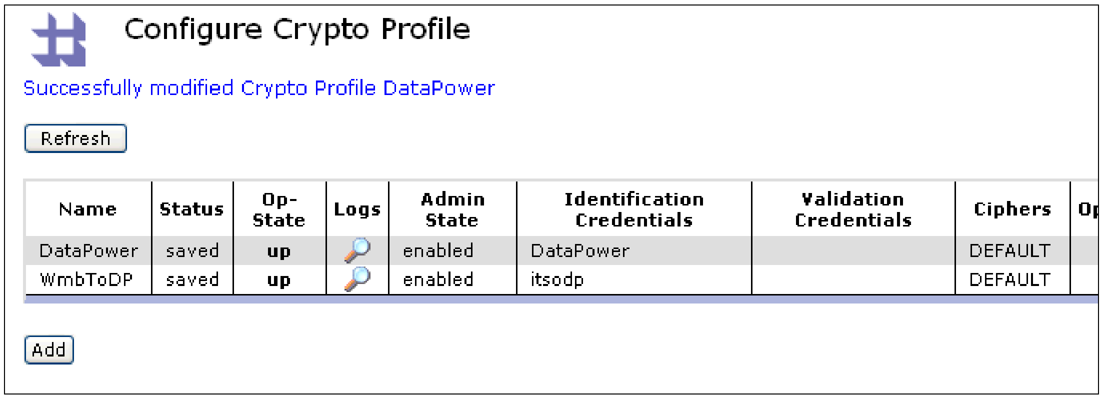
Crypto Profile

Name	DataPower *
Admin State	[✓ enabled] disabled
Identification Credentials	DataPower
Validation Credentials	(none)
Ciphers	DEFAULT
Options	[✓] OpenSSL default settings ☐ Disabled SSL version 2 ☐ Disabled SSL version 3 ☐ Disabled TLS version 1 *
Send Client CA List	on [✓ off]

Figure 2-38 Crypto Profile: DataPower

d. Back to the Configure Crypto Profile panel: You see that the DataPower and WmbToDP profiles were successfully saved with Op-State listed as `up` and Admin State listed as `enabled` as shown in Figure 2-39.

Note: You must remember to click **Apply** and **Save Config** whenever applicable.



Configure Crypto Profile

Successfully modified Crypto Profile DataPower

Name	Status	Op-State	Logs	Admin State	Identification Credentials	Validation Credentials	Ciphers	Op
DataPower	saved	up		enabled	DataPower		DEFAULT	
WmbToDP	saved	up		enabled	itsodp		DEFAULT	

Figure 2-39 Configure Crypto Profile report

Configure DataPower security for WMB message flows and HTTP(S)

To perform the tasks in this section, you must have deployed and tested the message flows for scenario one to make sure that they are operational. We cannot use the DataPower security wizard if the message flows are not deployed and the IS02 support pack is not installed. We have provided a Broker Archive (bar) file called `SOA2Legacy.bar` for your testing. You can also create the SOA2Legacy.bar file yourself if you choose to build the message broker artifacts yourself by following the instructions documented in Appendix B, “Building message flows in WebSphere Message Broker” on page 119.

Configuring the DataPower appliance using this wizard has four major steps:

- Select the HTTP and HTTPS input nodes to configure your security.
- Define the DataPower connection profile.
- Define the DataPower firewall and policies.
- Specify which specific crypto keys to use from the DataPower appliance.

You will learn how to use the DataPower Security wizard to configure the DataPower XML Firewall with WS-Security to provide front-end security for the message flows that contain the HTTP and HTTPS input nodes. You must first deploy the message flows to a message broker before you can configure the security for DataPower using the security wizard:

1. Invoke the security wizard for DataPower from within the WebSphere Message Broker Explorer:

   a. Start MQ Explorer from Windows: **Start** → **Programs** → **IBM WebSphere MQ** → **WebSphere MQ Explorer.**

   b. Select the deployed message flow **WS_To_Legacy_MF**, right-click the flow, select **DataPower**, and select **Security Wizard.**

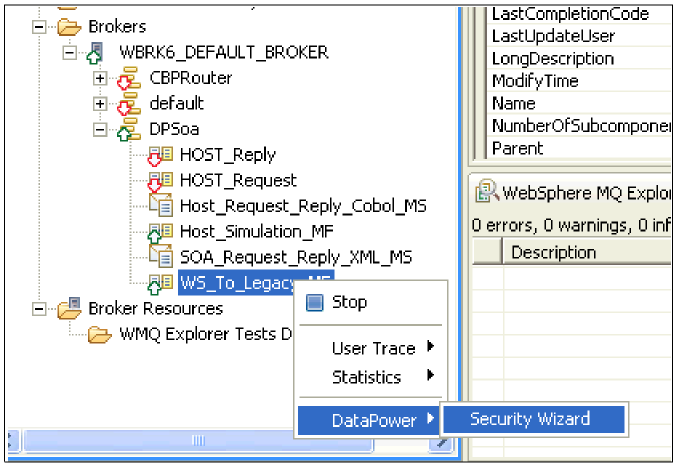


Figure 2-40 DataPower Security Wizard

Note: If you do not see the menu for DataPower, select **Windows** → **Preferences** → **Broker Explorer** → **DataPower** and make sure that the Display DataPower menu box is checked in the Broker Explorer.

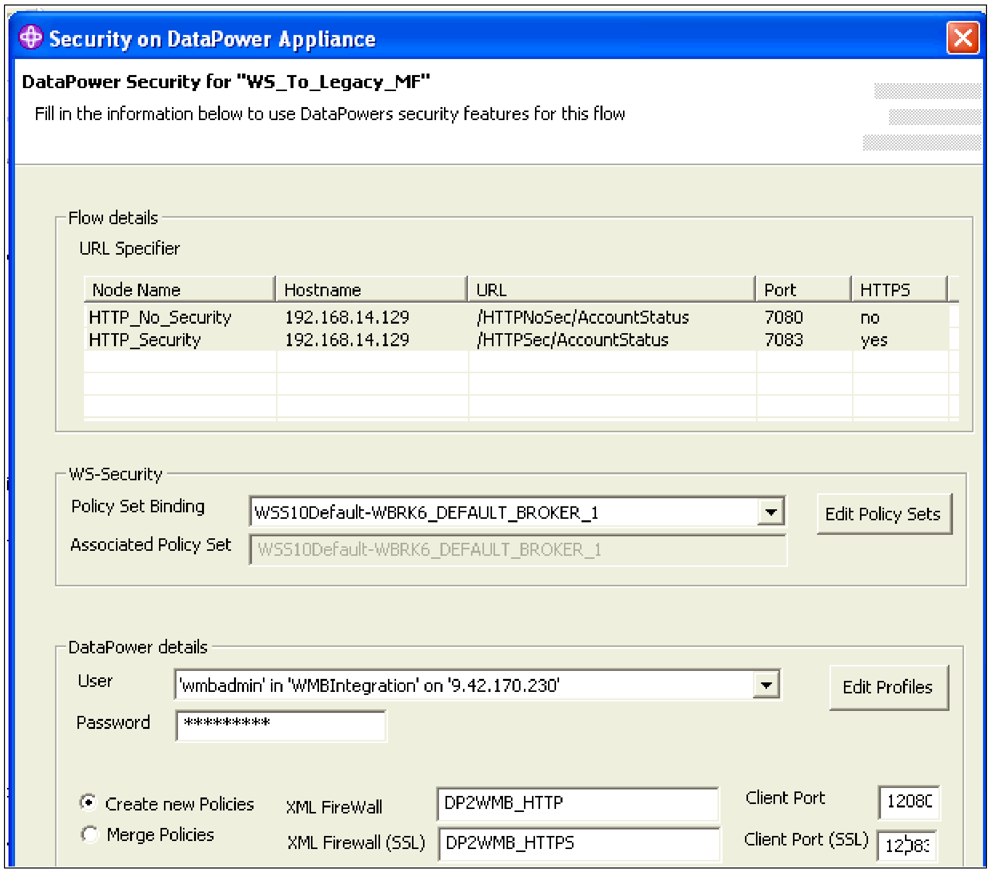
   c. Fill in the information as displayed in Figure 2-41 on page 50:

      i. Select the URLs discovered by the wizard under the section Flow details.

      The `WS_To_Legacy_MF` message flow contains an HTTPInput node and an HTTPSInput node. The wizard selects all HTTP(S) Input nodes by default. The URLs are specified in the properties of the HTTPInput and HTTPSInput nodes. They specify the location from where the message broker retrieves the Web service requests.

      ii. Select the default security policy named `WSS10Default-WBRK6_DEFAULT_BROKER_1`. This default security policy is prefilled for you. The WS-Security section shows

`Policy Set Binding` and `Associated Policy Set`. Their names show the relationship between the Policy Setting and the message broker. You cannot change this naming convention, but you can edit the Policy Set Binding and its associated Policy Set parameters. The Policy Set Binding contains information about the encryption and decryption key.



Security on DataPower Appliance

DataPower Security for "WS_To_Legacy_MF"

Fill in the information below to use DataPowers security features for this flow

Flow details

URL Specifier

Node Name	Hostname	URL	Port	HTTPS
HTTP_No_Security	192.168.14.129	/HTTPNoSec/AccountStatus	7080	no
HTTP_Security	192.168.14.129	/HTTPSec/AccountStatus	7083	yes

WS-Security

Policy Set Binding: WSS10Default-WBRK6_DEFAULT_BROKER_1

Associated Policy Set: WSS10Default-WBRK6_DEFAULT_BROKER_1

DataPower details

User: 'wmbadmin' in 'WMBIntegration' on '9.42.170.230'

Password: **********

[✓ Create new Policies] Merge Policies

XML FireWall: DP2WMB_HTTP

XML Firewall (SSL): DP2WMB_HTTPS

Client Port: 1208C

Client Port (SSL): 12083

Figure 2-41 DataPower Security wizard configuration

iii. Click **Edit Profiles** and add the connection information, such as the User name (`wmbadmin`), domain (`WMBIntegration`), IP address of the DataPower box, and the Mgmt Port (`5550` is the default Mgmt Port set when you create the application domain). This information is in "Initial configuration of DataPower appliance" on page 37.

iv. Enter the password for the user ID `wmbadmin`.

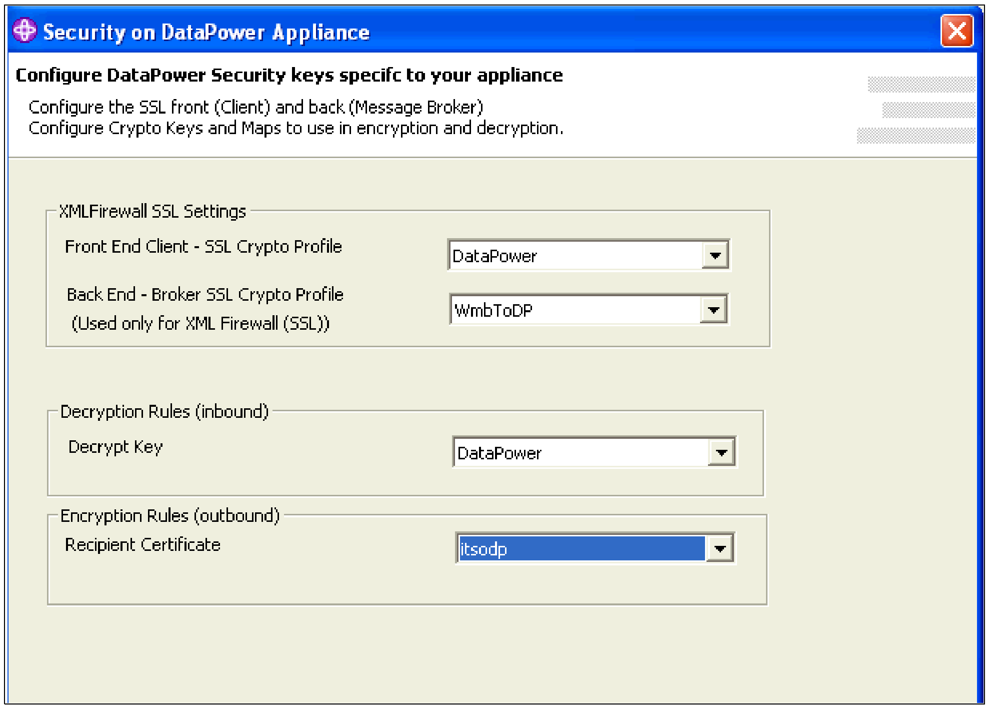
v. Enter the name of security/processing policy used by the firewall to enforce WS-security between the DataPower appliance and the message broker. The wizard will create two XML firewalls for the HTTP and HTTPS input nodes and the associated security policy. We have selected the names `DP2WMB_HTTP` and `DP2WMB_HTTPS`.

vi. Enter the Client Port for the DataPower appliance. By default, the wizard uses the same port number on which the broker listens. We have selected port `12080` for HTTP and port `12083` for HTTPS.

vii. Click **Next.**

viii.In Figure 2-42 on page 51, enter the encryption and decryption keys and security profiles that will be used by DataPower to enforce WS-security between DataPower and the message broker, and between DataPower and its clients. The **Front End**

Client - SSL Crypto Profile is used to configure the front-end communication profile between the clients and DataPower. **The Back End - Broker SSL Crypto Profile** is used to configure the communication between DataPower and the message broker.


Security on DataPower Appliance

Configure DataPower Security keys specifc to your appliance

Configure the SSL front (Client) and back (Message Broker)
Configure Crypto Keys and Maps to use in encryption and decryption.

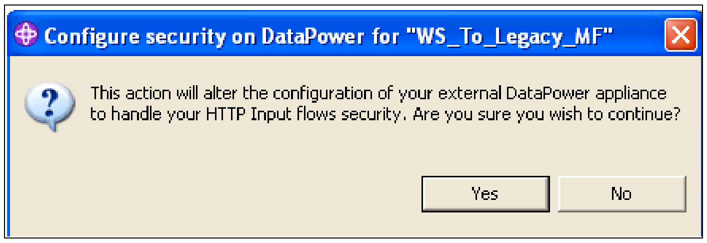
XMLFirewall SSL Settings
Front End Client - SSL Crypto Profile: DataPower
Back End - Broker SSL Crypto Profile (Used only for XML Firewall (SSL)): WmbToDP

Decryption Rules (inbound)
Decrypt Key: DataPower

Encryption Rules (outbound)
Recipient Certificate: itsodp

Figure 2-42 DataPower security wizard cryptographic Settings

ix. Click **Finish** and then click **Yes** to the question asked in Figure 2-43.

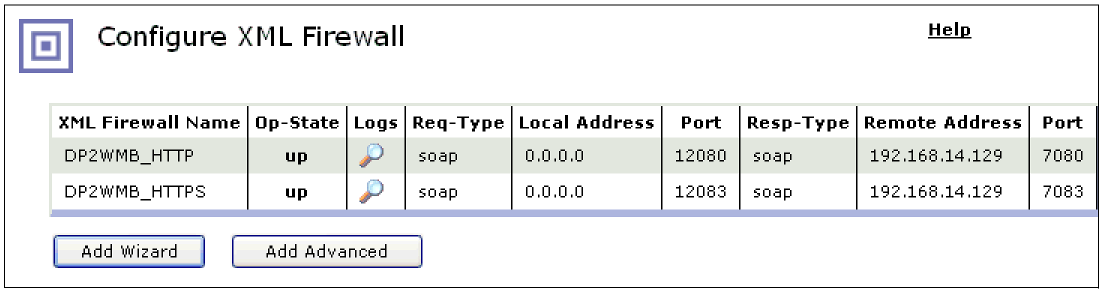

Configure security on DataPower for "WS_To_Legacy_MF"

This action will alter the configuration of your external DataPower appliance to handle your HTTP Input flows security. Are you sure you wish to continue?

Figure 2-43 Pop-up menu to confirm changes

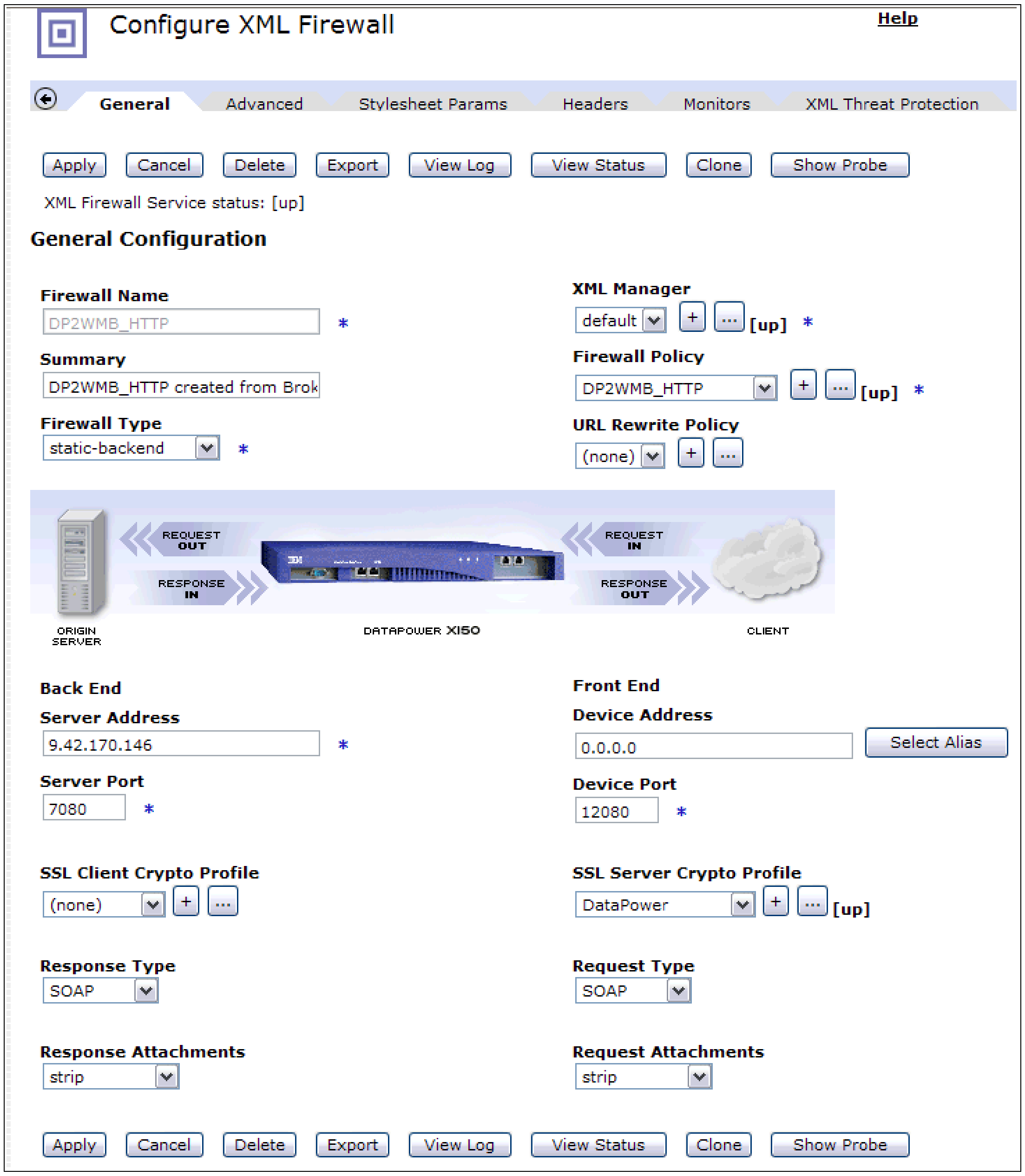
We have just configured DataPower and the message broker as shown in Figure 2-1 on page 19. The security wizard makes it easy to configure the XML Firewall and WS-Security within the DataPower appliance; see Figure 2-45 on page 52 (HTTP) and Figure 2-46 on page 53 (HTTPS). When the configuration is complete, the security wizard creates the following components. Follow these steps:

1. Two DataPower XML firewalls: One for the HTTPInput node (`DP2WMB_HTTP`) and another for the HTTPS input node (`DP2WMB_HTTPS`) as shown in Figure 2-44 on page 52.


Configure XML Firewall

XML Firewall Name	Op-State	Logs	Req-Type	Local Address	Port	Resp-Type	Remote Address	Port
DP2WMB_HTTP	up		soap	0.0.0.0	12080	soap	192.168.14.129	7080
DP2WMB_HTTPS	up		soap	0.0.0.0	12083	soap	192.168.14.129	7083

Figure 2-44   DataPower XML firewalls created by security wizard


Configure XML Firewall

XML Firewall Service status: [up]

General Configuration

Firewall Name	DP2WMB_HTTP *
Summary	DP2WMB_HTTP created from Brok
Firewall Type	static-backend *
XML Manager	default [up] *
Firewall Policy	DP2WMB_HTTP [up] *
URL Rewrite Policy	(none)

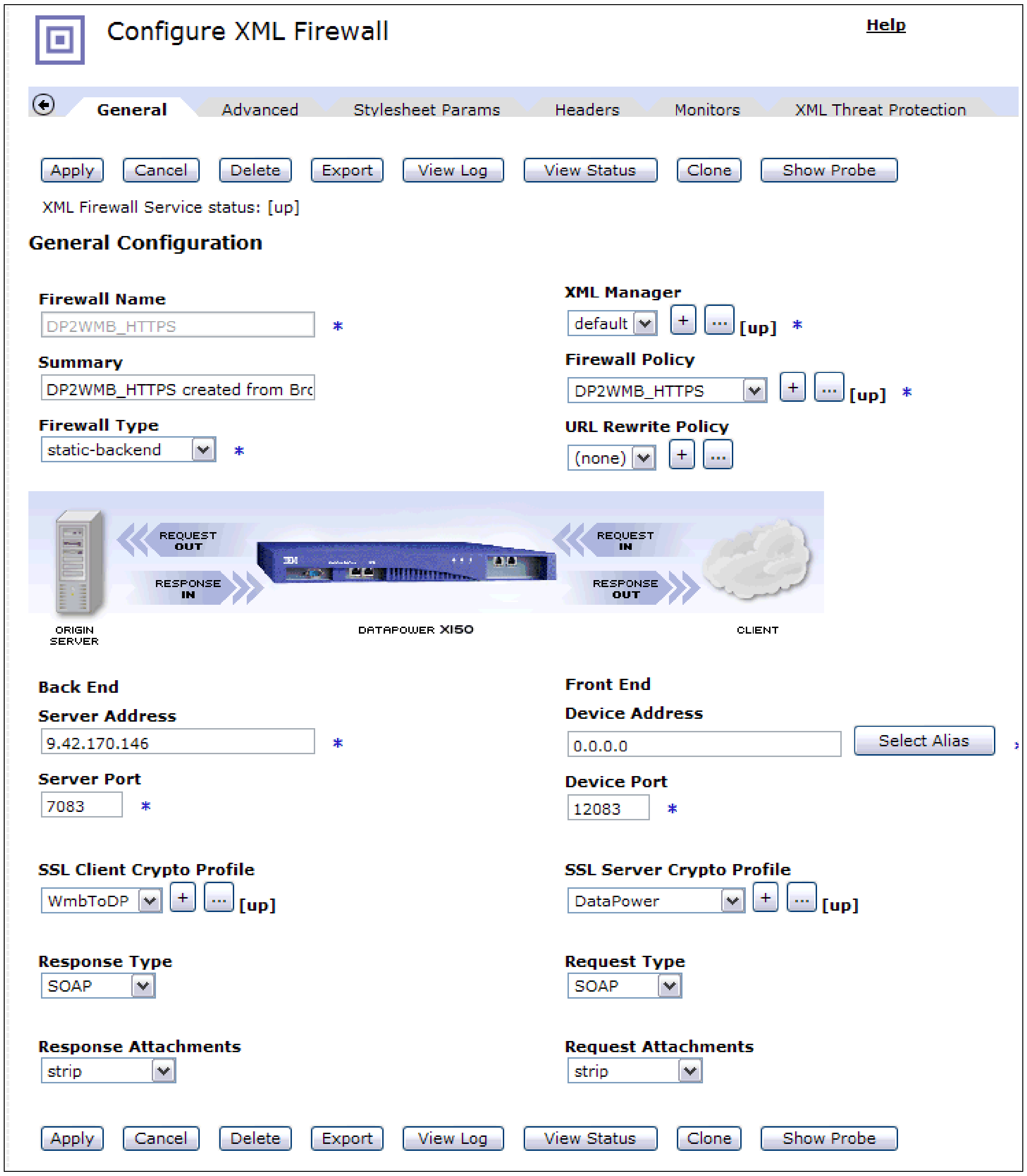
Back End

Server Address	9.42.170.146 *
Server Port	7080 *
SSL Client Crypto Profile	(none)
Response Type	SOAP
Response Attachments	strip

Front End

Device Address	0.0.0.0
Device Port	12080 *
SSL Server Crypto Profile	DataPower [up]
Request Type	SOAP
Request Attachments	strip

Figure 2-45   XML firewall detailed configuration for HTTP


Configure XML Firewall

Help

General | Advanced | Stylesheet Params | Headers | Monitors | XML Threat Protection

Apply | Cancel | Delete | Export | View Log | View Status | Clone | Show Probe

XML Firewall Service status: [up]

General Configuration

Firewall Name: DP2WMB_HTTPS *

Summary: DP2WMB_HTTPS created from Brc

Firewall Type: static-backend *

XML Manager: default [up] *

Firewall Policy: DP2WMB_HTTPS [up] *

URL Rewrite Policy: (none)

Back End

Server Address: 9.42.170.146 *

Server Port: 7083 *

Front End

Device Address: 0.0.0.0 Select Alias

Device Port: 12083 *

SSL Client Crypto Profile: WmbToDP [up]

SSL Server Crypto Profile: DataPower [up]

Response Type: SOAP

Request Type: SOAP

Response Attachments: strip

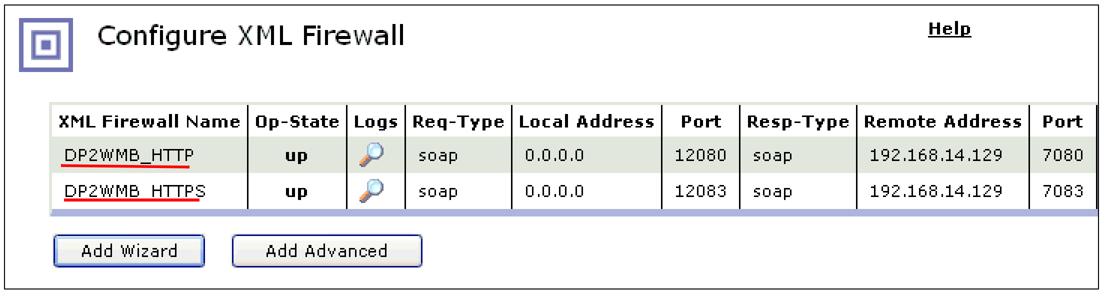
Request Attachments: strip

Apply | Cancel | Delete | Export | View Log | View Status | Clone | Show Probe

Figure 2-46 XML firewall detailed configuration for HTTPS

2. A DataPower security/processing policy for the XML firewalls:

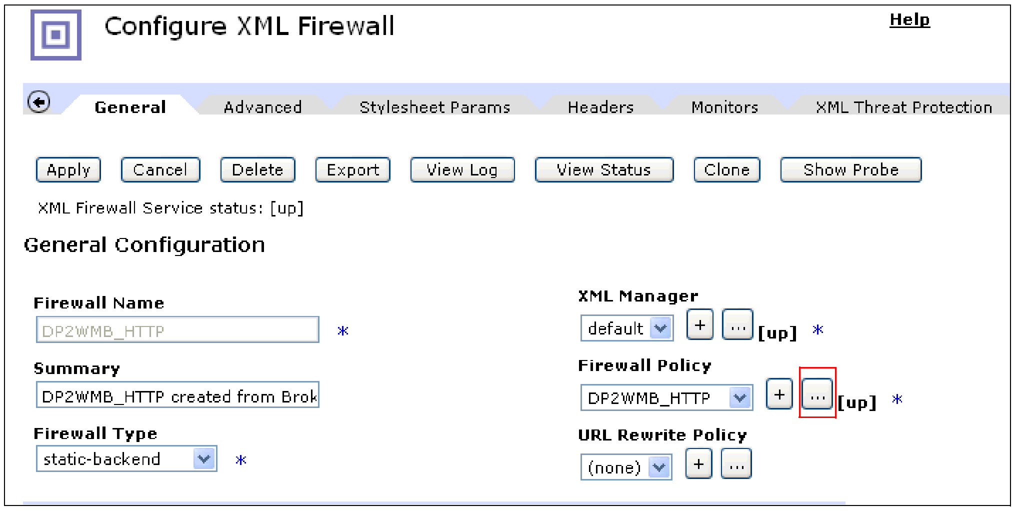
   a. Click the `DP2WMB_HTTP` or `DP2WMB_HTTPS` link to access the DataPower XML Firewall Gateway for each XML firewall as shown in Figure 2-47 on page 54.


Configure XML Firewall

XML Firewall Name	Op-State	Logs	Req-Type	Local Address	Port	Resp-Type	Remote Address	Port
DP2WMB_HTTP	up	🔍	soap	0.0.0.0	12080	soap	192.168.14.129	7080
DP2WMB_HTTPS	up	🔍	soap	0.0.0.0	12083	soap	192.168.14.129	7083

Figure 2-47 XML firewall links

b. Double-click the ellipsis (**...**) on the right side of the DP2WMB_HTTP Firewall Policy as shown in Figure 2-48 to view the DP2WMB_HTTP Firewall Policy. It has a pair of processing rules: request and response rules are listed under the Configured Rules section as shown in Figure 2-49 on page 55.


Configure XML Firewall

XML Firewall Service status: [up]

General Configuration

Firewall Name: DP2WMB_HTTP *

Summary: DP2WMB_HTTP created from Brok

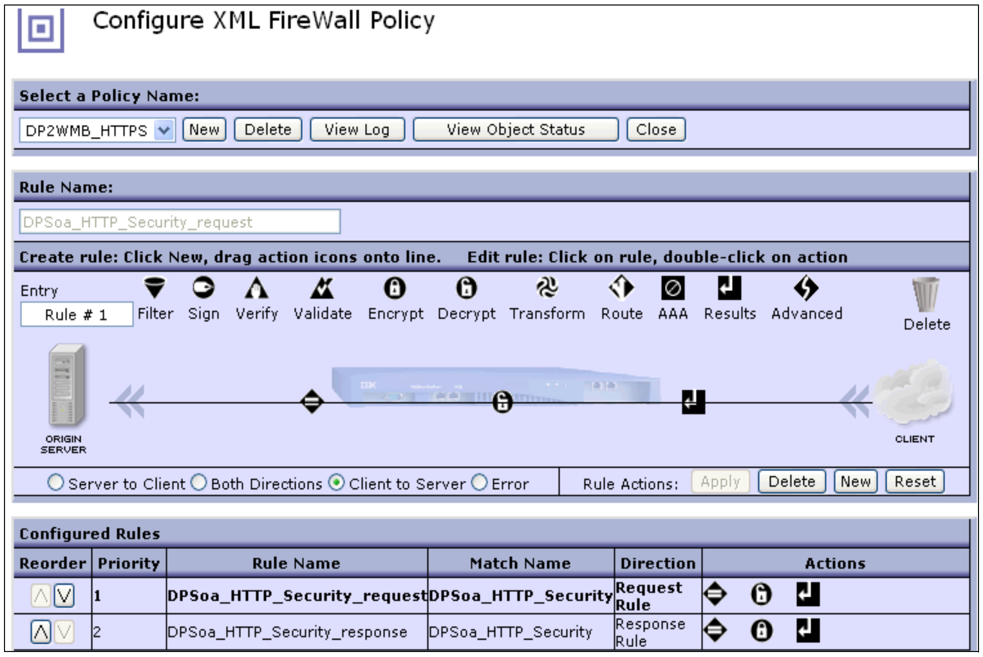
Firewall Type: static-backend *

XML Manager: default [up] *

Firewall Policy: DP2WMB_HTTP [up] *

URL Rewrite Policy: (none)

Figure 2-48 XML firewall processing policy: DP2WMB_HTTP


Configure XML FireWall Policy

Select a Policy Name: DP2WMB_HTTPS

Rule Name: DPSoa_HTTP_Security_request

Create rule: Click New, drag action icons onto line. Edit rule: Click on rule, double-click on action

Entry: Rule # 1

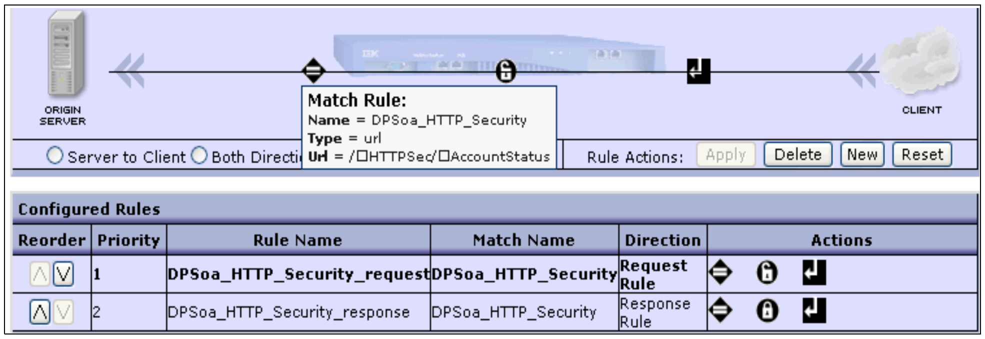
Server to Client / Both Directions / [✓ Client to Server] / Error

Configured Rules

Reorder	Priority	Rule Name	Match Name	Direction	Actions
	1	**DPSoa_HTTP_Security_request**	**DPSoa_HTTP_Security**	**Request Rule**	
	2	DPSoa_HTTP_Security_response	DPSoa_HTTP_Security	Response Rule	

Figure 2-49   XML firewall policy request rule

c. On the inbound request path, the URL specified in the Match Rule for the request message must match the URL specified in the property of the HTTP node. See Figure 2-50.

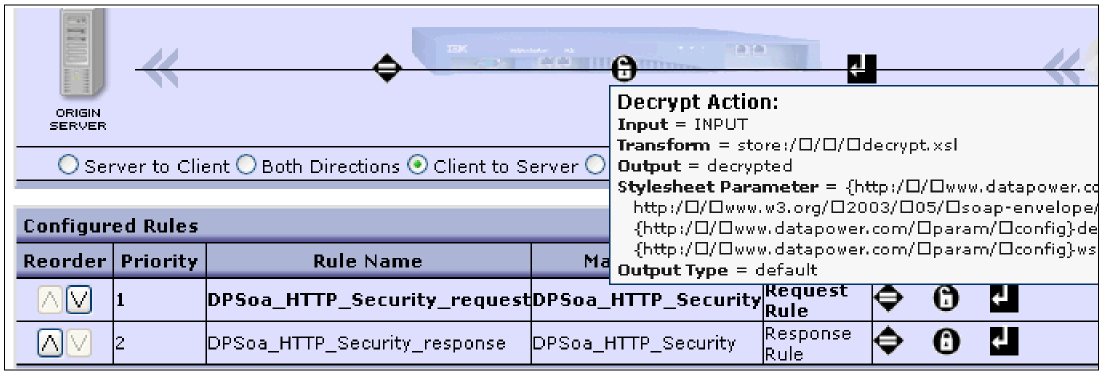

Match Rule:
Name = DPSoa_HTTP_Security
Type = url
Url = /□HTTPSec/□AccountStatus

Configured Rules

Reorder	Priority	Rule Name	Match Name	Direction	Actions
	1	**DPSoa_HTTP_Security_request**	**DPSoa_HTTP_Security**	**Request Rule**	
	2	DPSoa_HTTP_Security_response	DPSoa_HTTP_Security	Response Rule	

Figure 2-50   XML firewall policy request rule: Match Rule

d. The processing rule for the input message is configured to decrypt incoming traffic. The Decrypt Action uses the DataPower crypto key, which was created in the initial configuration of DataPower, to decrypt the message as shown in Figure 2-51 on page 56.



○ Server to Client ○ Both Directions [✓ Client to Server] ○

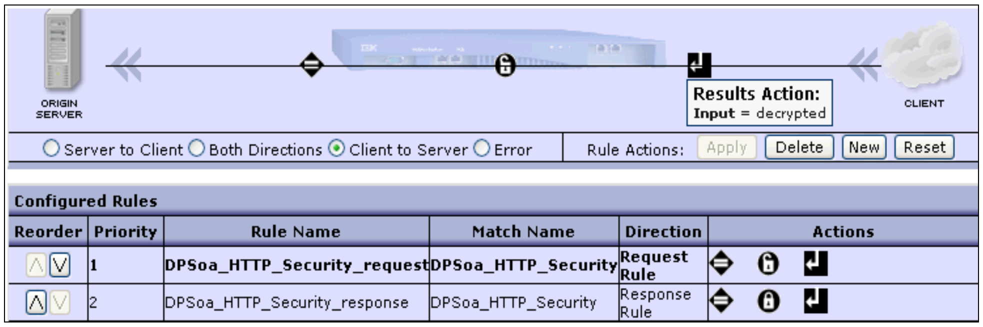
Decrypt Action:
Input = INPUT
Transform = store:/□/□/□decrypt.xsl
Output = decrypted
Stylesheet Parameter = {http:/□/□www.datapower.c
http:/□/□www.w3.org/□2003/□05/□soap-envelope/
{http:/□/□www.datapower.com/□param/□config}de
{http:/□/□www.datapower.com/□param/□config}ws
Output Type = default

Configured Rules

Reorder	Priority	Rule Name	Ma		
∧ ∨	1	**DPSoa_HTTP_Security_request**	**DPSoa_HTTP_Security**	**Request Rule**	
∧ ∨	2	DPSoa_HTTP_Security_response	DPSoa_HTTP_Security	Response Rule	

Figure 2-51 XML firewall policy request rule: Decrypt Action

e. The Results Action is a decrypted message as shown in Figure 2-52.



Results Action:
Input = decrypted

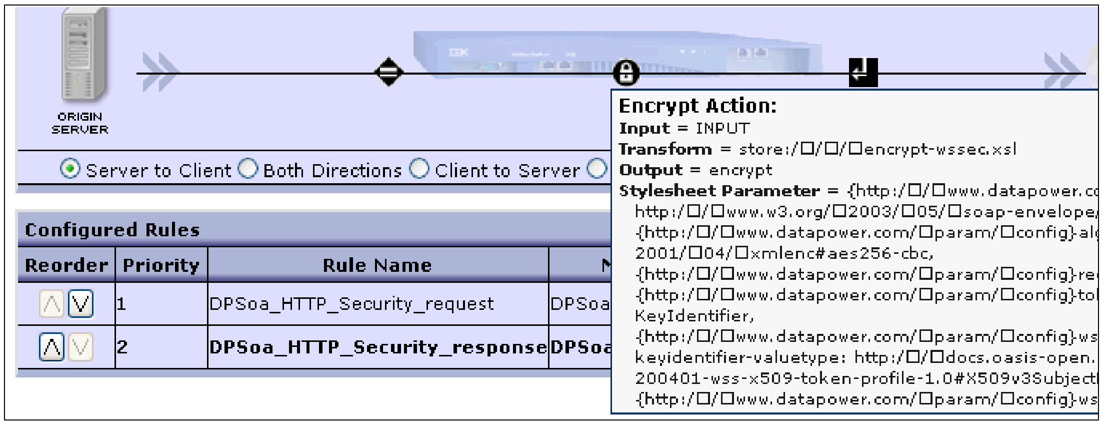
○ Server to Client ○ Both Directions [✓ Client to Server] ○ Error Rule Actions: Apply Delete New Reset

Configured Rules

Reorder	Priority	Rule Name	Match Name	Direction	Actions
∧ ∨	1	**DPSoa_HTTP_Security_request**	**DPSoa_HTTP_Security**	**Request Rule**	
∧ ∨	2	DPSoa_HTTP_Security_response	DPSoa_HTTP_Security	Response Rule	

Figure 2-52 XML firewall policy request rule: Results Action

f. On the outbound traffic, the security rule is configured to encrypt the message from the message broker before sending it to a client as shown in Figure 2-53.



[✓ Server to Client] ○ Both Directions ○ Client to Server ○

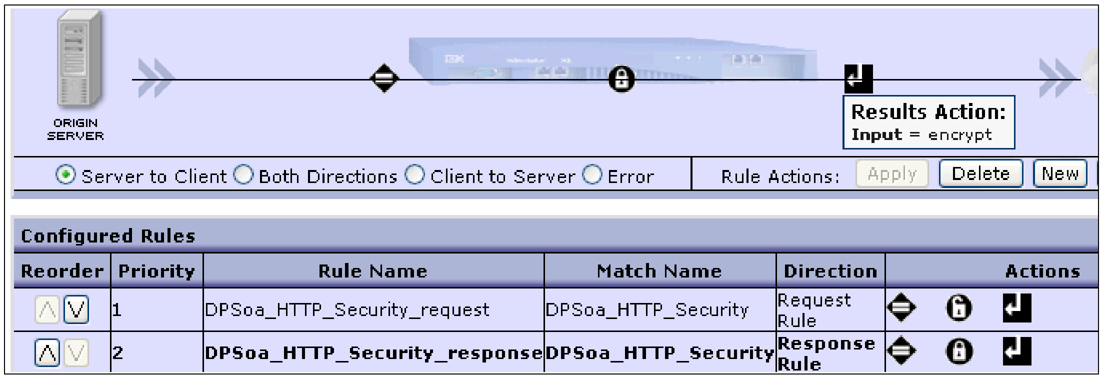
Encrypt Action:
Input = INPUT
Transform = store:/□/□/□encrypt-wssec.xsl
Output = encrypt
Stylesheet Parameter = {http:/□/□www.datapower.c
http:/□/□www.w3.org/□2003/□05/□soap-envelope/
{http:/□/□www.datapower.com/□param/□config}al
2001/□04/□xmlenc#aes256-cbc,
{http:/□/□www.datapower.com/□param/□config}re
{http:/□/□www.datapower.com/□param/□config}to
KeyIdentifier,
{http:/□/□www.datapower.com/□param/□config}ws
keyidentifier-valuetype: http:/□/□docs.oasis-open.
200401-wss-x509-token-profile-1.0#X509v3Subject
{http:/□/□www.datapower.com/□param/□config}ws

Configured Rules

Reorder	Priority	Rule Name	N
∧ ∨	1	DPSoa_HTTP_Security_request	DPSoa
∧ ∨	2	**DPSoa_HTTP_Security_response**	**DPSoa**

Figure 2-53 XML firewall policy Reply rule: Encrypt Action

g. The Results Action is an encrypted reply message to be sent to a client as shown in Figure 2-54 on page 57.

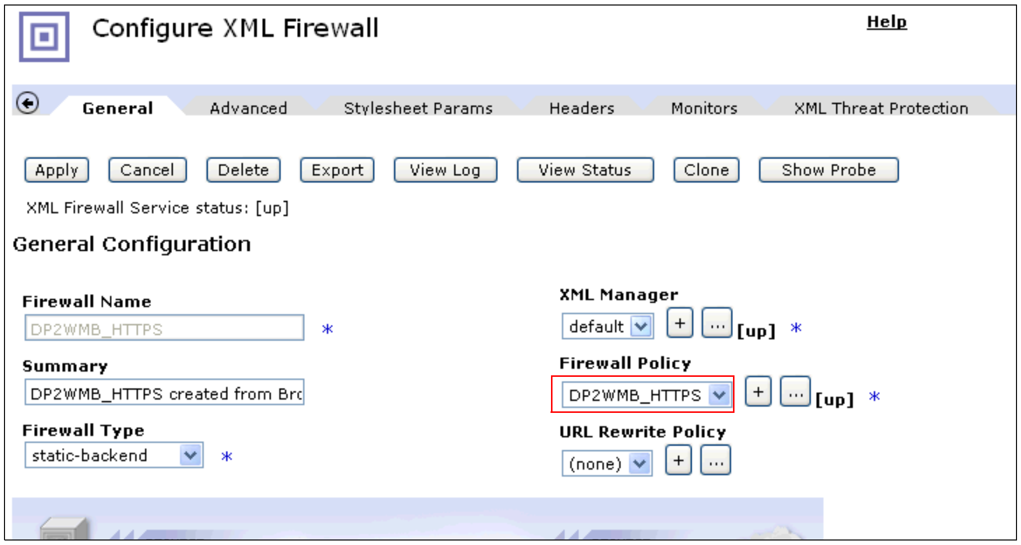


Configured Rules

Reorder	Priority	Rule Name	Match Name	Direction	Actions
∧ ∨	1	DPSoa_HTTP_Security_request	DPSoa_HTTP_Security	Request Rule	
∧ ∨	2	DPSoa_HTTP_Security_response	DPSoa_HTTP_Security	Response Rule	

Figure 2-54 XML firewall processing policy: Response Rule

3. You can repeat step 2 to examine the XML firewall policy created by the wizard for the HTTPS input node.



XML Firewall Service status: [up]

General Configuration

Firewall Name	DP2WMB_HTTPS *
Summary	DP2WMB_HTTPS created from Brc
Firewall Type	static-backend *
XML Manager	default [up] *
Firewall Policy	DP2WMB_HTTPS [up] *
URL Rewrite Policy	(none)

Figure 2-55 XML firewall policy: DP2WMB_HTTPS

Important note: You have completed and verified the configuration of the XML Firewall Gateway for the HTTP and HTTPS connectivity to WMB using the security wizard. The XML firewall is configured to encrypt a request message and decrypt a reply message.

You need to shop at this point and start testing this configuration. It is better to test each configuration as you complete it before moving on to the next configuration.

4. Test the XML Firewall Gateway used in scenario one by following the instructions in 2.4.3, “Running the scenarios” on page 79.

2.3.4 Configure DataPower to add support for Digital Signature

In this section, we explain how to add a policy rule to an existing XML firewall policy to support digital signature. Digital signature is an important feature of Web services security that ensures message integrity. We added this extra step, because the current version of the security wizard, provided by the IS02 support pack, does not let you configure digital signatures.

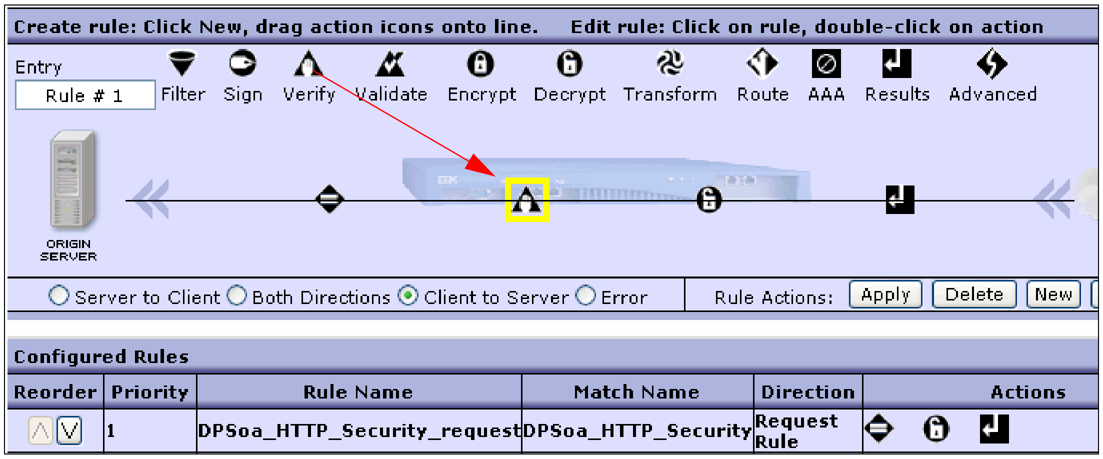
You will learn the steps to add the support for a message signature into an existing DataPower XML Firewall policy:

1. Add a Verify action to verify a signed request message.
2. Add a Sign action to sign a response message.

Add a Verify action to verify a signed request message

On inbound traffic, the XML firewall will be configured to verify the digital signatures contained in the signed messages and documents by adding the Verify action to the policy:

1. Select the **XML Firewall** icon from the Control Panel.
2. Select the XMP firewall **DP2WMB_HTTP** from the Configure XML Firewall pane as shown in Figure 2-47 on page 54.
3. Select the **DP2WMB_HTTP** policy under the Firewall Policy section and click the ellipsis (**...**) next to it to edit the policy as shown in Figure 2-48 on page 54.
4. Drag the **Verify** action icon before the Encrypt action from the Configure XML Firewall Policy pane as shown in Figure 2-56.



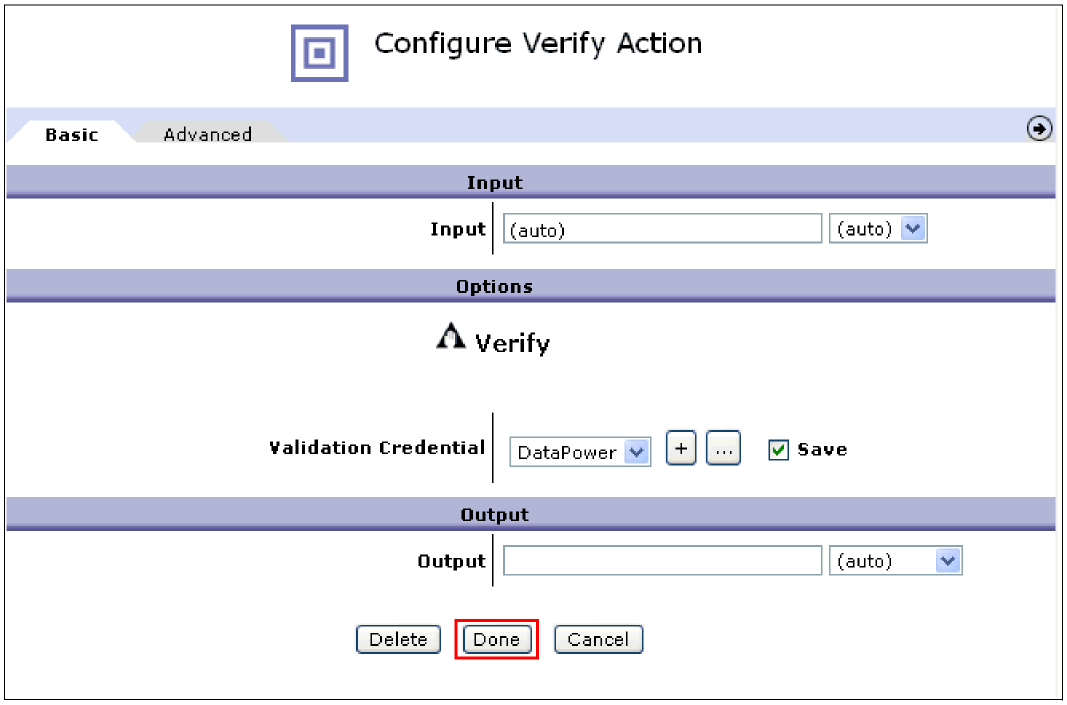
Reorder	Priority	Rule Name	Match Name	Direction	Actions
	1	DPSoa_HTTP_Security_request	DPSoa_HTTP_Security	Request Rule	

Figure 2-56 Configure XML Firewall Policy: Verify action

5. Double-click the **Verify** icon (highlighted in a yellow box) to display the Configure Verify Action panel.
6. Select the following options as shown in Figure 2-57 on page 59:
   a. Input: **auto**
   b. Validation credential: **DataPower**
   c. Output: **auto**
7. You can click the Advanced tab to define advanced settings. Refer to the *DataPower Web GUI Guide* for instructions about how to configure advanced settings. It is an online document. You can download it as part of the common documentation from:

   `http://www-1.ibm.com/support/docview.wss?rs=2362&uid=swg24014405`

   This sample does not require advanced settings.
8. Click **Done** to complete the action, as shown in Figure 2-57 on page 59.


Configure Verify Action

Basic | Advanced

Input

Input	(auto) (auto)

Options

Verify

Validation Credential	DataPower [✓ Save]

Output

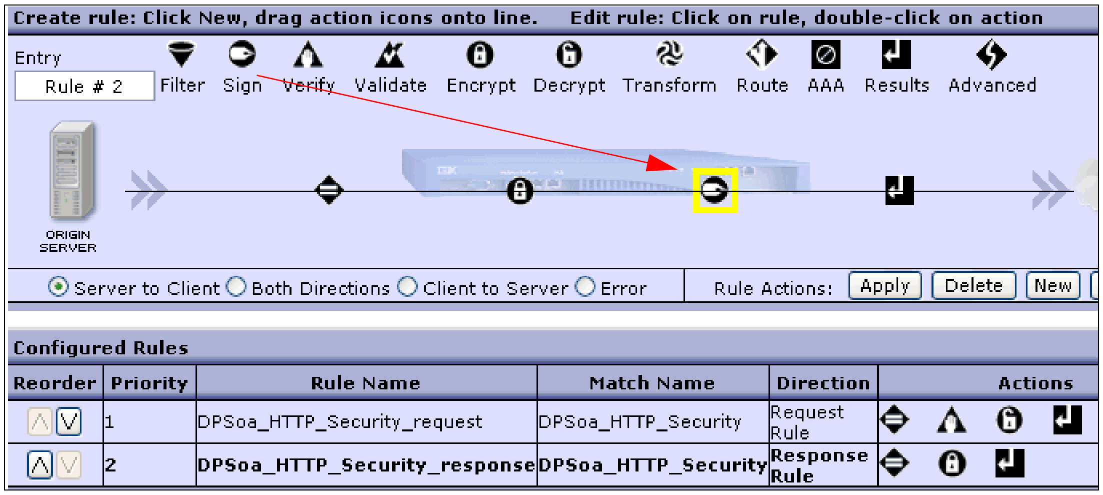
Output	(auto)

Figure 2-57 XML Firewall Policy: Add Verify action

Add a Sign action to sign a response message

On outbound traffic, we will configure the XML firewall to add a Sign action to the processing policy to attach a digital signature to messages and documents.

1. From the Configure XML Firewall Policy pane, select the **Response Rule** to highlight it.
2. Drag and drop the **Sign** icon as shown in Figure 2-58.


Create rule: Click New, drag action icons onto line. Edit rule: Click on rule, double-click on action

[✓ Server to Client] Both Directions Client to Server Error

Configured Rules

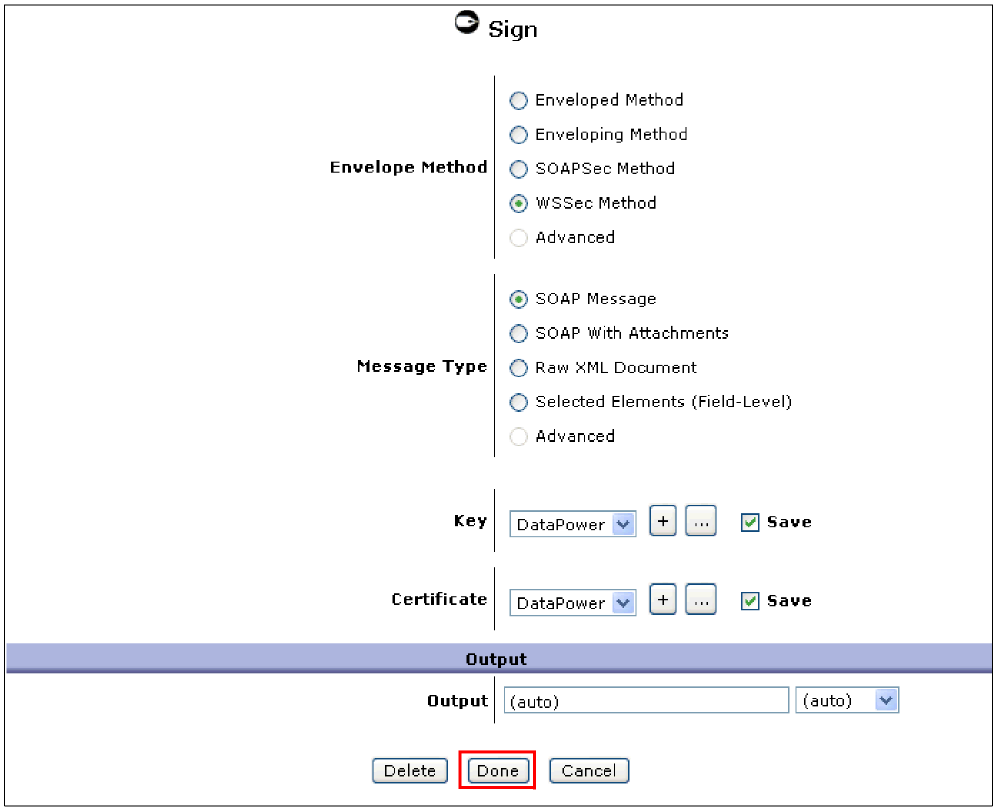
Reorder	Priority	Rule Name	Match Name	Direction	Actions
	1	DPSoa_HTTP_Security_request	DPSoa_HTTP_Security	Request Rule	
	2	DPSoa_HTTP_Security_response	DPSoa_HTTP_Security	Response Rule	

Figure 2-58 Configure XML firewall policy: Add Sign action

3. Double-click the **Sign** icon to display the Sign Action panel and select the following information as shown in Figure 2-59 on page 60:
   a. Envelope Method: **WSSec Method**
   b. Message Type: **SOAP Message**

c. Key: **DataPower**

d. Certificate: **DataPower**


Sign

Envelope Method: ( ) Enveloped Method ( ) Enveloping Method ( ) SOAPSec Method [✓ WSSec Method] ( ) Advanced

Message Type: [✓ SOAP Message] ( ) SOAP With Attachments ( ) Raw XML Document ( ) Selected Elements (Field-Level) ( ) Advanced

Key: DataPower ✓ Save

Certificate: DataPower ✓ Save

Output

Output: (auto) (auto)

Figure 2-59 Sign action

4. Click **Done** to complete.
5. Back to the Configure XML firewall policy panel (Figure 2-58 on page 59), click **Apply** next to the Rule Actions, because this is the last action that you add to the policy.
6. Click **Close** in the Select a Policy Name section at the top of the panel.
7. Back to the Configure XML Firewall panel, click **Apply.**
8. Click **Save Config** to complete.

Note: You have added the Sign action to an XML firewall policy. The processing policy is now configured to handle encryption, decryption, and digital signature. You need to stop at this point and start testing this configuration.

9. Test the configuration by following the instruction in “Step 3: Add digital signature to the loopback firewall and test” on page 86 if you choose to create a Loopback firewall and use it to generate your own encrypted and signed message. You can go directly to “Step three: Test XML firewalls with prebuilt, encrypted/signed message” on page 92 if you use the sample message that is provided for you.

2.4 Configuration of DataPower and WMB for Scenario two

In this chapter, we explain the processing logic of the message flows for scenario two. You will learn how to configure DataPower and WebSphere Message Broker to test the scenario in which the message broker invokes the service provided by the existing application via MQ. You will also learn how to test and perform simple problem determination.

We use a DataPower Multi-Protocol Gateway as a front-end security gateway to process the WS-security and MQ connectivity. We explain how to manually configure DataPower without using the security wizard, because the current version of the security wizard does not support the configuration of the Multi-Protocol Gateway that is needed to support MQ connectivity. The benefit of this manual exercise is to show you all of the steps required to configure DataPower end-to-end. You will have a better understanding of DataPower after this exercise.

First, you will learn the message flow logic that provides transformation and connectivity to the front-end and back-end subsystems. Unlike the first scenario where the message flow uses different protocols to connect to the front-end and back-end subsystems, the message flow in this scenario uses MQ protocol for connectivity end-to-end.

Next, you will learn to configure DataPower Multi-Protocol Gateway to provide Web service security for the message flows. This manual configuration does not employ the security wizard.

2.4.1 Message flow logic

Two main message flows provide the logical processing of scenario two:

- HOST_Request.msgflow
- HOST_Reply.msgflow

The message flows use two subflows:

- SaveOriginalMQMD_Sub.msgflow
- RestoreOriginalMQMD_Sub.msgflow

Similar to the first scenario, the HOST_Simulation_MF message flow is used to simulate a CICS COBOL application. It monitors the HOST_REQUEST queue for incoming messages, processes them, and puts the reply message in the HOST_REPLY queue. This message flow also reuses the Error_Handler subflow for error processing.

HOST_Request.msgflow

This flow, which is shown in Figure 2-60 on page 62, provides connectivity to the DataPower appliance and the CICS application using MQ. It also transforms messages from SOAP/XML to fixed format/COBOL to integrate an SOA application with the existing application. It uses the same input data, data definitions (message sets), and transformation logic as the first scenario and has the following processing logic:

- Receives a SOAP message via the MQInput node. The input message is a request for customer information and status. The request contains the customer account number.
- Transforms the SOAP message from XML/SOAP into a fixed format via the XML_To_COBOL compute node.
- Puts the transformed message to the HOST_REQUEST queue, which is monitored by the CICS Simulation flow.
- Builds the second state message with MQ Message Descriptor (MQMD) only. There is no need to store the payload data. The purpose of this second message is to save the

state/context of the MQMD (message identified and ReplyToQ) that is used to build the reply message.

- Writes this state message to the MQMD_STOREQ for later retrieval (see HOST_Reply.msgflow). It uses a StoreOriginalMQMD_Sub subflow to persist the store message to a queue.

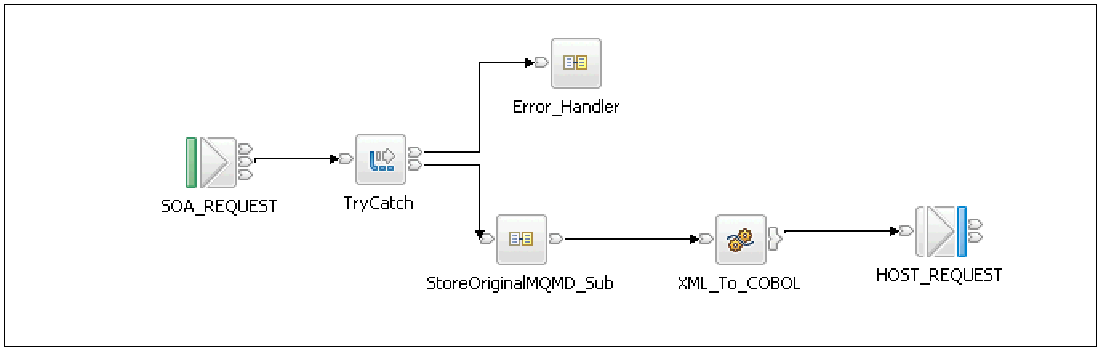


Figure 2-60 HOST_Request message flow

HOST_Reply.msgflow

This flow processes the reply message from the CICS application. It transforms data to the correct format and restores the MQMD context of the original request to build a message header for the reply message. The message header of the reply message must contain the ReplyToQ and original message Identifier/correlation ID. It performs the following functions:

- Retrieves the reply message from CICS from the HOST_REPLY2 queue.
- Transforms the SOAP message from a fixed format/COBOL to XML/SOAP via the COBOL_To_XML compute node.
- Retrieves the state message from the MQMD_STOREQ to restore the original MQMD using the RestoreOriginalMQMD_Sub subflow.
- Puts the transformed message to the ReplyToQ that is specified in the MQ message header MQMD.

The message flow HOST_Reply.msgflow processing logic is illustrated in Figure 2-61.

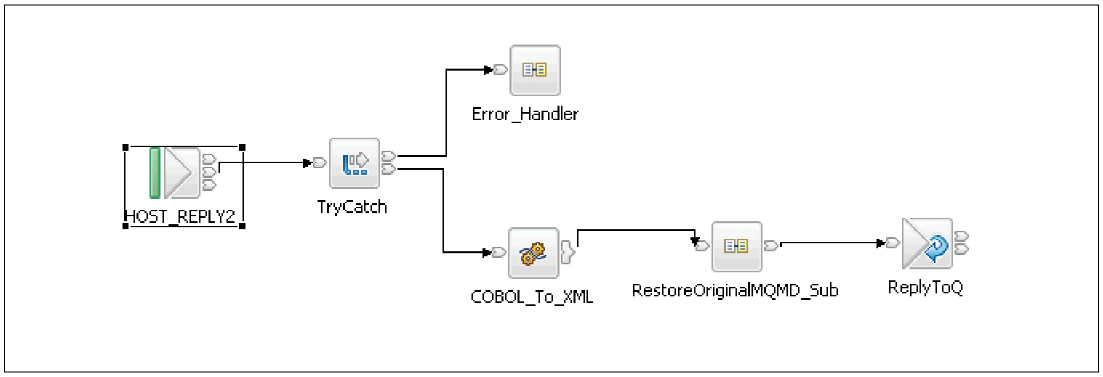


Figure 2-61 HOST_Reply message flow

SaveOriginalMQMD_Sub.msgflow

This subflow is used to save the original MQMD context from the original request message to the MQMD_STOREQ. It is required in the request and reply processing pattern to preserve the original message identifier in the reply message to correlate the request and reply messages. Figure 2-62 shows the SaveOriginalMQMD_Sub.msgflow.

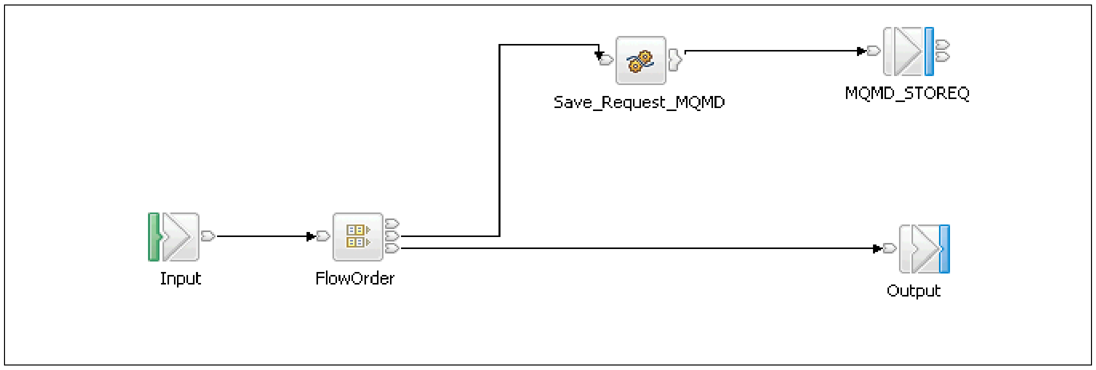


Figure 2-62 SaveOriginalMQMD_Sub.msgflow

RestoreOriginalMQMD_Sub.msgflow

This subflow, which is shown in Figure 2-63, retrieves the state message from the MQMD_STOREQ. It uses an MQGET node with the MQGET property set to *Get by correlation ID*. The MQGET node has three possible results:

- Successfully get a message from MQMD_STOREQ based on the correlation ID.
- Get a message with a warning.
- Unable to get a message based on the correlation ID.

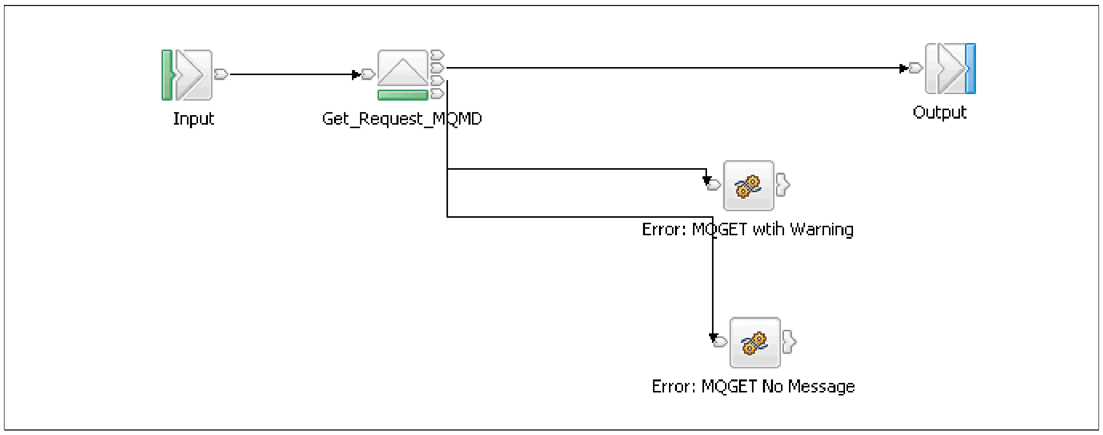


Figure 2-63 RestoreOriginalMQMD_Sub subflow

2.4.2 Configure the DataPower Multi-Protocol Gateway

In the second scenario, the Multi-Protocol Gateway, a different type of gateway, provides the Web services security front end and connectivity to MQ. The DataPower Multi-Protocol Gateway is configured to provide the same XML security service for message encryption and decryption. You can configure the Multi-Protocol Gateway to process messages from various protocols (MQ, FTP, HTTP, TIBCO EMS, ODBC, and so forth). However, the scope of this sample is limited to the MQ protocol. You can expand this sample to include other protocols to reflect actual use cases. You can optionally add digital signature support to the configuration as shown in 2.3.4, "Configure DataPower to add support for Digital Signature" on page 57. We only configure it for message encryption and decryption.

The configuration of WebSphere Message Broker for this scenario is simple. You only need to define the queues that are used by the message flows. The definitions of these queues are in the MQ_Conf.bat script. You have already run this script as part of the configuration for WebSphere Message Broker for scenario one. The list of queues is shown in Table 2-2 on page 25.

We will configure the DataPower Multi-Protocol Gateway based on Figure 2-2 on page 20. We can reuse the following artifacts or objects previously created in scenario one:

- Application domain: `WMBIntegration`
- User ID: `wmbadmin`
- Crypto Keys
- Crypto Identification Credentials
- Crypto Profile

The steps to perform the configuration are:

1. Create a DataPower Multi-Protocol Gateway.
2. Add the processing and security policy to the gateway.
3. Create DataPower security policies to decrypt inbound messages and encrypt outbound messages.
4. Configure the back-end URL.
5. Configure the MQ Front Side Handler.
6. Test the sample using RFHUtil.

Step 1: Create a DataPower Multi-Protocol Gateway

The DataPower Multi-Protocol Gateway is integrated with WebSphere Message Broker via MQ. it is also the front-end gateway for the MQ client that needs the current service provided by the CICS application:

1. Log on to DataPower application domain `WMBIntegration` using the user ID `wmbadmin`. Make sure that you are in the correct domain.
2. Select the **Multi-Protocol Gateway** icon and click **Add** to display the General configuration panel (Figure 2-64).

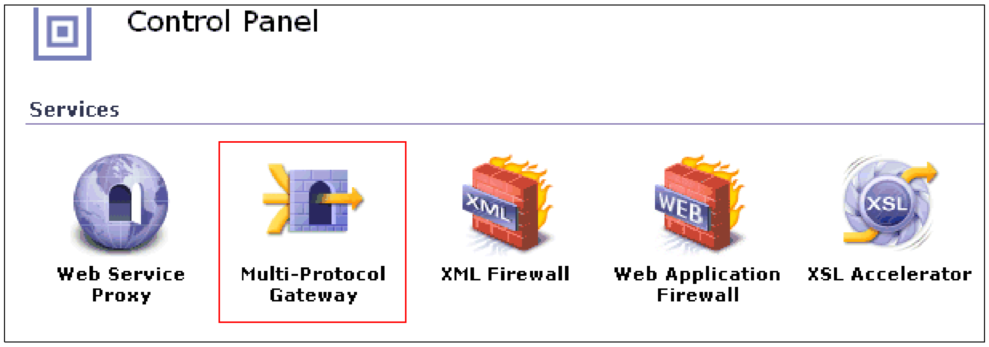
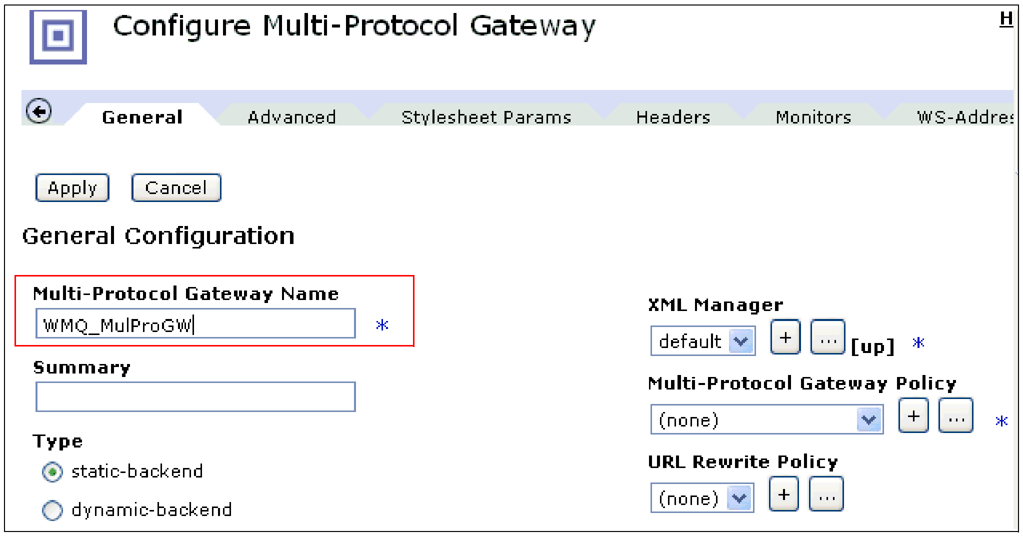


Figure 2-64 Control Panel: Multi-Protocol Gateway

3. In Figure 2-65 on page 65, set the **Multi-Protocol Gateway Name** to `WMQ_MulProGW` and accept other default values at this point. At anytime, you can click the Help link for detailed information about any options on a panel. For simplicity, we use the basic settings. There are advanced settings that you can explore to meet your specific requirements. Refer to the DataPower manuals for detailed information about the advanced settings.



Multi-Protocol Gateway Name: WMQ_MulProGW *

Summary:

Type: [✓ static-backend] dynamic-backend

XML Manager: default [up] *

Multi-Protocol Gateway Policy: (none) *

URL Rewrite Policy: (none)

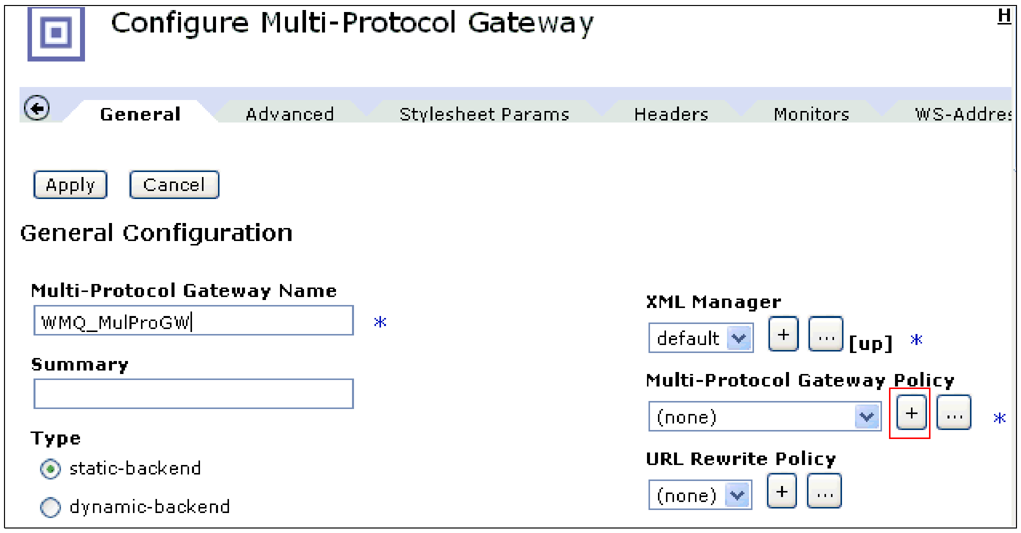
Figure 2-65 Configure Multi-Protocol Gateway

Step 2: Add the processing policy to the gateway

The *processing policy* defines the rules that are used against the messages that pass through the gateway. A *rule* consists of a matching rule and a processing rule. This scenario uses a URL matching rule that employs a simple URL matching pattern to match incoming traffic. When the incoming traffic passes the matching rule, processing rules are applied against the message or document.

In this scenario, we use the same processing rules that are used in the policy configured for the XML Firewall Gateway in scenario one. The URL Match rules are different, because we use MQ protocol instead of HTTP/HTTPS. The processing rules encrypt the inbound messages and decrypt the outbound messages that pass through the gateway. This step explains how to create the processing policy for the WMQ_MulProGW gateway:

1. Click the plus symbol (**+**) beside the **Multi-Protocol Gateway Policy** field as illustrated in Figure 2-66.



Multi-Protocol Gateway Name: WMQ_MulProGW *

Summary:

Type: [✓ static-backend] dynamic-backend

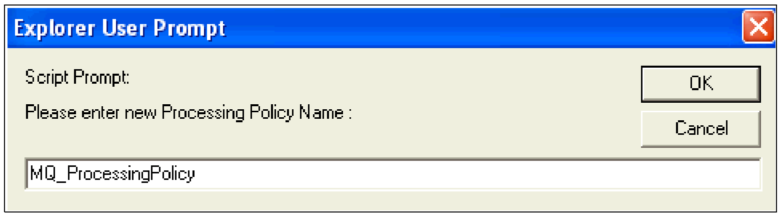
XML Manager: default [up] *

Multi-Protocol Gateway Policy: (none) *

URL Rewrite Policy: (none)

Figure 2-66 Add the processing policy to the Multi-Protocol Gateway

2. Enter the name `MQ_ProcessingPolicy` in the Multi-Protocol Gateway Processing Policy Name field and click **OK**. Click **OK** again if you receive the warning message "`Create your New Policy Rule, and then` **`Apply`** `to commit changes`".


Explorer User Prompt

Script Prompt:

Please enter new Processing Policy Name :

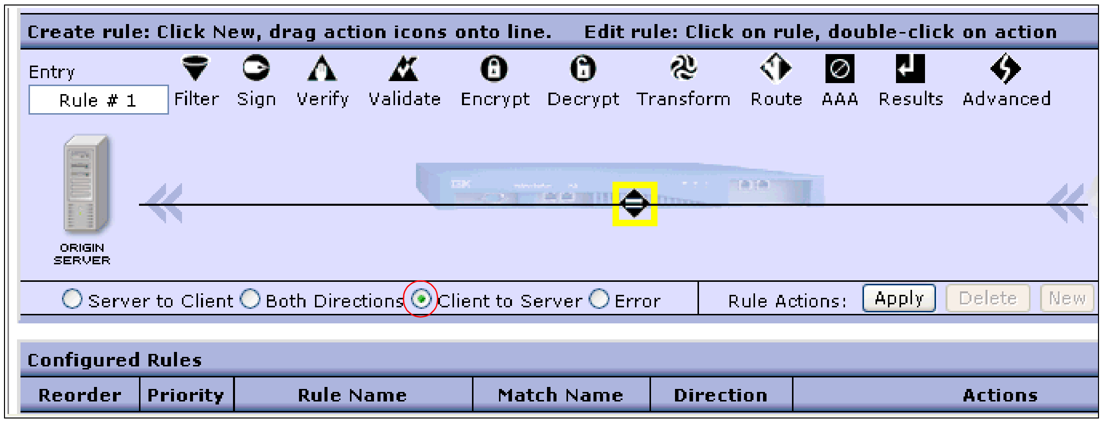
MQ_ProcessingPolicy

Figure 2-67 Specify Processing Policy Name

3. From the Configure Multi-Protocol Gateway Policy panel:

   a. Click **Client to Server** (highlighted in red). The default is Both directions. For this scenario, we have to configure two paths differently: one path for the inbound (Client to Server) and another path for the outbound (Server to Client).

   b. Double-click the **Match** icon (highlighted in a yellow box in Figure 2-68) to configure a matching rule.


Create rule: Click New, drag action icons onto line. Edit rule: Click on rule, double-click on action

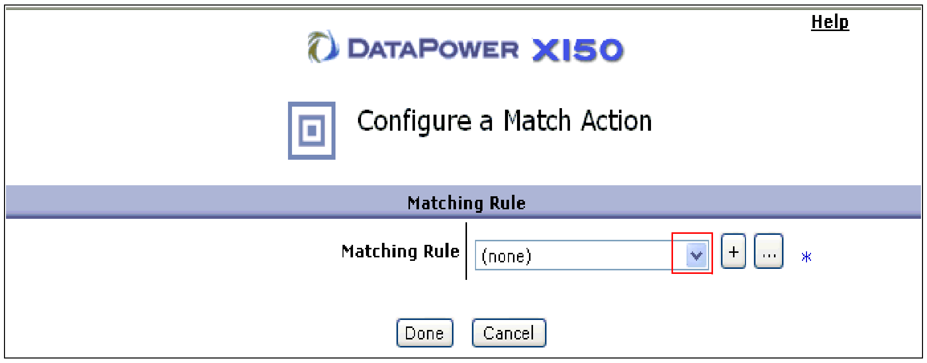
Entry: Rule # 1

○ Server to Client ○ Both Directions [✓ ● Client to Server] ○ Error

Reorder	Priority	Rule Name	Match Name	Direction	Actions

Figure 2-68 Configure processing policy for request path: match rule

4. Click **+** to create a new matching rule.

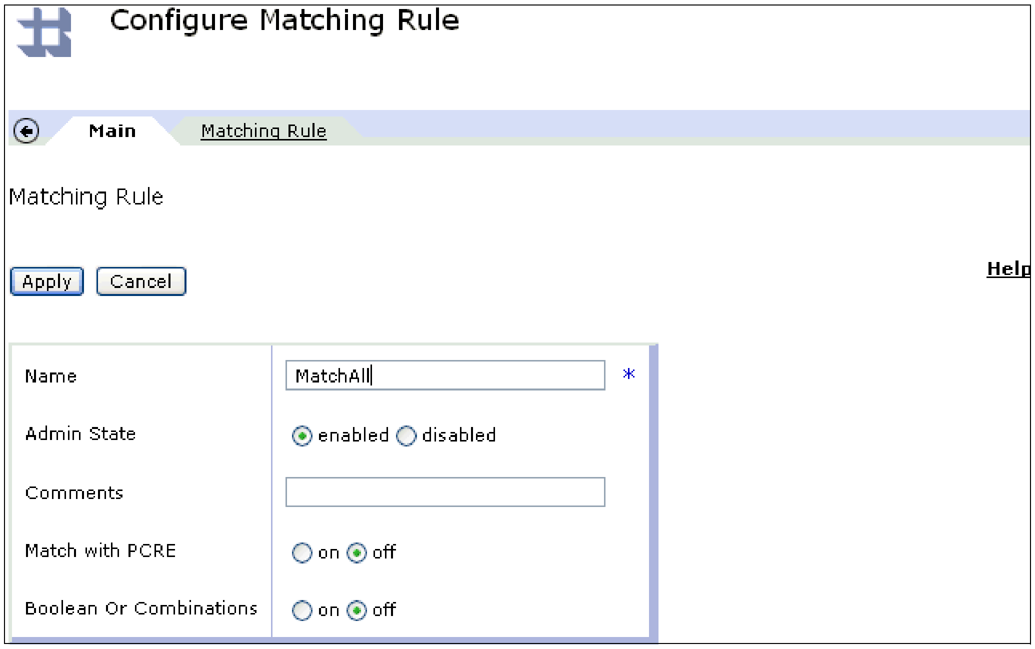

Configure a Match Action

Matching Rule

Matching Rule: (none)

Figure 2-69 Configure a matching rule: Select from an existing matching rule

5. Specify `MatchAll` as the rule Name and set Admin State to **enabled** on the Main tab. You have created the MatchAll rule, as shown in Figure 2-70 on page 67.

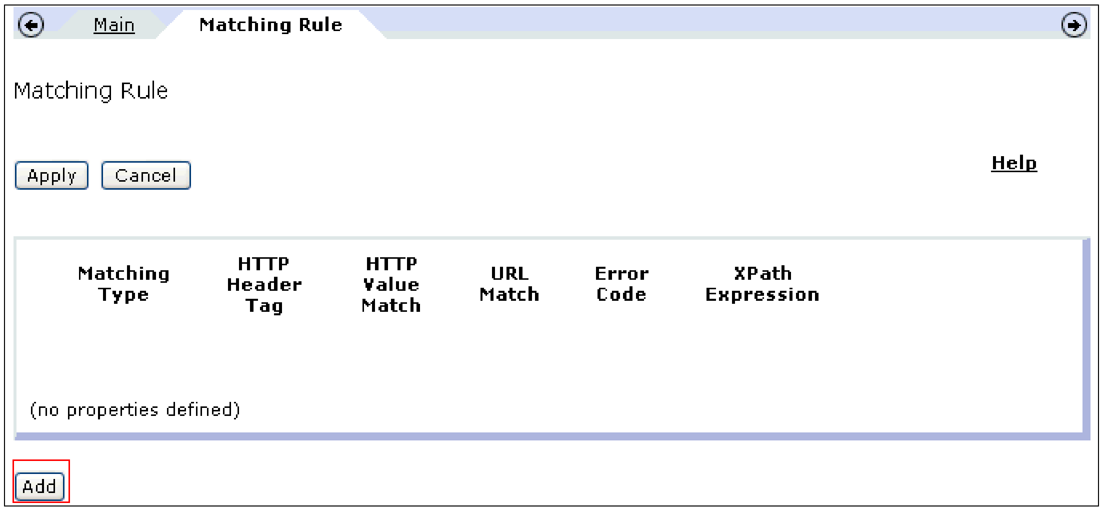

Configure Matching Rule

Matching Rule

Name	MatchAll *
Admin State	[✓ enabled] disabled
Comments	
Match with PCRE	on [✓ off]
Boolean Or Combinations	on [✓ off]

Figure 2-70 Configure Matching Rule Main tab

6. Click the **Matching Rule** tab and then click **Add**.



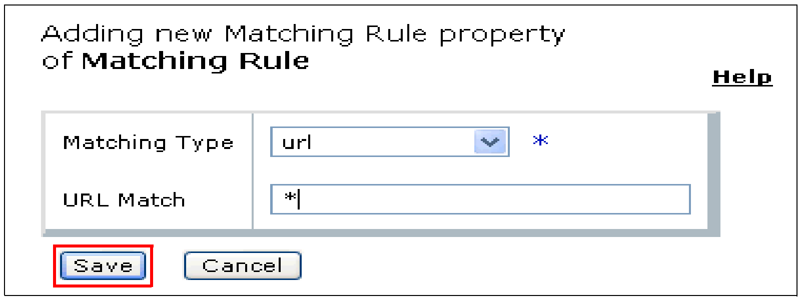
Matching Rule

Matching Type	HTTP Header Tag	HTTP Value Match	URL Match	Error Code	XPath Expression

(no properties defined)

Figure 2-71 Configure Matching Rule: Matching Rule tab

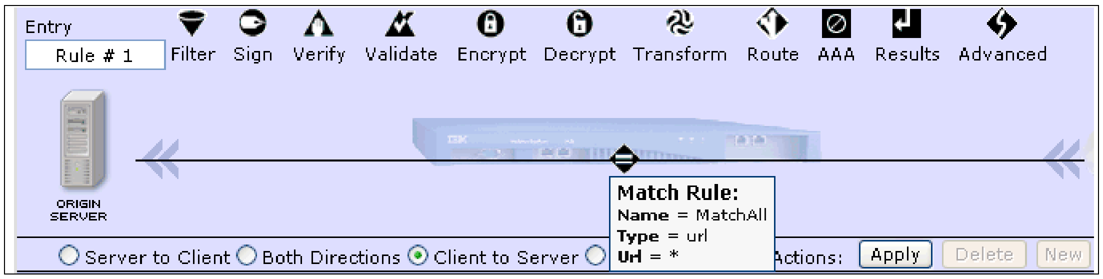
7. In Figure 2-72 on page 68, select **url** in the Matching Type field, enter an asterisk character (*) in the Url Match field, and click **Save**.


Adding new Matching Rule property of **Matching Rule**

Matching Type	url *
URL Match	*

Figure 2-72 Matching rule specification

8. Back at the Matching Rule panel, click **Apply** to finish.
9. Hover your mouse over the **Match** icon/action to verify that the rule has been successfully created, as shown in Figure 2-73.



Match Rule:
Name = MatchAll
Type = url
Url = *

Figure 2-73 Verify Match Rule

10. Drag and drop the **Decrypt** icon into the configuration path (the horizontal line) and to the right of the Match rule, as shown in Figure 2-74.

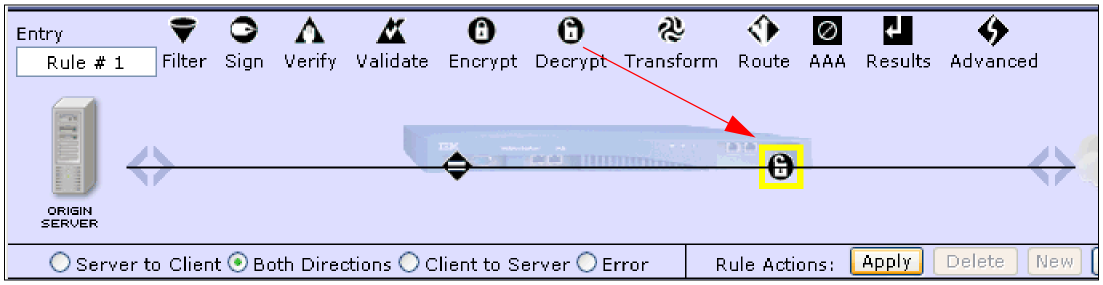
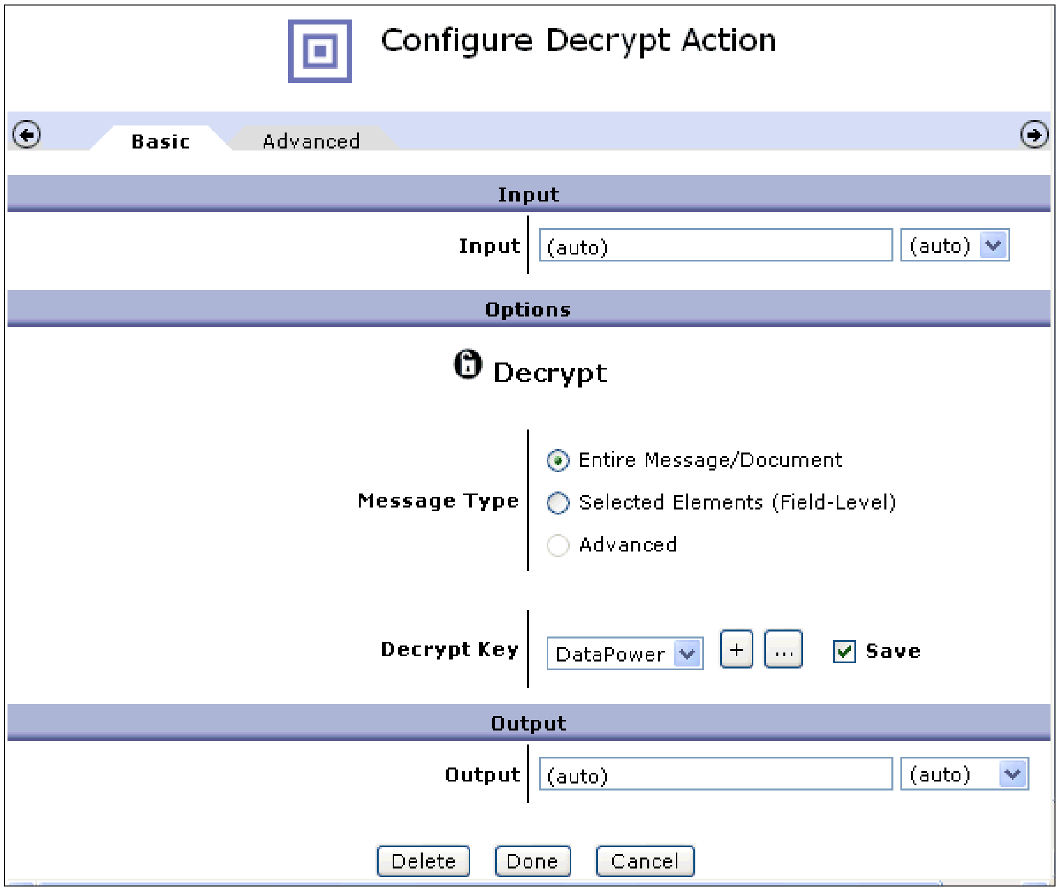


Figure 2-74 Add Decrypt action

11. Double-click the **Decrypt** icon to display the Decrypt Action panel and enter the information as shown in Figure 2-75 on page 69. We are reusing the Crypto Key DataPower created for the XML Firewall Gateway in “Step four: Crypto settings for DataPower and clients connectivity” on page 46:
    a. Select the Message Type radio button **Entire Message/Document.**
    b. Select the DataPower Crypto Key **Decrypt Key** from the drop-down list.
    c. Set Output to **auto** from the drop-down list.


Configure Decrypt Action

Basic | Advanced

Input

Input: (auto) (auto)

Options

Decrypt

Message Type: [✓ Entire Message/Document] Selected Elements (Field-Level) Advanced

Decrypt Key: DataPower + ... [✓ Save]

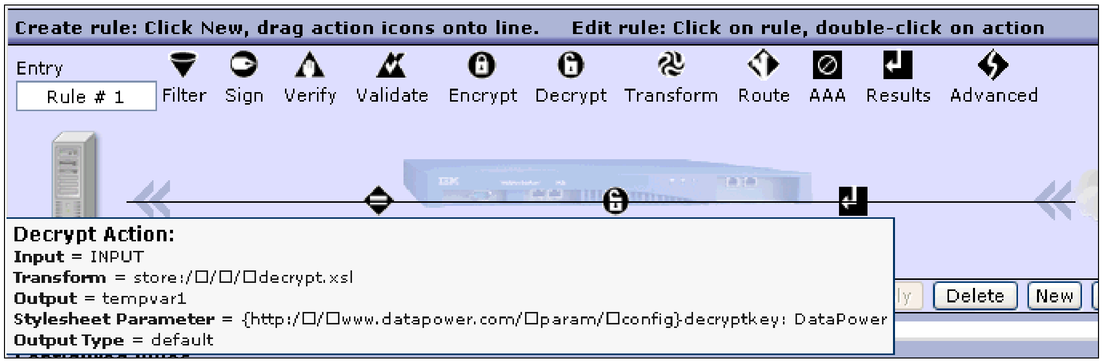
Output

Output: (auto) (auto)

Delete Done Cancel

Figure 2-75 Configure Decrypt action

12.Click **Done** to finish and verify that the decryption rule is created as shown in Figure 2-76.


Create rule: Click New, drag action icons onto line. Edit rule: Click on rule, double-click on action

Decrypt Action:
Input = INPUT
Transform = store:///decrypt.xsl
Output = tempvar1
Stylesheet Parameter = {http://www.datapower.com/param/config}decryptkey: DataPower
Output Type = default

Figure 2-76 Verify Decrypt Action

13.In Figure 2-77 on page 70, drag and drop the **Results** icon (highlighted in a yellow box) into the configuration path. The Results Action sends the result to the remote server and awaits a result from the server.

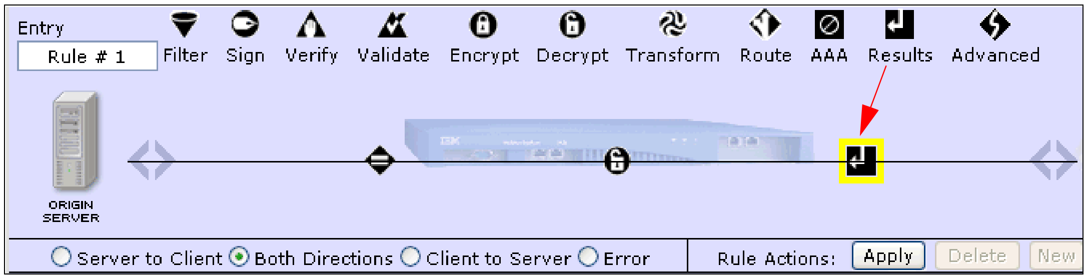
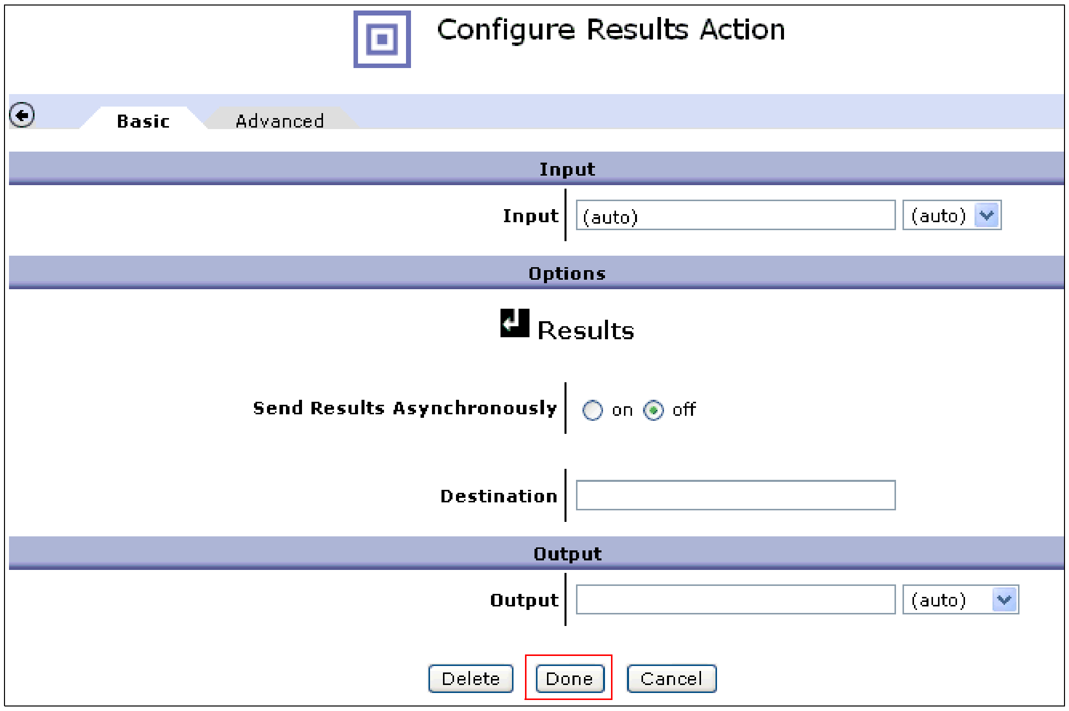


Figure 2-77 Add Results action

14. Accept the default information as shown in Figure 2-78.



Configure Results Action

Basic | Advanced

Input

Input: (auto) (auto)

Options

Results

Send Results Asynchronously: on [✓ off]

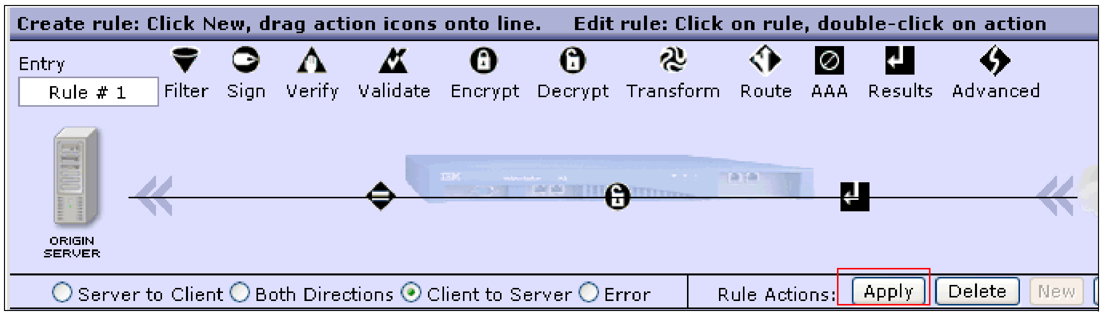
Destination:

Output

Output: (auto)

Figure 2-78 Configure Results Action

15. Click **Done**.
16. Back on the Configure Multi-Protocol Gateway Policy panel (Figure 2-79), click **Apply**.
17. Click **Save Config** to save configuration.



Create rule: Click New, drag action icons onto line. Edit rule: Click on rule, double-click on action

Figure 2-79 Apply changes to the processing rule for the request path

At this point, you have completed the creation of the match and processing rules for the request message. Next, you are going to create the match and processing rules for the response message. While messages for inbound traffic are decrypted, messages for outbound traffic are encrypted.

18. Click **New** to create a processing path for outgoing traffic (response rules) as shown in the following Figure 2-80.

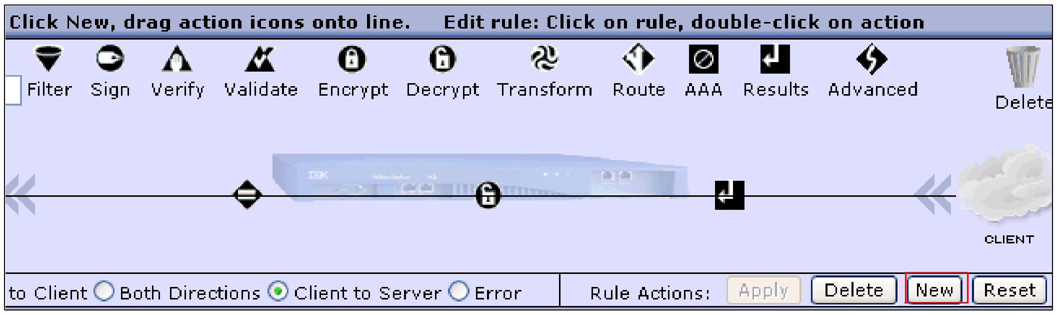


Figure 2-80 Add response rule path

19. From the Configure Multi-Protocol Gateway Policy panel (Figure 2-81):

   a. Click **Server to Client** (highlighted in red) for the outbound processing path. The default is Both Directions.

   b. Double-click the **Match** icon (highlighted in a yellow box in Figure 2-81) to configure the matching rules.

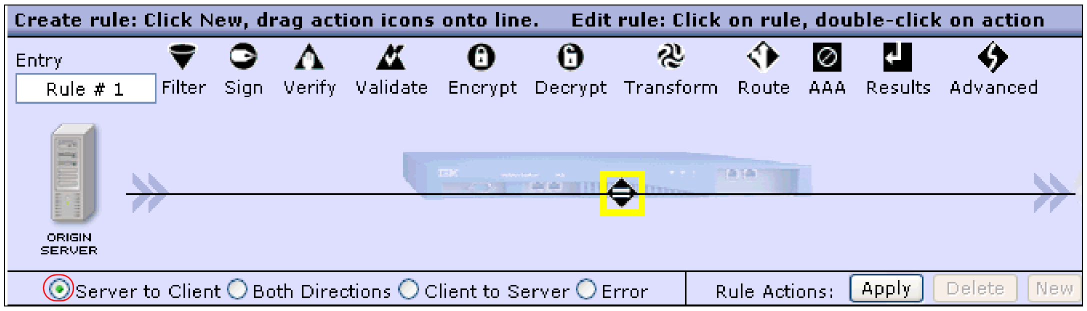


Figure 2-81 Configure processing rules for request path

20. Double-click the **Match** icon (highlighted in a yellow box), select the **MatchAll** rule that was previously created in Figure 2-72 on page 68, and click **Done** to finish as shown in Figure 2-82.

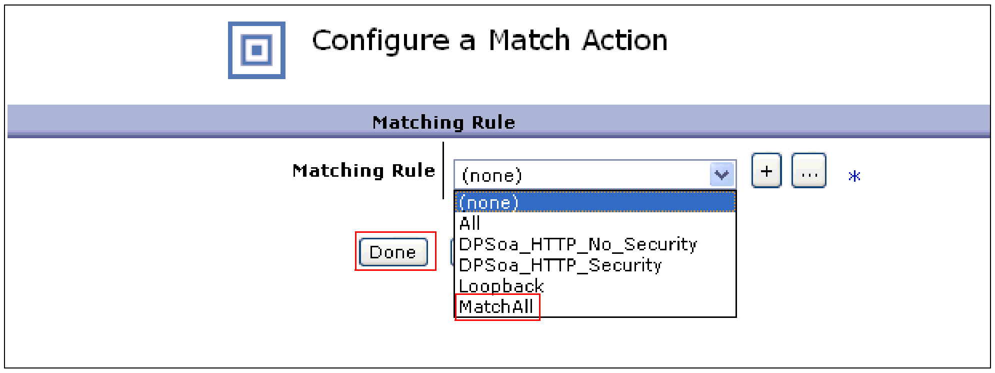


Figure 2-82 Configure a Match Action

21. Back to the Configure Multi-Protocol Gateway Policy panel (Figure 2-83), drag and drop the **Encryption** Action into the configuration path.

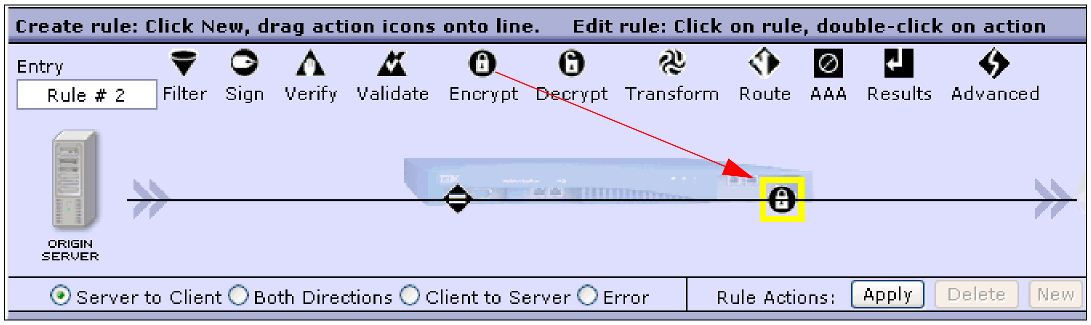
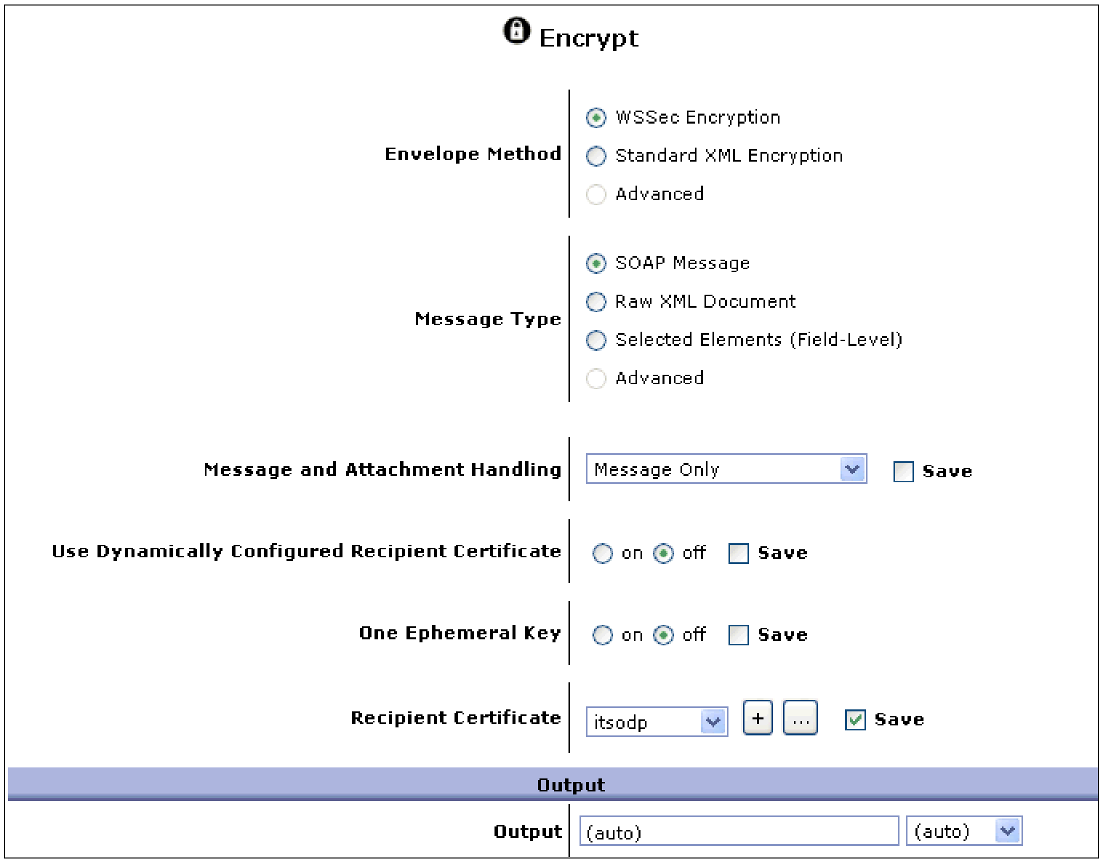


Figure 2-83 Add Encryption Action

22. Double-click the **Encrypt** (highlighted in a yellow box) icon to display the Encrypt Action panel and fill in information as shown in Figure 2-84:
    a. Select **WSSec Encryption** for Envelope Method.
    b. Select **SOAP Message** for Message Type.
    c. Set Message and Attachment Handling to **Message Only**.
    d. Select **itsodp** from the Recipient Certificate drop-down list.
    e. Set Output to **auto**.



Encrypt

Envelope Method	[✓ WSSec Encryption] Standard XML Encryption Advanced
Message Type	[✓ SOAP Message] Raw XML Document Selected Elements (Field-Level) Advanced
Message and Attachment Handling	Message Only ☐ Save
Use Dynamically Configured Recipient Certificate	on [✓ off] ☐ Save
One Ephemeral Key	on [✓ off] ☐ Save
Recipient Certificate	itsodp + ... ☑ Save

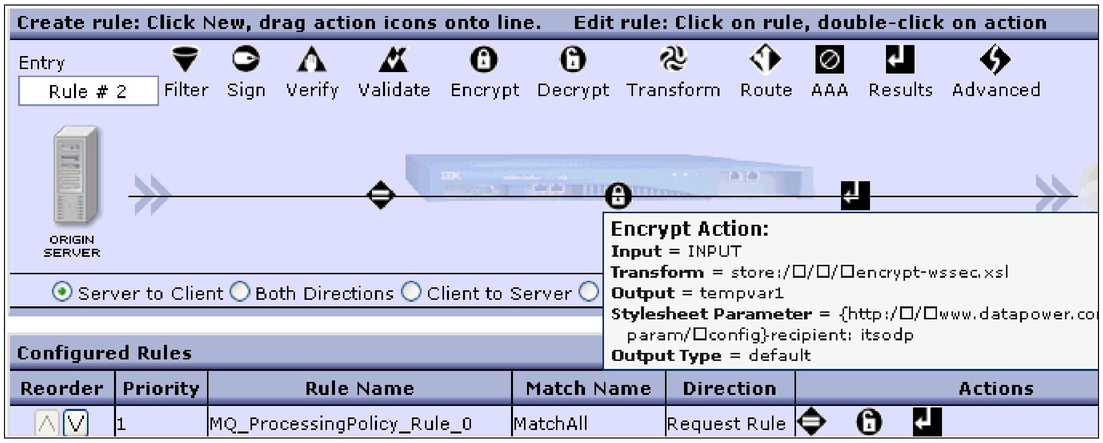
Output

Output	(auto) (auto)

Figure 2-84 Configure Encryption action

23. Click **Done** at the bottom to finish.

24.Verify that the Encryption rule is created as shown in Figure 2-85.



Encrypt Action:
Input = INPUT
Transform = store:///encrypt-wssec.xsl
Output = tempvar1
Stylesheet Parameter = {http://www.datapower.co param/config}recipient: itsodp
Output Type = default

Server to Client [✓] Both Directions Client to Server

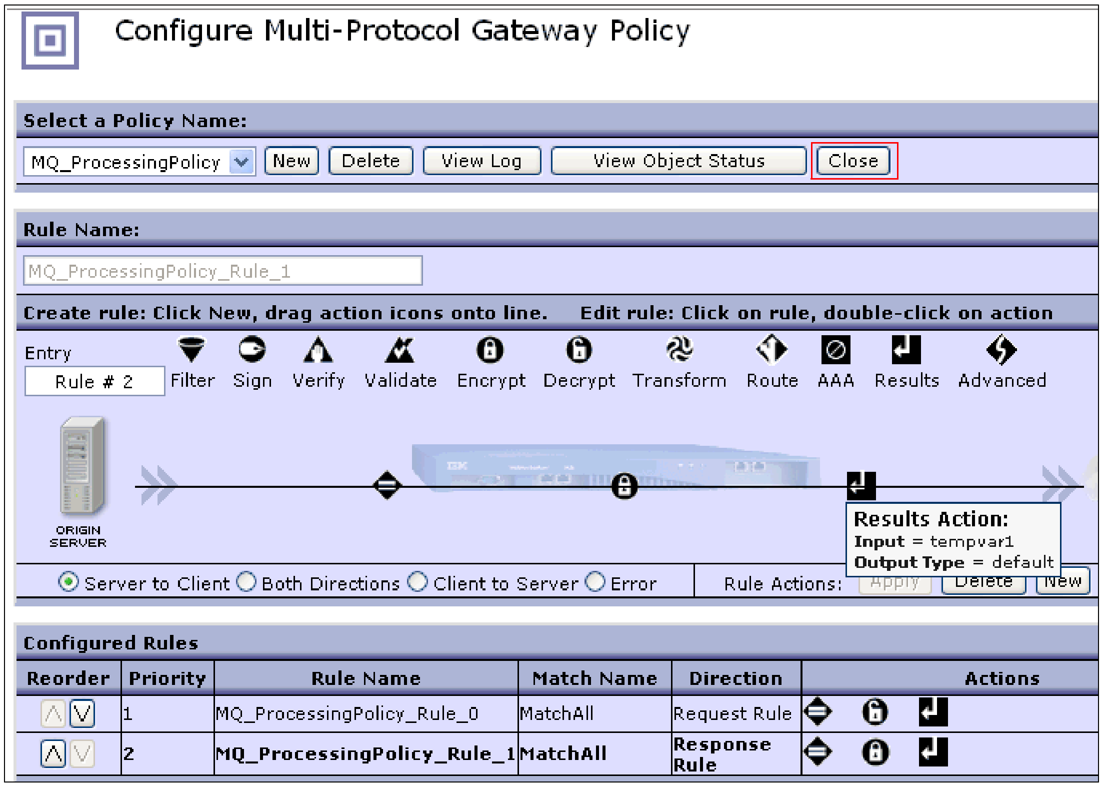
Reorder	Priority	Rule Name	Match Name	Direction	Actions
	1	MQ_ProcessingPolicy_Rule_0	MatchAll	Request Rule	

Figure 2-85 Verify Encrypt Action

25.Create the Results Action as shown in Figure 2-77 on page 70 and Figure 2-78 on page 70.

26.Back in the Configure Multi-Protocol Gateway Policy (Rule Actions section) panel, click **Apply** as shown in Figure 2-79 on page 70 to finish creating the Response policy.

27.Click **Close** in the Select a Policy Name section at the top of the panel. The Multi-Protocol Gateway Policy panel shows the two policy rules that you have configured as shown in Figure 2-86.



Select a Policy Name: MQ_ProcessingPolicy

Rule Name: MQ_ProcessingPolicy_Rule_1

Results Action:
Input = tempvar1
Output Type = default

Server to Client [✓] Both Directions Client to Server Error

Reorder	Priority	Rule Name	Match Name	Direction	Actions
	1	MQ_ProcessingPolicy_Rule_0	MatchAll	Request Rule	
	2	**MQ_ProcessingPolicy_Rule_1**	**MatchAll**	**Response Rule**	

Figure 2-86 Multi-Protocol Gateway configured for request and response rules

28.Click **Save Config** to save the configuration.

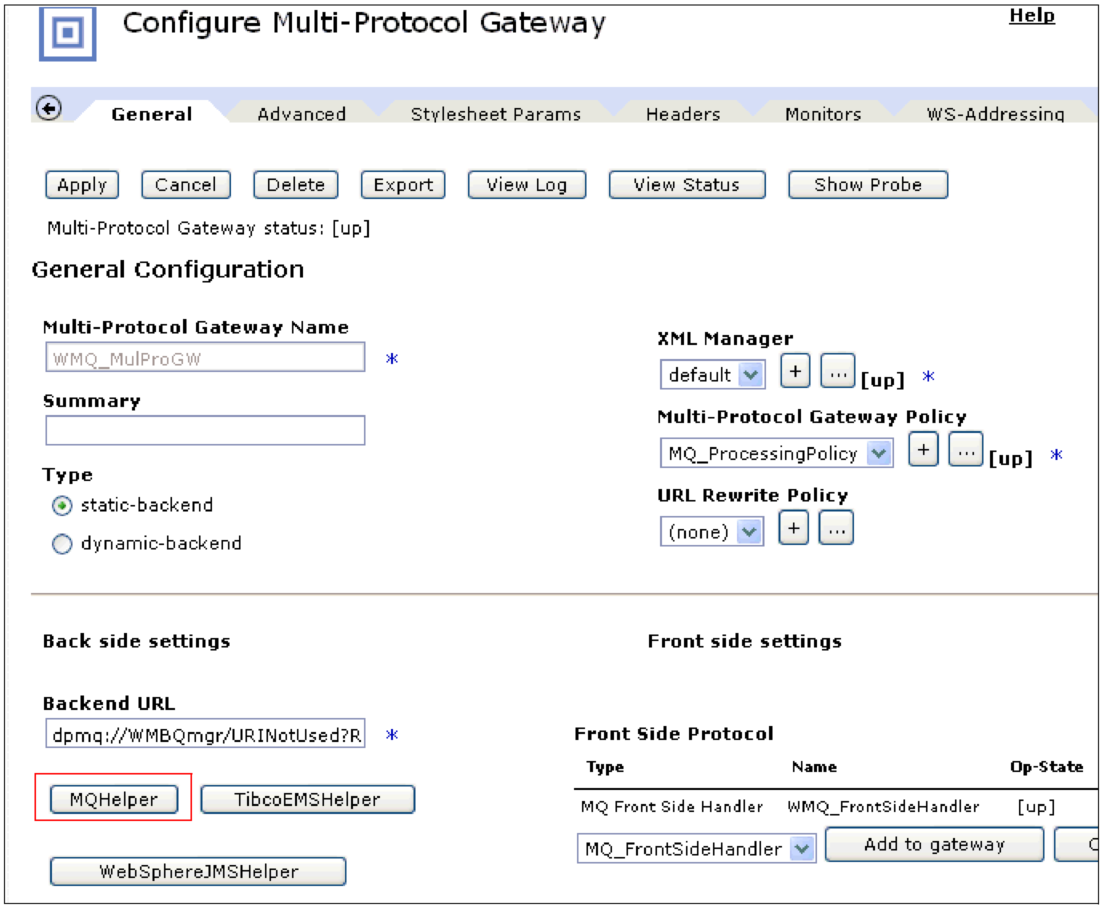
Step 3: Configure the back-end URL

DataPower has two types of protocol handlers: front side and back side handlers. The *Front Side Handler* handles communication between the client and the DataPower appliance. The *Back Side Handler* handles the communication between DataPower and the back-end systems. They are decoupled to simplify supporting different protocol switching.

These handlers determine network communication protocols, addresses, ports, and other protocol-specific settings. They support multiple protocols: MQ, JMS, FTP, HTTP, TIBCO EMS, and more front side handlers and back side handlers. We limited the scope of this scenario to the DataPower handlers for the MQ protocol.

The Back Side and Front Side Handler settings sections are listed under the General Configuration section in the Configure Multi-Protocol Gateway panel (Figure 2-87). It is extremely important to specify the correct communication properties for these handlers. For simplicity, we use the same queue manager for the front and back ends.

1. In Figure 2-87, click **MQHelper** under Back side settings to create a URL for the back-end queue manager. The *back-end queue manager* is the queue manager to which the message broker is connected.



Multi-Protocol Gateway status: [up]

General Configuration

Multi-Protocol Gateway Name: WMQ_MulProGW *

Summary:

Type: [✓ static-backend] dynamic-backend

XML Manager: default [up] *

Multi-Protocol Gateway Policy: MQ_ProcessingPolicy [up] *

URL Rewrite Policy: (none)

Back side settings

Backend URL: dpmq://WMBQmgr/URINotUsed?R *

Front side settings

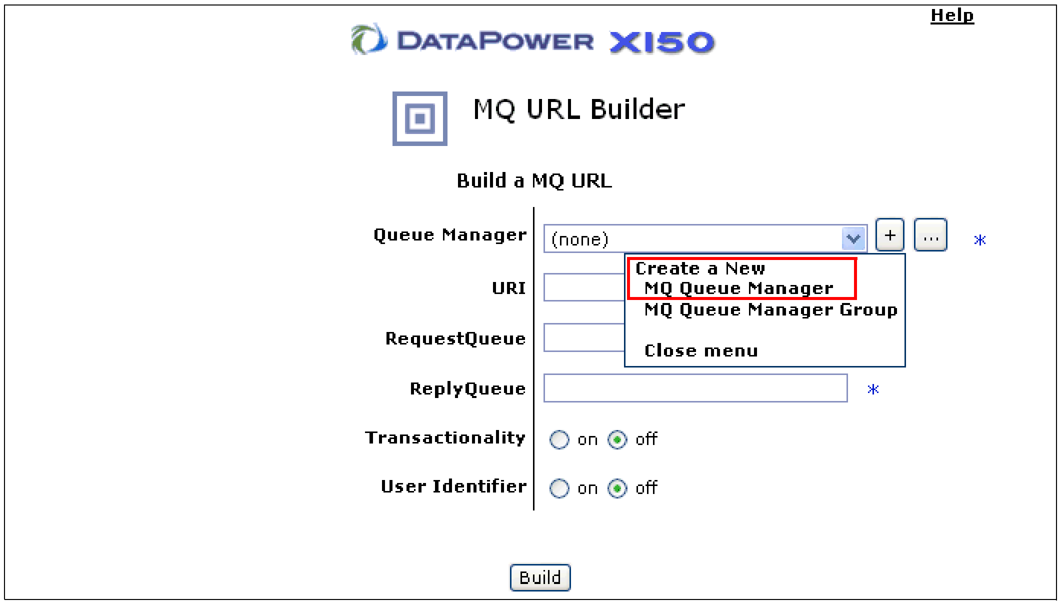
Front Side Protocol

Type	Name	Op-State
MQ Front Side Handler	WMQ_FrontSideHandler	[up]

MQ_FrontSideHandler

Figure 2-87   Configure Multi-Protocol Gateway: Back side settings

2. In Figure 2-88 on page 75, click the plus symbol (**+**) next to Queue Manager and select **Create a new MQ Queue Manager**.

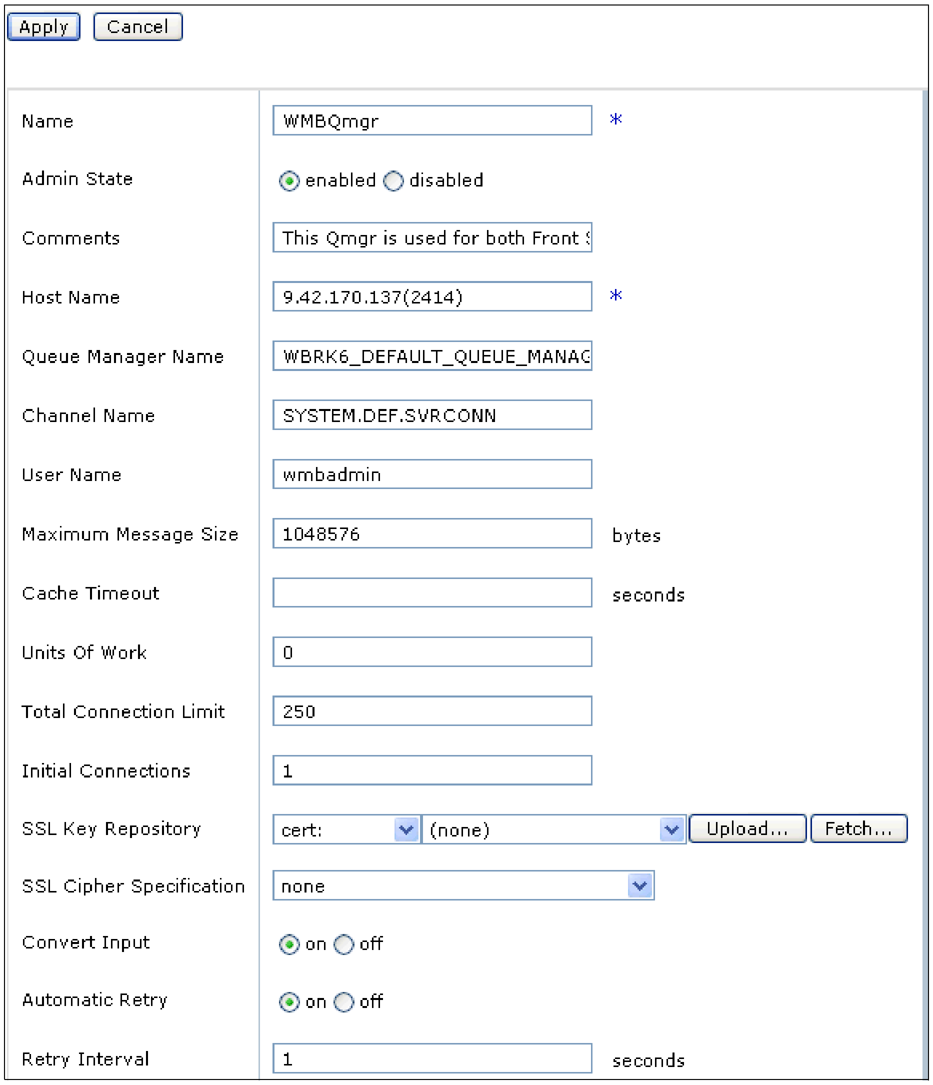


Transactionality: on [✓ off]

User Identifier: on [✓ off]

Figure 2-88 MQ URL Builder

3. Configure the queue manager for the MQ URL Builder. Fill in the information as shown in Figure 2-89 on page 76:
   a. Set Name to `WMBQmgr`.
   b. Set Host Name to the host name or the IP address where your queue manager is running, followed by a port number in parentheses. This is the port on which your queue manager is listening.
   c. Set Queue Manager Name to your queue manager name. We use the queue manager name `WBRK6_DEFAULT_QUEUE_MANAGER`.
   d. The Channel Name is prefilled with the system default channel `SYSTEM.DEF.SVRCONN`. Optionally, you can create a different server connection channel and use it here.


Apply | Cancel

Name	WMBQmgr *
Admin State	[✓ enabled] disabled
Comments	This Qmgr is used for both Front
Host Name	9.42.170.137(2414) *
Queue Manager Name	WBRK6_DEFAULT_QUEUE_MANAG
Channel Name	SYSTEM.DEF.SVRCONN
User Name	wmbadmin
Maximum Message Size	1048576 bytes
Cache Timeout	seconds
Units Of Work	0
Total Connection Limit	250
Initial Connections	1
SSL Key Repository	cert: (none) Upload... Fetch...
SSL Cipher Specification	none
Convert Input	[✓ on] off
Automatic Retry	[✓ on] off
Retry Interval	1 seconds

Figure 2-89 Configure the MQ URL

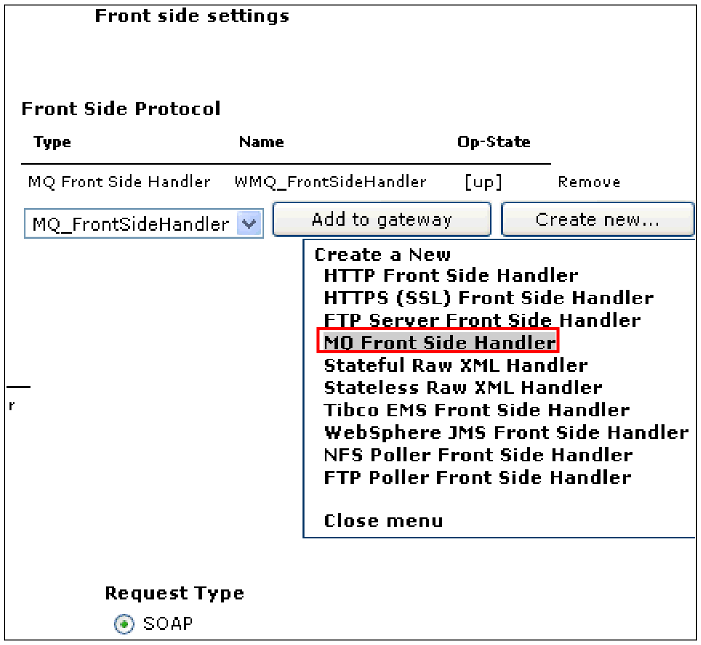
4. Click **Apply** at the top of the panel in Figure 2-89.
5. Back in the MQ URL Builder panel (Figure 2-88 on page 75), fill in or select the following information:
   - Queue Manager: `WMBQmgr`
   - URI: `URINotUsed`
   - RequestQueue: `SOA_REQUEST`
   - ReplyQueue: `SOA_REPLY`
   - Transactionality: **off**
   - User Identifier: **off**
6. Click **Build** at the bottom of the MQ URL Builder panel to finish and click **Apply.**
7. Verify that the **Backend URL** under the Back side settings section is created as shown in Example 2-1 on page 77.

Example 2-1 Generated Backend URL

```
dpmq://WMBQmgr/URINotUsed?RequestQueue=SOA_REQUEST;ReplyQueue=SOA_REPLY
```

Step 4: Configure the Front Side Handler

1. Create a new MQ Front Side Handler by doing the following steps:
   - In the **Front Side Setting** section, click **Create new.**
   - Select **MQ Front Side Handler** from the drop-down list, as shown in Figure 2-90.



Front side settings

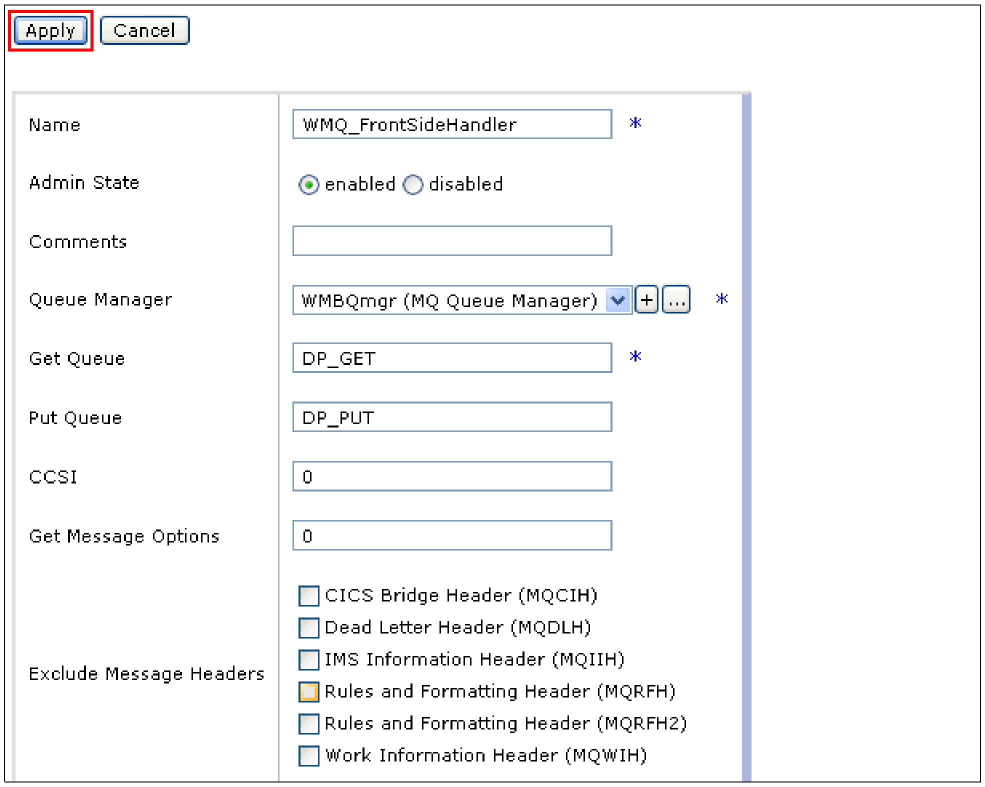
Front Side Protocol

Type	Name	Op-State	
MQ Front Side Handler	WMQ_FrontSideHandler	[up]	Remove

MQ_FrontSideHandler
Add to gateway
Create new...
Create a New
HTTP Front Side Handler
HTTPS (SSL) Front Side Handler
FTP Server Front Side Handler
MQ Front Side Handler
Stateful Raw XML Handler
Stateless Raw XML Handler
Tibco EMS Front Side Handler
WebSphere JMS Front Side Handler
NFS Poller Front Side Handler
FTP Poller Front Side Handler
Close menu
Request Type
SOAP

Figure 2-90 Create MQ Front Side Handler

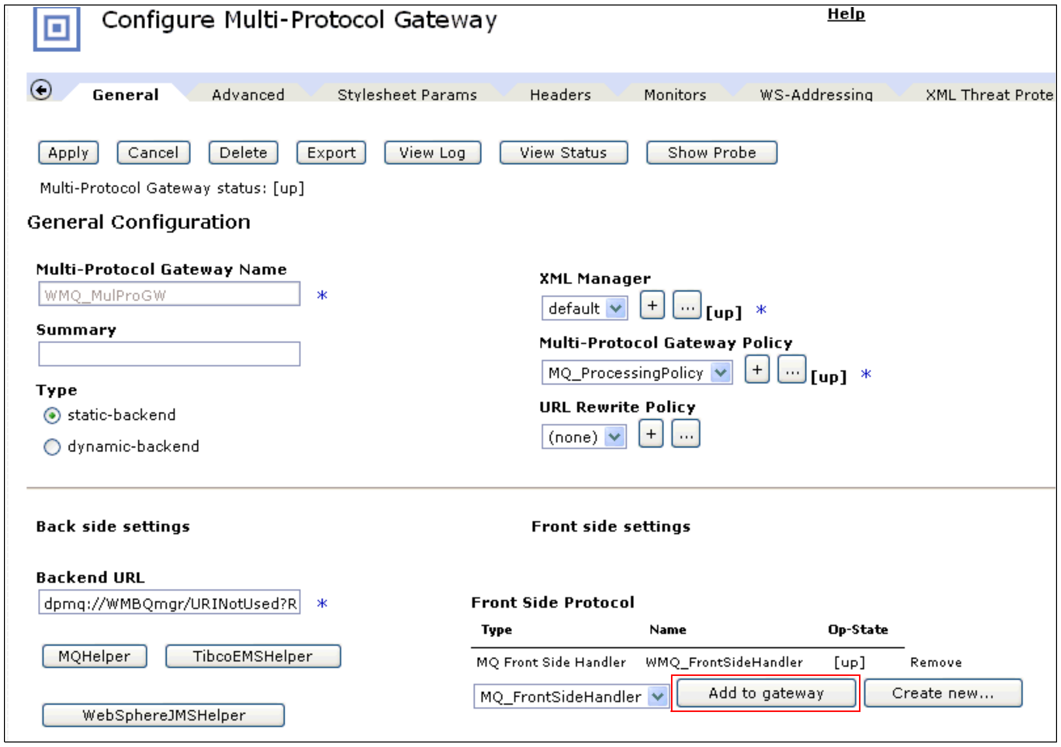
2. Configure the Front Side Handler by entering or selecting the following information in the Configure MQ Front Side Handler Main tab, which is shown in Figure 2-91 on page 78:
   - Enter Name: `WMQ_FrontSideHandler`.
   - Select **enabled** next to Admin State.
   - Set Queue Manager Name to **WMBQmgr** from the drop-down list.
   - Enter Get Queue `DP_GET`.
   - Enter Put Queue `DP_PUT`.
3. Click **Apply.**



Name	WMQ_FrontSideHandler *
Admin State	[✓ enabled] disabled
Comments	
Queue Manager	WMBQmgr (MQ Queue Manager) *
Get Queue	DP_GET *
Put Queue	DP_PUT
CCSI	0
Get Message Options	0
Exclude Message Headers	CICS Bridge Header (MQCIH) Dead Letter Header (MQDLH) IMS Information Header (MQIIH) Rules and Formatting Header (MQRFH) Rules and Formatting Header (MQRFH2) Work Information Header (MQWIH)

Figure 2-91 Front Side Handler configuration

4. Add the Front Side Handler to the gateway by clicking **Add to gateway** (Figure 2-92).



Multi-Protocol Gateway status: [up]

General Configuration

Multi-Protocol Gateway Name: WMQ_MulProGW *

Summary:

Type: [✓ static-backend] dynamic-backend

XML Manager: default [up] *

Multi-Protocol Gateway Policy: MQ_ProcessingPolicy [up] *

URL Rewrite Policy: (none)

Back side settings

Backend URL: dpmq://WMBQmgr/URINotUsed?R *

Front side settings

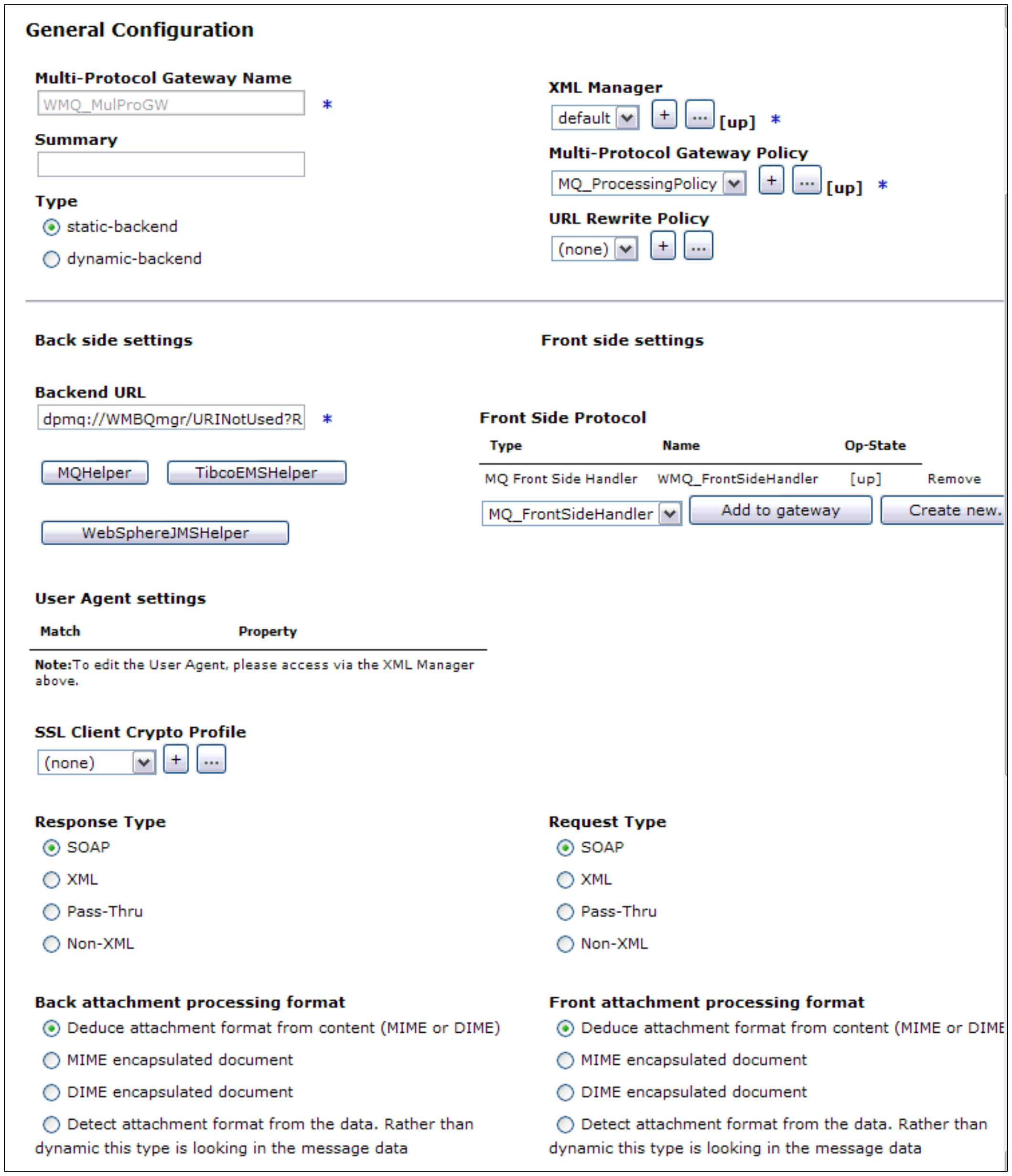
Front Side Protocol

Type	Name	Op-State	
MQ Front Side Handler	WMQ_FrontSideHandler	[up]	Remove

MQ_FrontSideHandler

Figure 2-92 Add Front Side Handler to Multi-Protocol Gateway

5. Back to the Configure Multi-Protocol Gateway panel, click **Apply** and **Save Config** to finish.
6. Verify that the Multi-Protocol Gateway is configured as shown in Figure 2-93.



General Configuration

Multi-Protocol Gateway Name
WMQ_MulProGW *

Summary

Type
(●) static-backend
( ) dynamic-backend

XML Manager
default [up] *

Multi-Protocol Gateway Policy
MQ_ProcessingPolicy [up] *

URL Rewrite Policy
(none)

Back side settings

Backend URL
dpmq://WMBQmgr/URINotUsed?R *

MQHelper | TibcoEMSHelper | WebSphereJMSHelper

Front side settings

Front Side Protocol

Type	Name	Op-State	
MQ Front Side Handler	WMQ_FrontSideHandler	[up]	Remove

MQ_FrontSideHandler | Add to gateway | Create new.

User Agent settings

Match	Property

Note: To edit the User Agent, please access via the XML Manager above.

SSL Client Crypto Profile
(none)

Response Type
(●) SOAP
( ) XML
( ) Pass-Thru
( ) Non-XML

Request Type
(●) SOAP
( ) XML
( ) Pass-Thru
( ) Non-XML

Back attachment processing format
(●) Deduce attachment format from content (MIME or DIME)
( ) MIME encapsulated document
( ) DIME encapsulated document
( ) Detect attachment format from the data. Rather than dynamic this type is looking in the message data

Front attachment processing format
(●) Deduce attachment format from content (MIME or DIME
( ) MIME encapsulated document
( ) DIME encapsulated document
( ) Detect attachment format from the data. Rather than dynamic this type is looking in the message data

Figure 2-93 WMB_MulProGW configuration

Note: You have completed the configuration of the DataPower Multi-Protocol Gateway for MQ protocol. You must stop here and test scenario two by following the instructions in "Testing scenario two" on page 92.

2.4.3 Running the scenarios

The DataPower appliance and WMB are configured to seamlessly integrate with one another according to the configuration shown in Figure 2-1 on page 19 or Figure 2-2 on page 20.

In this section, we show you how to test the XML firewall configuration used in scenario one and two. We explain how to do the following tasks:

1. Review the configuration for DataPower and the message broker.
2. Configure the DataPower XML Loopback firewall used to generate an encrypted message to test the XML Firewall Gateway created by the security wizard within WebSphere Message Broker Explorer.
3. Add the Sign action to the processing policy of the DataPower XML Loopback firewall to generate an encrypted and signed message to test the XML Firewall Gateway after the sign action was added to its existing processing policy.
4. Set up the DataPower Probe to perform problem determination.
5. Turn on the trace for the Message Broker.
6. Test scenario one and two using the prebuilt messages provided for testing.

Step 1: Review of the configuration for DataPower and WMB

This step ensures that the DataPower appliance and the message broker have the expected configuration as shown in Table 2-3. According to the table, all requests will go through the DataPower front end via the URLs set in the DataPower XML firewalls. DataPower connects to WebSphere Message Broker via the URLs set in the WebSphere Message Broker HTTP and HTTPS input nodes. We have provided many checkpoints for you to verify your configuration as you progress. This is more of a review of your configuration.

Table 2-3 Configuration

DataPower XML firewalls	**WMB HTTP and HTTPS input nodes**
HTTPS://<*DataPowerIP*>:12080	http://<*wmbIP*>:7080/HTTPNoSec/AccountStatus
HTTPS://<*DataPowerIP*>:12083	https://<*wmbIP*>:7083/HTTPSec/AccountStatus

Note: External systems always connect to DataPower via HTTP over SSL. Internally, DataPower connects to WebSphere Message Broker via HTTP or HTTPS.

Step 2: Create the loopback gateway and test scenario one

Note: We provide this section if you want to generate an encrypted and signed message yourself. You can use the prebuilt test messages to test your configuration. You can skip this step and go directly to “Testing scenario one” on page 90 to test the configuration of the XML Firewall Gateway or “Testing scenario two” on page 92 to test the configuration of the Multi-Protocol Gateway.

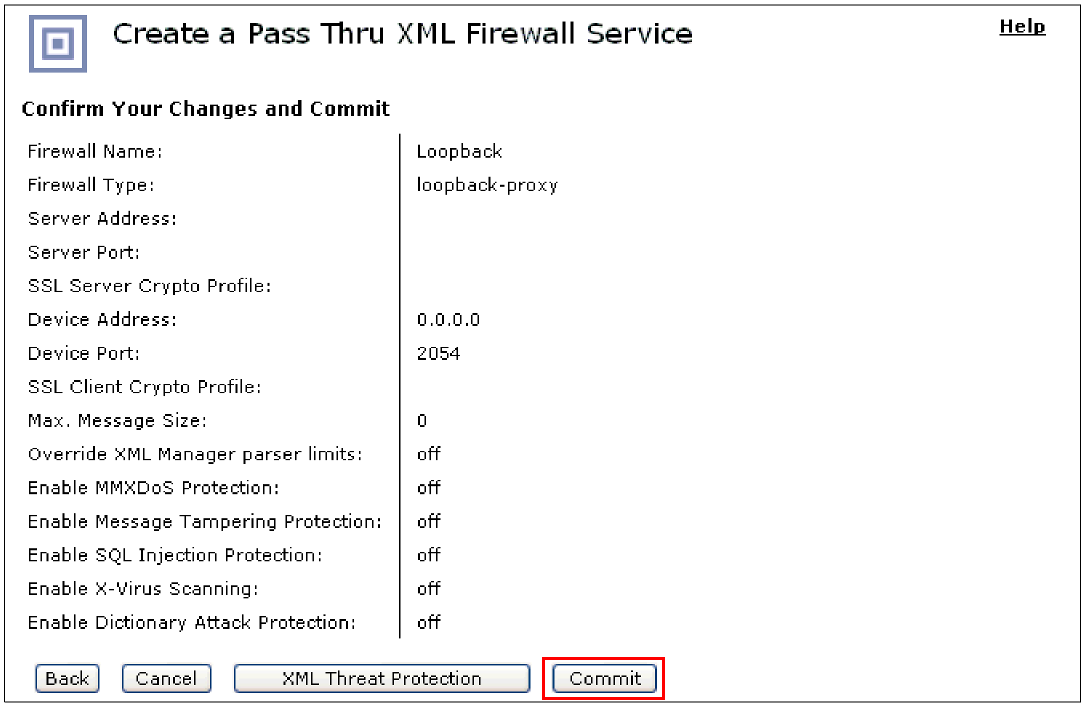
A loopback firewall processes request messages without a back-end server and does not require client credentials. A loopback firewall is used to create encrypted and signed messages using crypto settings configured for the firewalls. We use it to generate an encrypted and signed message to test the XML Firewall Gateway or the Multi-Protocol Gateway configuration. The major tasks are:

- Create a loopback firewall using the crypto settings created in initial DataPower settings.
- Create an encrypted message for testing the XML firewall gateway created by using the security wizard as described in 2.3.3, “Configure DataPower XML Firewall Gateway” on page 36.

- Create an encrypted and signed message for testing the XML firewall after the Sign action was added to the XML Firewall Gateway as explained in 2.3.4, “Configure DataPower to add support for Digital Signature” on page 57.

Perform these steps:

1. Create a loopback firewall:
   a. Click **Control Panel** → **XML FireWall** → **Add** Wizard.
   b. Select **Pass Thru** (testing only) and click **Next**.
   c. Enter the name `Loopback` in the Firewall Name field and click **Next**.
   d. Select the **loopback-proxy** from the Firewall Type drop-down list and click **Next**.
   e. Accept the default settings shown in Figure 2-94. Make sure that “Do you want to use SSL?” is **off** and click **Next**.
   f. Verify the settings as shown in Figure 2-94 and click **Commit.**

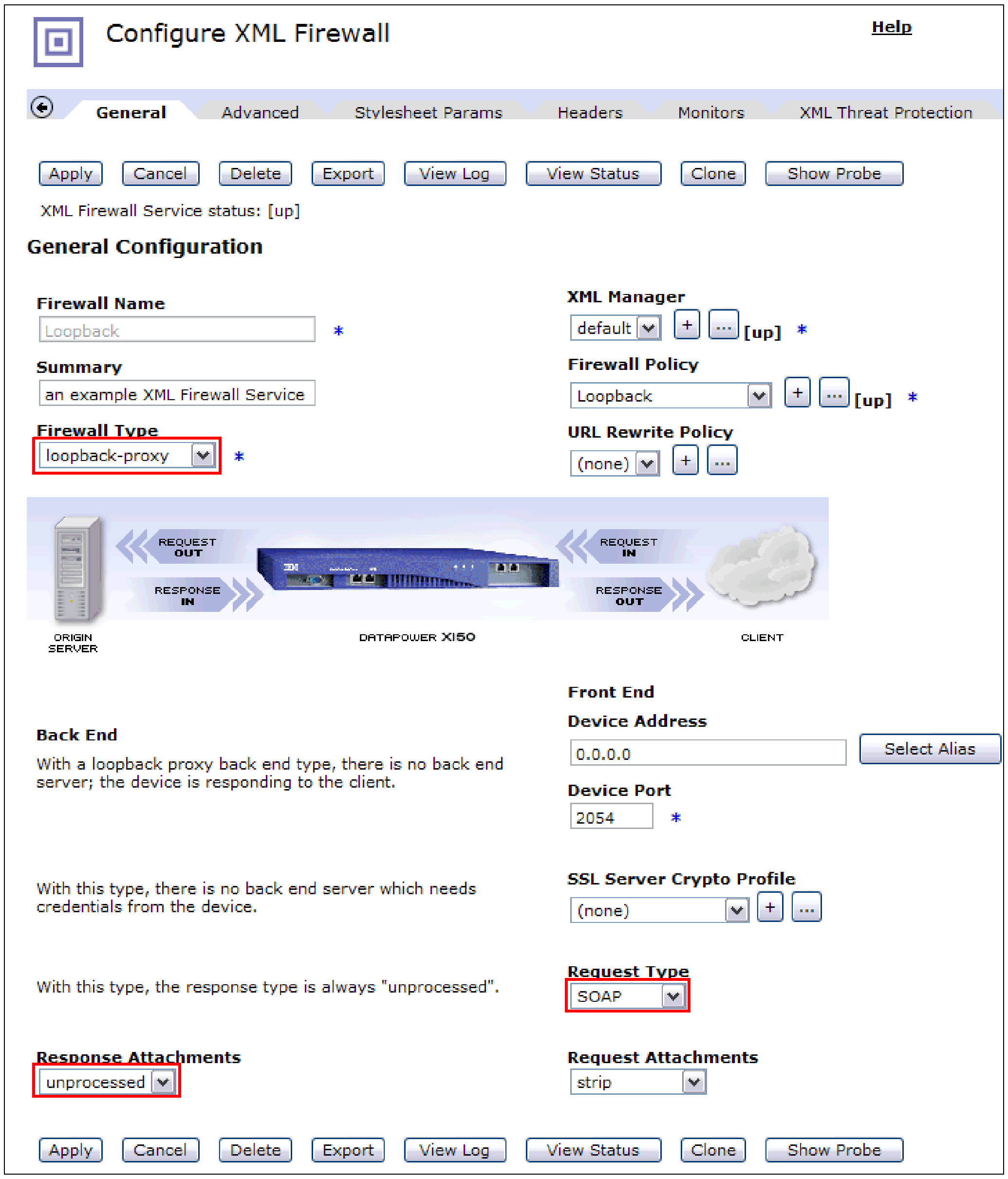


Confirm Your Changes and Commit

Firewall Name:	Loopback
Firewall Type:	loopback-proxy
Server Address:	
Server Port:	
SSL Server Crypto Profile:	
Device Address:	0.0.0.0
Device Port:	2054
SSL Client Crypto Profile:	
Max. Message Size:	0
Override XML Manager parser limits:	off
Enable MMXDoS Protection:	off
Enable Message Tampering Protection:	off
Enable SQL Injection Protection:	off
Enable X-Virus Scanning:	off
Enable Dictionary Attack Protection:	off

Figure 2-94 Create a Pass Thru XML Firewall Service

   g. Click **Done** in the Create a Pass Thru XML Firewall Service pane.
   h. Back on the Configure XML Firewall pane, select **unprocessed** (drop-down list) under Response Attachments.
   i. Click **Apply** at the top of the panel and then **Save Config** to save the configuration.
   j. Verify that the Loopback XML Firewall is configured as shown in Figure 2-95 on page 82.



XML Firewall Service status: [up]

General Configuration

Firewall Name: Loopback *

Summary: an example XML Firewall Service

Firewall Type: loopback-proxy *

XML Manager: default [up] *

Firewall Policy: Loopback [up] *

URL Rewrite Policy: (none)

Back End

With a loopback proxy back end type, there is no back end server; the device is responding to the client.

Front End

Device Address: 0.0.0.0

Device Port: 2054 *

With this type, there is no back end server which needs credentials from the device.

SSL Server Crypto Profile: (none)

With this type, the response type is always "unprocessed".

Request Type: SOAP

Response Attachments: unprocessed

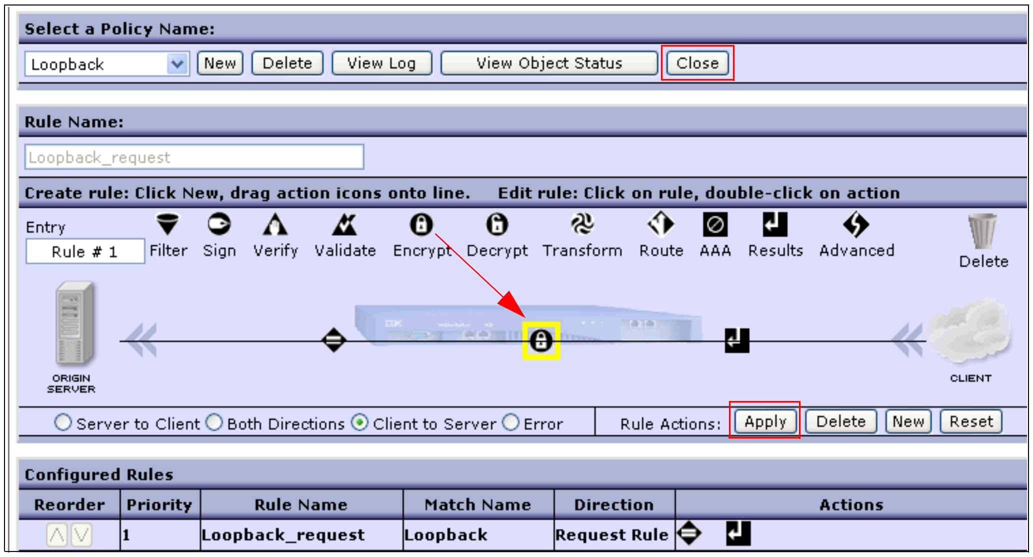
Request Attachments: strip

Figure 2-95 Loopback XML Firewall configuration

2. Add message encryption and decryption to the Loopback XML Firewall policies.

   a. In the Configure XML Firewall panel, double-click the ellipsis (**...**) next to Firewall Policy to edit the processing policy for the Loopback firewall.

   b. Drag and drop the **Encrypt** action to the configuration path as shown in Figure 2-96 on page 83.



Reorder	Priority	Rule Name	Match Name	Direction	Actions
	1	Loopback_request	Loopback	Request Rule	

Figure 2-96 Add Encrypt action to the request rule

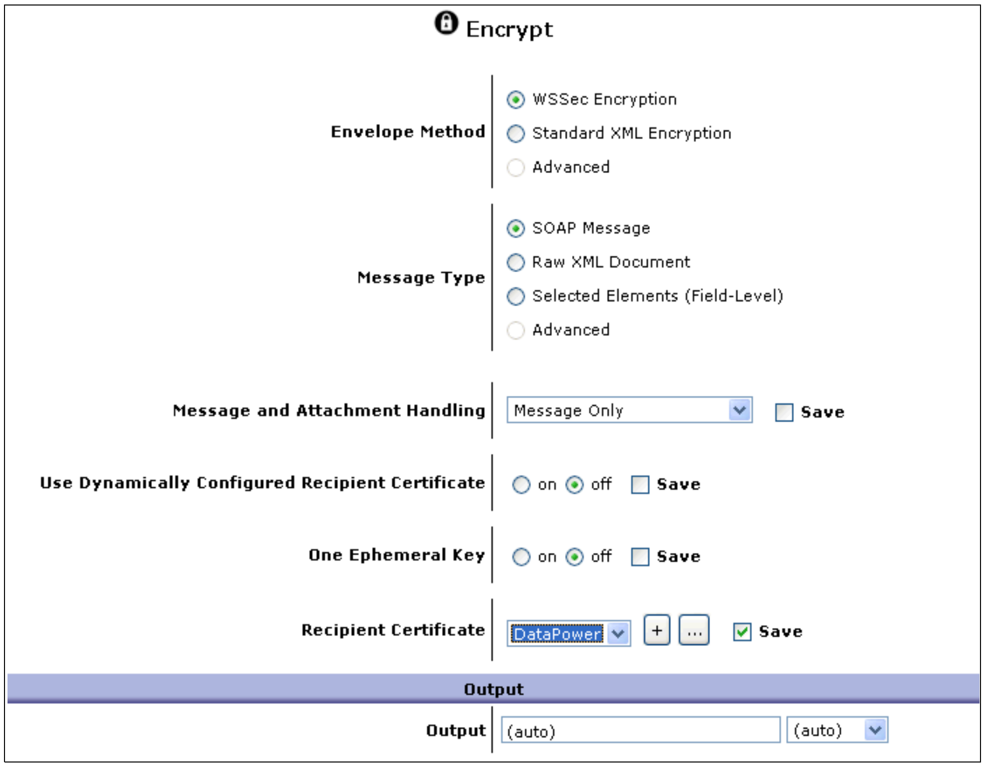
c. Double-click the **Encrypt** action (highlighted in a yellow box) to add the action rule, select the information as shown in Figure 2-97, and click **Done**:

i. Select **WSSec Encryption** next to Envelope Method.

ii. Select **SOAP Message** next to Message Type.

iii. Select **DataPower** from the Recipient Certificate drop-down list.

iv. Set Output to **auto** from the drop-down list.



Encrypt

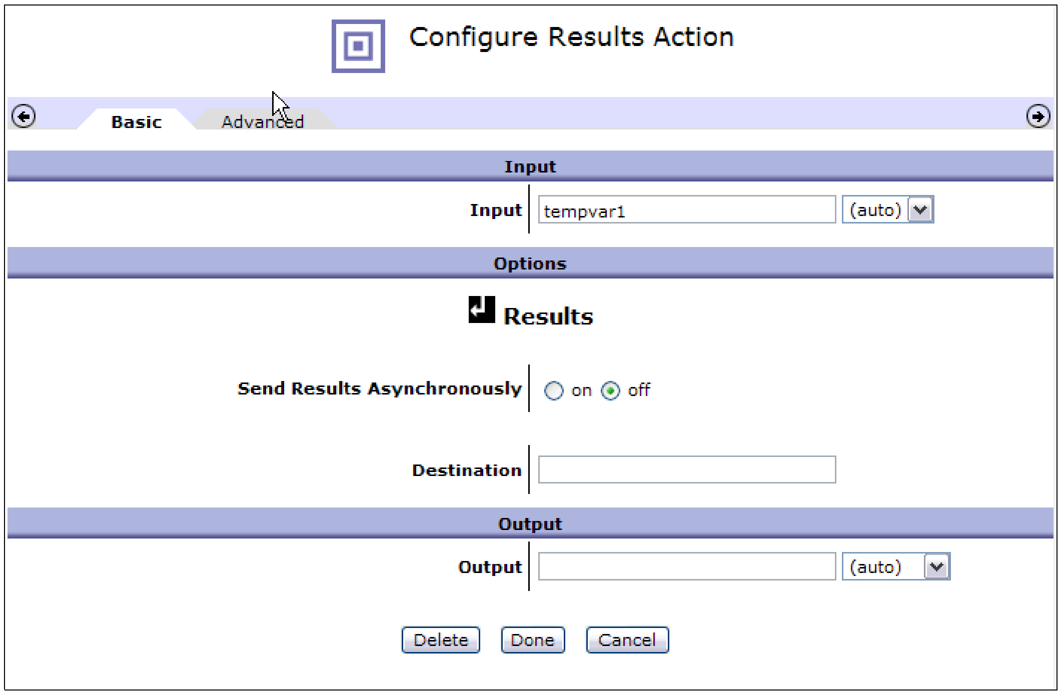
Envelope Method	[✓ WSSec Encryption] Standard XML Encryption Advanced
Message Type	[✓ SOAP Message] Raw XML Document Selected Elements (Field-Level) Advanced
Message and Attachment Handling	Message Only ☐ Save
Use Dynamically Configured Recipient Certificate	on [✓ off] ☐ Save
One Ephemeral Key	on [✓ off] ☐ Save
Recipient Certificate	DataPower + ... ☑ Save

Output

Output	(auto) (auto)

Figure 2-97 Encryption action configuration

3. Double-click **Results** to configure the Results action as shown in Figure 2-98:
   a. Select **auto** from the drop-down list. It will automatically set Input to tempvar1.
   b. Click **Done**.



Results

Input: tempvar1 (auto)

Send Results Asynchronously: on [✓ off]

Destination:

Output: (auto)

Figure 2-98 Results action configuration

   c. Click **Apply** next to Rule Actions.
   d. Click **Close** next to View Object Status.
   e. Back on the Configure XML Firewall Policy panel, click **Apply** and click **Save Config**.

   You have configured the loopback XML firewall. You will use this XML firewall to generate the encrypted message to test the `DP2XML_HTTP` and the `DP2XML_HTTPS` XML firewalls, which were generated by the security wizard.

4. Download **`curl`** before you run the scripts. You can download **`curl`** from the additional materials link.

   Note: **`curl`** is a free command line tool for sending files with URL syntax. **`curl`** supports SSL connection with certificates, HTTP, HTTPS, HTTP form-based upload, user and password authentication, proxies, cookies, and other protocols such as FTP, TELNET, and so forth.

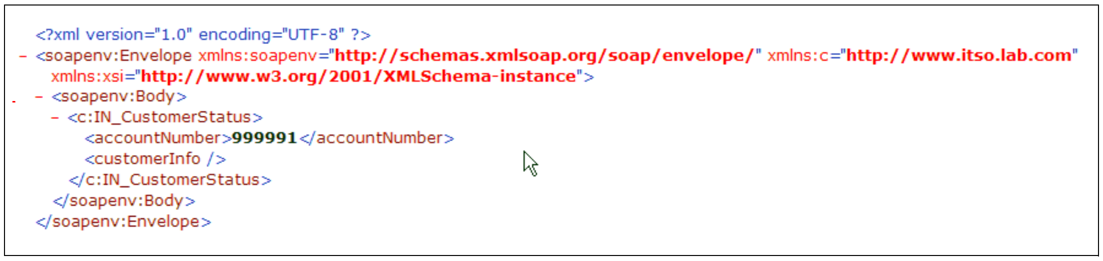
5. Generate an encrypted message to test the XML firewalls created by the security wizard within WebSphere Message Broker.

   Note: You must edit the Test_GenEncryptSOAP.cmd to change the IP address (`9.42.170.230`) and port (`2054`) to match the address and the port that were configured for the loopback firewall gateway that you have just created.

Run the script Test_GenEncryptSOAP.cmd to create an encrypted message in the file TestEncryptSOAP_IN.xml**.** This script executes the following command:

```
curl -v --data-binary @TestRegularSOAP_IN.xml http://9.42.170.230:2054 >
TestEncryptSOAP_IN.xml
```

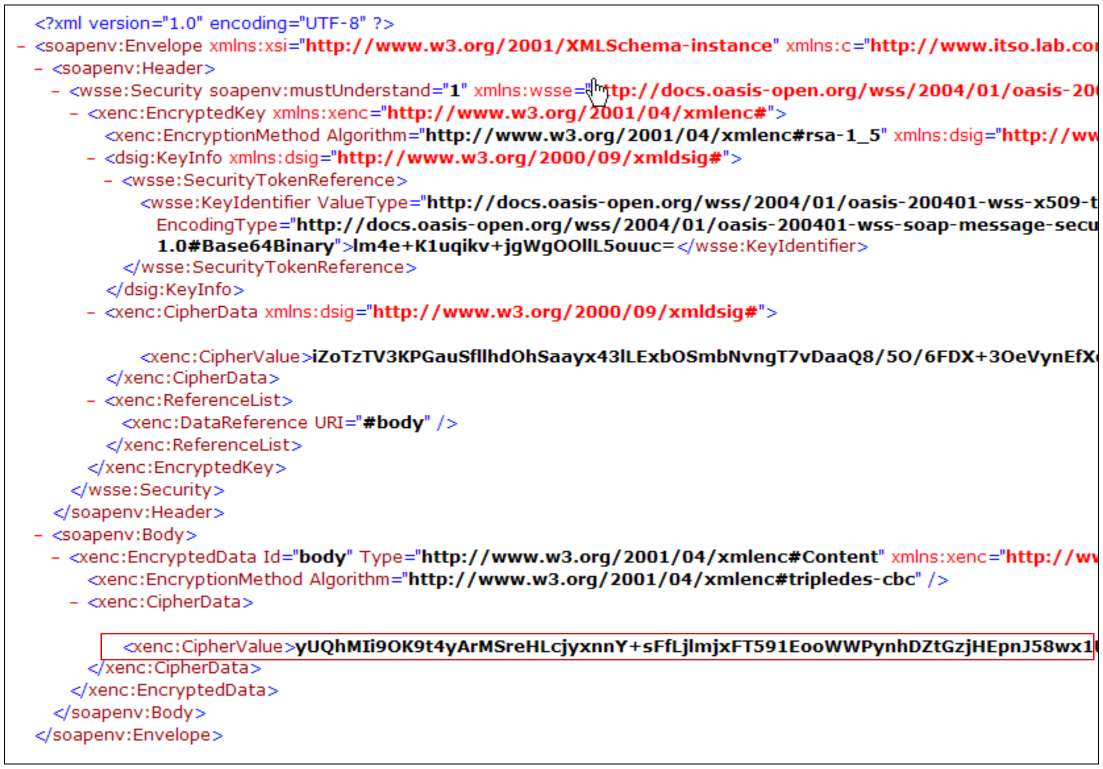
The input message, TestRegularSOAP_IN.xml, is shown in Figure 2-99.



```
<?xml version="1.0" encoding="UTF-8" ?>
- <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:c="http://www.itso.lab.com"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <soapenv:Body>
    - <c:IN_CustomerStatus>
        <accountNumber>999991</accountNumber>
        <customerInfo />
      </c:IN_CustomerStatus>
    </soapenv:Body>
  </soapenv:Envelope>
```

Figure 2-99 Input message

The generated output message is encrypted and written to TestEncryptSOAP_IN.xml. This sample encrypted message is shown in Figure 2-100. The body of the data is encrypted as highlighted in the red rectangle.



```
<?xml version="1.0" encoding="UTF-8" ?>
- <soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:c="http://www.itso.lab.co
  - <soapenv:Header>
    - <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-20
      - <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
          <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" xmlns:dsig="http://ww
        - <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
          - <wsse:SecurityTokenReference>
              <wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-t
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-secu
                1.0#Base64Binary">lm4e+K1uqikv+jgWgOOllL5ouuc=</wsse:KeyIdentifier>
            </wsse:SecurityTokenReference>
          </dsig:KeyInfo>
        - <xenc:CipherData xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">

              <xenc:CipherValue>iZoTzTV3KPGauSfllhdOhSaayx43lLExbOSmbNvngT7vDaaQ8/5O/6FDX+3OeVynEfX
          </xenc:CipherData>
        - <xenc:ReferenceList>
            <xenc:DataReference URI="#body" />
          </xenc:ReferenceList>
        </xenc:EncryptedKey>
      </wsse:Security>
    </soapenv:Header>
  - <soapenv:Body>
    - <xenc:EncryptedData Id="body" Type="http://www.w3.org/2001/04/xmlenc#Content" xmlns:xenc="http://ww
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      - <xenc:CipherData>

            <xenc:CipherValue>yUQhMIi9OK9t4yArMSreHLcjyxnnY+sFfLjlmjxFT591EooWWPynhDZtGzjHEpnJ58wx1
        </xenc:CipherData>
      </xenc:EncryptedData>
    </soapenv:Body>
  </soapenv:Envelope>
```

Figure 2-100 Encrypted message

6. Run the script Test_EncryptSOAP_DP_HTTP.cmd to test the XML Firewall `DP2WMB_HTTP`. You must edit the script to change the IP address and port to point to your `DP2WMB_HTTP` firewall. This script executes the following command:

```
curl -k -v --data-binary @C:\ITSO_DP\Data\TestEncryptSOAP_IN.xml
https://9.42.170.230:12080/HTTPNoSec/AccountStatus
```

7. Run the script Test_EncryptSOAP_DP_HTTPS.cmd to test the XML firewall `DP2WMB_HTTP`. You must edit the script to change the IP address and port to point to your `DP2WMB_HTTP` firewall. This script executes the following command:

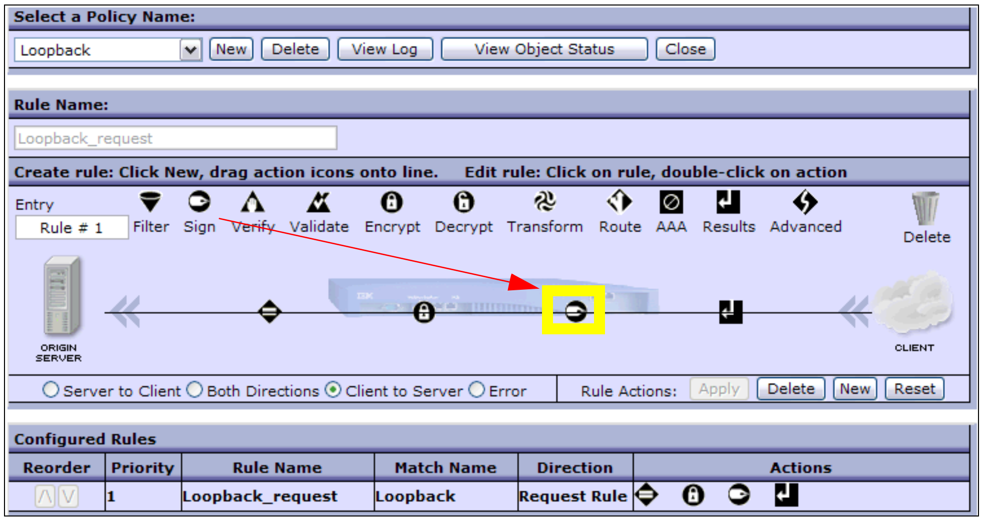
```
curl -k -v --data-binary @TestEncryptSOAP_IN.xml
https://9.42.170.230:12083/HTTPSec/AccountStatus
```

Note: You must go back to 2.3.4, "Configure DataPower to add support for Digital Signature" on page 57 if you want to add digital signature to the SOAP message.

Step 3: Add digital signature to the loopback firewall and test

The loopback firewall currently supports message encryption and decryption. Optionally, you can add the support for digital signature to the loopback firewall and use it to generate an encrypted and signed message to test the `DP2WMB_HTTP` and `DP2WMB_HTTPS` XML firewalls:

1. Add the Sign action to the processing policy of the Loopback XML firewall:
   a. Click **Control Panel** → **XML Firewall** → **Loopback**.
   b. In the Configure XML Firewall panel, double-click the ellipsis (**...**) next to Firewall Policy to edit the processing policy for the Loopback firewall.
   c. Drag and drop the **Sign** action on the configuration path of the processing policy as shown in Figure 2-101.

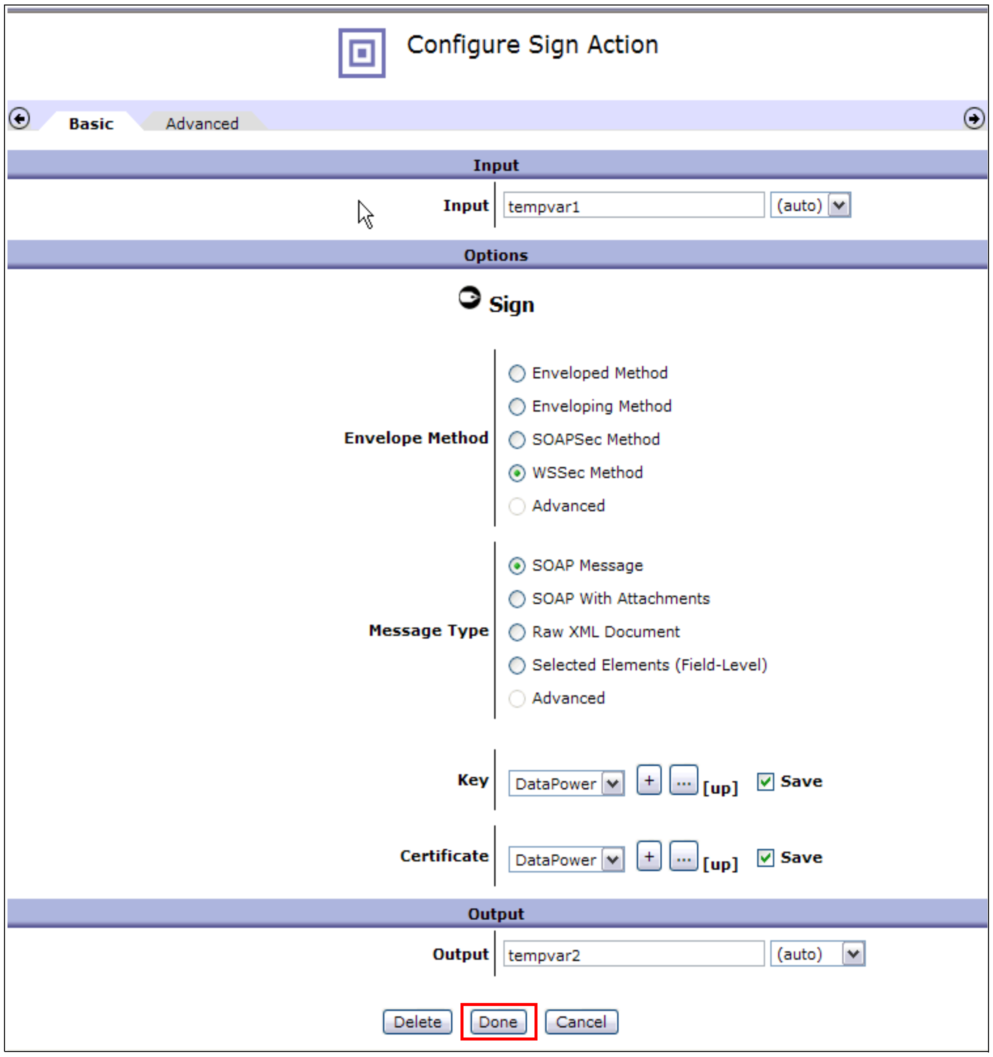


Configured Rules

Reorder	Priority	Rule Name	Match Name	Direction	Actions
	1	Loopback_request	Loopback	Request Rule	

Figure 2-101 Add Sign action

   d. Double-click the **Sign** icon (highlighted in a yellow box) to add the action rule.
   e. Select the information, as shown in Figure 2-102 on page 87:
      i. For Envelope Method, select the signature type: **WSSec Method**. *WSSec* means that the signature is included in a WS Security security header.
      ii. Select **SOAP Message** to set the value of Message Type. SOAP message tells the Loopback firewall gateway to sign an entire SOAP message.
      iii. Set the Crypto Key to **DataPower**.
      iv. Set the Certificate to **DataPower**.
      v. Set Output to **auto** from the drop-down list.



Input: tempvar1 (auto)

Envelope Method: ( ) Enveloped Method ( ) Enveloping Method ( ) SOAPSec Method [✓ WSSec Method] ( ) Advanced

Message Type: [✓ SOAP Message] ( ) SOAP With Attachments ( ) Raw XML Document ( ) Selected Elements (Field-Level) ( ) Advanced

Key: DataPower [up] ✓ Save

Certificate: DataPower [up] ✓ Save

Output: tempvar2 (auto)

Figure 2-102 Sign Action configuration

f. Click **Done.**

g. In the Configure XML Firewall Policy panel, click **Apply** next to the Rule Actions and click **Close**.

h. Back on the Configure XML Firewall panel, click **Apply** and click **Save Config** to save the configuration.

2. Generate the encrypted and signed message by executing the Test_GenEncSignSOAP.cmd script to create an encrypted and signed message.

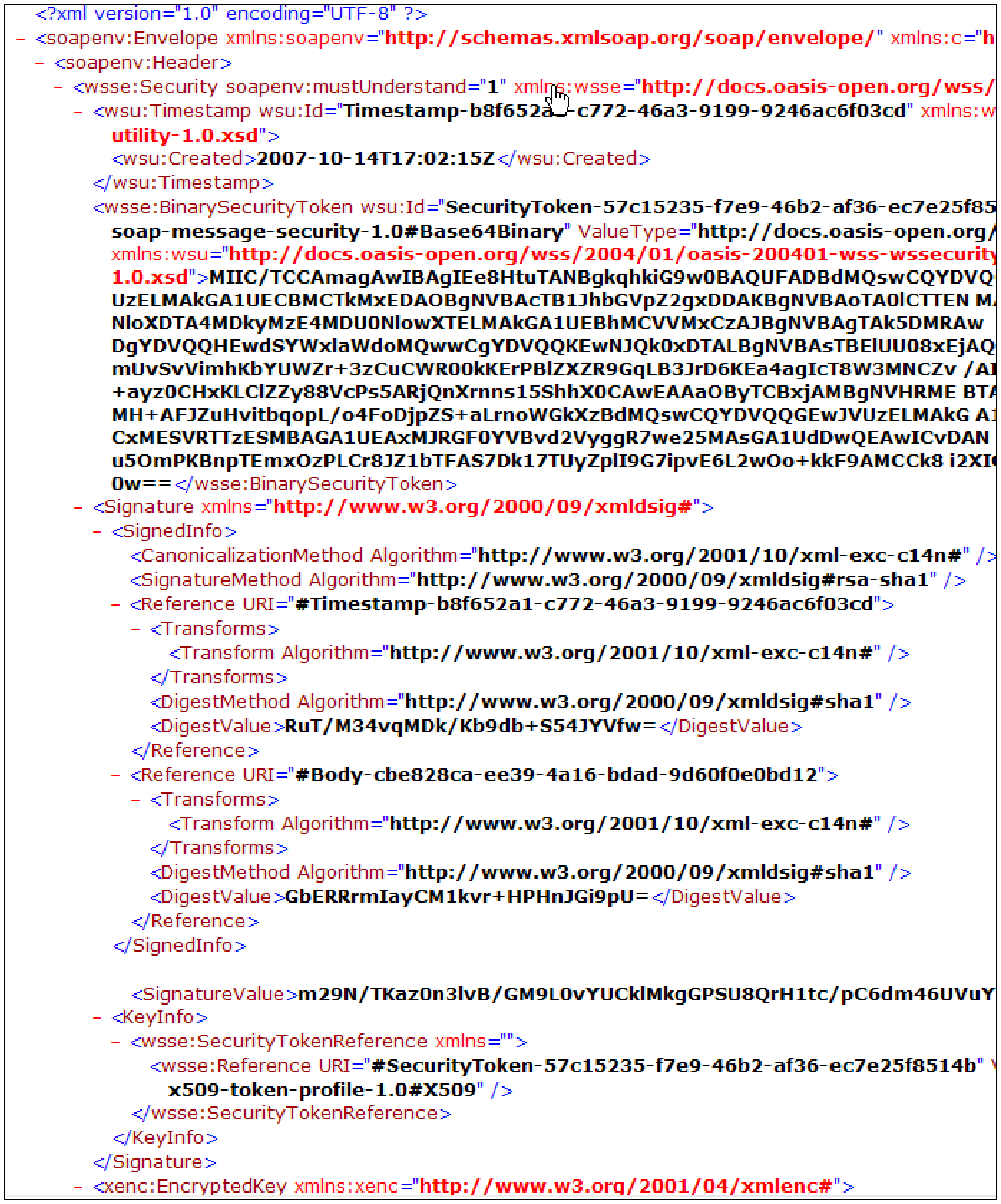
Note: You must edit the Test_genEncSignSOAP.cmd script to change the IP address (`9.42.170.230`) and Port (`2054`) to the ones configured for the Loopback XML FireWall.

The Test_GenEncSignSOAP.cmd script executes the following sample command to generate the encrypted and signed message using the Loopback XML Firewall:

```
curl -v --data-binary @C:\ITSO_DP\Data\TestRegularSOAP_IN.xml
http://9.42.170.230:2054 > C:\ITSO_DP\Data\TestEncSignSOAP_IN.xml
```

The command uses the same input message TestRegularSOAP_IN and generates an encrypted and signed message in TestEncSignSOAP_IN.xml.

The generated output message is encrypted and signed. It is written to TestEncSignSOAP_IN.xml. The encrypted and signed message is shown in Figure 2-103.



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:c="h
  - <soapenv:Header>
    - <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/
      - <wsu:Timestamp wsu:Id="Timestamp-b8f652a -c772-46a3-9199-9246ac6f03cd" xmlns:w
          utility-1.0.xsd">
          <wsu:Created>2007-10-14T17:02:15Z</wsu:Created>
        </wsu:Timestamp>
        <wsse:BinarySecurityToken wsu:Id="SecurityToken-57c15235-f7e9-46b2-af36-ec7e25f85
          soap-message-security-1.0#Base64Binary" ValueType="http://docs.oasis-open.org/
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
          1.0.xsd">MIIC/TCCAmagAwIBAgIEe8HtuTANBgkqhkiG9w0BAQUFADBdMQswCQYDVQ
          UzELMAkGA1UECBMCTkMxEDAOBgNVBAcTB1JhbGVpZ2gxDDAKBgNVBAoTA0lCTTEN M
          NloXDTA4MDkyMzE4MDU0NlowXTELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAk5DMRAw
          DgYDVQQHEwdSYWxlaWdoMQwwCgYDVQQKEwNJQk0xDTALBgNVBAsTBElUU08xEjAQ
          mUvSvVimhKbYUWZr+3zCuCWR00kKErPBlZXZR9GqLB3JrD6KEa4agIcT8W3MNCZv /A
          +ayz0CHxKLClZZy88VcPs5ARjQnXrnns15ShhX0CAwEAAaOByTCBxjAMBgNVHRME BT
          MH+AFJZuHvitbqopL/o4FoDjpZS+aLrnoWGkXzBdMQswCQYDVQQGEwJVUzELMAkG A
          CxMESVRTTzESMBAGA1UEAxMJRGF0YVBvd2VyggR7we25MAsGA1UdDwQEAwICvDAN
          u5OmPKBnpTEmxOzPLCr8JZ1bTFAS7Dk17TUyZplI9G7ipvE6L2wOo+kkF9AMCCk8 i2XI
          0w==</wsse:BinarySecurityToken>
      - <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        - <SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
          - <Reference URI="#Timestamp-b8f652a1-c772-46a3-9199-9246ac6f03cd">
            - <Transforms>
                <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
              </Transforms>
              <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
              <DigestValue>RuT/M34vqMDk/Kb9db+S54JYVfw=</DigestValue>
            </Reference>
          - <Reference URI="#Body-cbe828ca-ee39-4a16-bdad-9d60f0e0bd12">
            - <Transforms>
                <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
              </Transforms>
              <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
              <DigestValue>GbERRrmIayCM1kvr+HPHnJGi9pU=</DigestValue>
            </Reference>
          </SignedInfo>

            <SignatureValue>m29N/TKaz0n3lvB/GM9L0vYUCklMkgGPSU8QrH1tc/pC6dm46UVuY
        - <KeyInfo>
          - <wsse:SecurityTokenReference xmlns="">
              <wsse:Reference URI="#SecurityToken-57c15235-f7e9-46b2-af36-ec7e25f8514b" V
                x509-token-profile-1.0#X509" />
            </wsse:SecurityTokenReference>
          </KeyInfo>
        </Signature>
      - <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
```

Figure 2-103 Encrypted and signed test message

3. Run the script Test_EncSign_SOAP_DP_HTTP.cmd to test the `DP2WMB_HTTP` firewall with an encrypted and signed message. This script executes the following command:

```
curl -k -v --data-binary @C:\ITSO_DP\Data\TestEncSignSOAP_IN.xml
https://9.42.170.230:12080/HTTPNoSec/AccountStatus
```

4. Run the script Test_EncSign_SOAP_DP_HTTPS.cmd to test the `DP2WMB_HTTPS` firewall with an encrypted and signed message. This is a sample of the command that it executes:

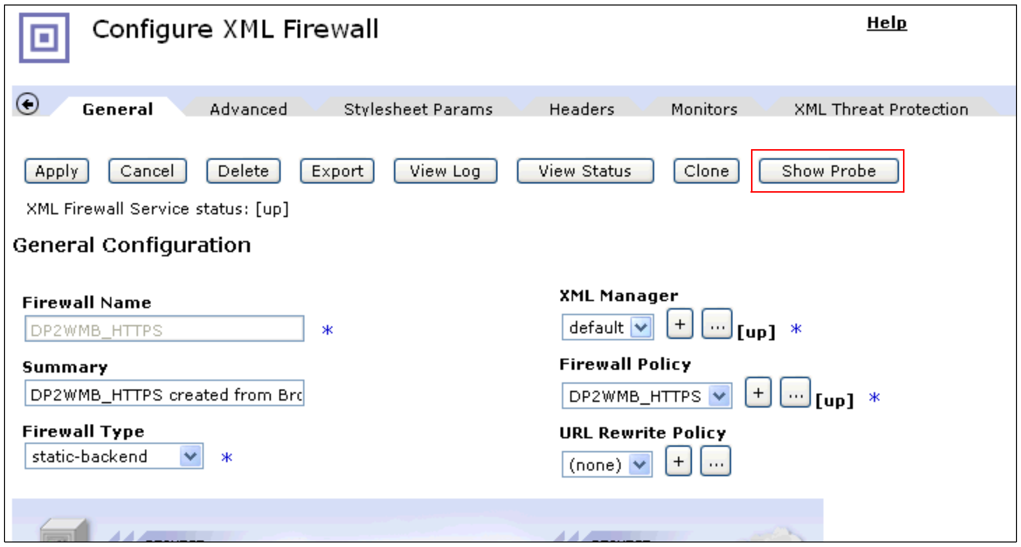
```
curl -k -v --data-binary @C:\ITSO_DP\Data\TestEncSignSOAP_IN.xml
https://9.42.170.230:12083/HTTPSec/AccountStatus
```

Step 4: Set up the DataPower Probe to trace the messages

It is important to know basic problem determination for DataPower. DataPower provides a valuable tool called the *Probe* for testing and problem determination. The Probe traces messages at each step of the processing rules. For example, it shows you the request, response, and timestamps before and after encryption and decryption rules, the reason for the error, and so forth. It is not possible to test the scenario without turning on the Probe.

1. Turn on the Probe by following these steps:

   a. Click **Control Panel** → **XML Firewall** → *XML Firewall Name* → **Show Probe** → **Enable Probe** where the XML Firewall Name is `DB2WMB_HTTP` or `DB2WMB_HTTPS`.



XML Firewall Service status: [up]

General Configuration

Firewall Name: DP2WMB_HTTPS *

Summary: DP2WMB_HTTPS created from Brc

Firewall Type: static-backend *

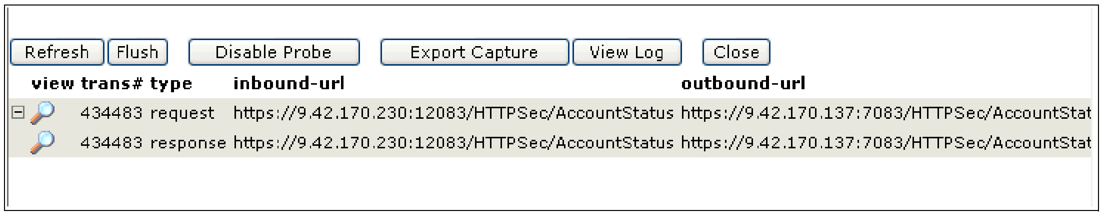
XML Manager: default [up] *

Firewall Policy: DP2WMB_HTTPS [up] *

URL Rewrite Policy: (none)

Figure 2-104 Enable Probe

   b. Click **Show Probe** and click **Enable Probe** to turn on the Probe. After the Probe is enabled and a valid or invalid message is sent to DataPower, the request and response will be shown in the Probe pane as shown in Figure 2-105.



```
view trans# type     inbound-url                                        outbound-url
     434483 request  https://9.42.170.230:12083/HTTPSec/AccountStatus https://9.42.170.137:7083/HTTPSec/AccountStat
     434483 response https://9.42.170.230:12083/HTTPSec/AccountStatus https://9.42.170.137:7083/HTTPSec/AccountStat
```

Figure 2-105 Probe report

   c. Click the magnifying glass icon under **view** to see the details.

In addition to the Probe, you can view the system logs:

1. Click the blue link **Troubleshooting Enabled (The performance of the device may be impacted)** on the top of the pane.

2. In the Troubleshooting panel, click the magnifying glass in the Logging section to view the system log.

Step 5: Set up the message broker to trace the message flows

It is also important to know basic problem determination for the message broker in the event of errors. You can look at the Windows Event Viewer for a WebSphere Message Broker error log or turn on the message broker trace by entering the following commands:

1. On Windows XP, browse the Event View Application log:

   Click **Start** → **Control Panel** → **Performance and Maintenance** → **Administrative tool** → **Event Viewer** → **Application**.

2. Turn on the message broker user trace by entering the following commands in the WebSphere Message Broker command console:

   a. `mqsichangetrace <Broker Name> -u -e DPSoa -l debug`

      In this command, *-u* is the user trace, *-e* is the execution group, and *-l* is the trace level.

   b. `mqsireadlog <Broker Name> -u -e DPSoa -o trace.xml`

   c. `mqsiformatlog -i trace.xml -o trace.txt`

   Browse the trace.txt file to search for errors.

Testing scenario one

We explain how to test the configuration of the DataPower XML Firewall and the integration of DataPower and WebSphere Message Broker via HTTP and HTTPS protocols. We provide the scripts to test the scenarios. We use the `curl` utility to test the Web services provided by the message broker.

Step one: Test WMB configuration for HTTP and HTTPS

You must first ensure that the message broker is configured correctly for HTTP and HTTPS connection and is operational before testing the end-to-end connectivity from the requesting client to DataPower and WebSphere Message Broker.

Note: You must edit the scripts and change the IP address and port of the DataPower XML Firewall and WebSphere Message Broker before running them. Both scripts in this step use the same input message and receive the same output message. None of these messages is encrypted.

The input SOAP message is shown in Example 2-2.

Example 2-2 Input non-encrypted SOAP message

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:c="http://www.itso.lab.com"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <c:IN_CustomerStatus>
         <accountNumber>999991</accountNumber>
                  <customerInfo>
         </customerInfo>
      </c:IN_CustomerStatus>
   </soapenv:Body>
```

```
</soapenv:Envelope>
```

The correct output message is shown in Example 2-3.

Example 2-3 Output non-encrypted SOAP message

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:tns="http://www.itso.lab.com"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:mrm="http://tempuri.org/SOA_XML_Request_Reply_MsgSet"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
   <tns:IN_CustomerStatus>
    <accountNumber>999991</accountNumber>
    <customerInfo>
      <status>P</status>
      <firstName>John</firstName>
      <lastName>Doe</lastName>
      <street>2345 Harriman Street</street>
      <city>Great Falls</city>
      <state>VA</state>
      <country>USA</country>
      <zipCode>22066</zipCode>
      <phone>703-111-9999</phone>
    </customerInfo>
   </tns:IN_CustomerStatus>
  </soapenv:Body>
 </soapenv:Envelope>
```

Follow these steps:

1. Run the script C:\ITSO_DP\Scripts\Test_RegularSOAP_MB_HTTP.cmd to test the HTTP connection and message flow. This script sends a non-encrypted SOAP message to the HTTP listener (port 7080) running in the message broker via the URL HTTPNoSec/AccountStatus. The sample script is:

   ```
   curl --data-binary @C:\ITSO_DP\Data\TestRegularSOAP_IN.xml
   http://localhost:7080/HTTPNoSec/AccountStatus
   ```

2. Run the script Test_RegularSOAP_MB_HTTPS.cmd to test the HTTP over SSL connection and message flow. This script sends a non-encrypted SOAP message to the HTTPS listener (port 7083) running in the message broker via the URL HTTPSec/AccountStatus. The sample script is:

   ```
   curl -k -v --data-binary @C:\ITSO_DP\Data\TestRegularSOAP_IN.xml
   https://localhost:7083/HTTPSec/AccountStatus
   ```

Step two: Test XML firewalls using the prebuilt, encrypted message

This step provides instructions for testing the XML Firewall Gateways within DataPower using a prebuilt, encrypted message that we have provided:

1. Run the C:\ITSO_DP\Scripts\Test_EncryptSOAP_DP_HTTP.cmd to test an encrypted SOAP message. This script sends an encrypted SOAP message TestEncryptSOAP_IN.xml message to the XML Firewall Gateway for the HTTP connection via the IP address 9.42.170.230 and port 12080. This IP address and port are configured using the security wizard. On the inbound transaction, DataPower decrypts the request SOAP message before sending it to WebSphere Message Broker. On the

outbound transaction, DataPower encrypts the response that it gets from WebSphere Message Broker before sending it to the client. The command used by this script is listed in the following example:

```
curl -k -v --data-binary @C:\ITSO_DP\Data\TestEncryptSOAP_IN.xml
https://9.42.170.230:12080/HTTPNoSec/AccountStatus
```

2. Run the script C:\ITSO_DP\Scripts\Test_EncryptSOAP_DP_HTTPS.cmd. This script provides the same functions as the script in step 1, except that it uses the HTTP over an SSL connection and a different firewall configured for the HTTPS connectivity. The command used by this script is:

```
curl -k -v --data-binary @C:\ITSO_DP\Data\TestEncryptSOAP_IN.xml
https://9.42.170.230:12083/HTTPSec/AccountStatus
```

Note: The DataPower XML Firewall for the HTTP connection is configured to listen on port 12080 and URL /HTTPNoSec/AccountStatus. The DataPower XML Firewall for the HTTPS connection is configured to listen on port 12083 and URL /HTTPSec/AccountStatus.

Step three: Test XML firewalls with prebuilt, encrypted/signed message

This step provides instructions for testing the XML Firewall gateways within DataPower using a prebuilt, encrypted, and signed message provided for your testing:

1. Run the script C:\ITSO_DP\Scripts\Test_EncSignSOAP_DP_HTTP.cmd to test a prebuilt, encrypted, and signed SOAP message. This script sends an encrypted and signed SOAP message TestEncryptSOAP_IN.xml message to the XML Firewall Gateway for the HTTP connection via the IP address `9.42.170.230` and port `12080`. On the inbound transaction, DataPower decrypts and verifies the signed SOAP message before sending it to WebSphere Message Broker. On the outbound transaction, DataPower encrypts and signs the response that it gets from WebSphere Message Broker before sending it to the client. The command used by this script is:

```
curl -k -v --data-binary @C:\ITSO_DP\Data\TestEncSignSOAP_IN.xml
https://9.42.170.230:12080/HTTPNoSec/AccountStatus
```

2. Run the script C:\ITSO_DP\Scripts\Test_EncSignSOAP_DP_HTTPS.cmd to test a prebuilt, encrypted, and signed SOAP message for the HTTPS connection. This script is similar to the previous script, except that it uses a different port number and HTTPS URL. The command used by this script is:

```
curl -k -v --data-binary @C:\ITSO_DP\Data\TestEncSignSOAP_IN.xml
https://9.42.170.230:12083/HTTPSec/AccountStatus
```

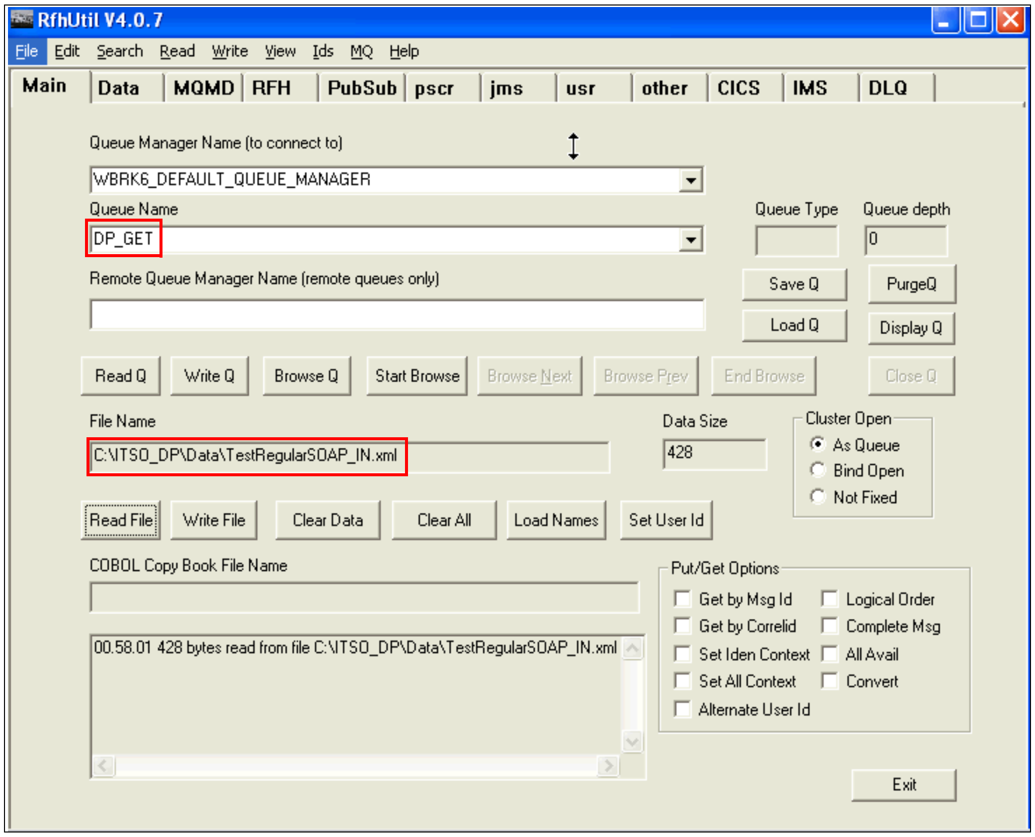
Testing scenario two

We explain how to test the configuration of the DataPower Multi-Protocol Gateway and the integration of DataPower and WebSphere Message Broker using MQ in this section. We use the *RFHUtil* utility to put and get messages from a queue for testing. RFHUtil is a free support pack that you can download from this Web site:

http://www-1.ibm.com/support/docview.wss?uid=swg24000637

According to Figure 2-2 on page 20, the MQ client application (RFHUtil) communicates with DataPower via a pair of queues DP_GET (to put a request message to DataPower) and DP_PUT (to get a response message from DataPower). DataPower is configured to send a request to WebSphere Message Broker via the SOA_REQUEST queue. DataPower receives responses from WebSphere Message Broker in the SOA_REPLY queues. The following steps explain how to test scenario two:

1. Test DataPower Multi-Protocol Gateway with a non-encrypted or encrypted message using the RFHUtil:
   a. Start the RFHUtil utility and enter the following information as shown in Figure 2-106.
      i. Queue Manager Name: `WBRK6_DEFAULT_QUEUE_MANAGER`
      ii. Queue Name: `DP_GET`
      iii. Click **Read File** and select the file name **C:\ITSO_DP\Data\TestRegularSOAP_IN.xml**. First, we use a non-encrypted message, and we will use an encrypted message next. The output is an encrypted message in both tests.


RfhUtil V4.0.7
File Edit Search Read Write View Ids MQ Help
Main Data MQMD RFH PubSub pscr jms usr other CICS IMS DLQ
Save Q PurgeQ Load Q Display Q
Read Q Write Q Browse Q Start Browse Browse Next Browse Prev End Browse Close Q
Read File Write File Clear Data Clear All Load Names Set User Id
Put/Get Options
Get by Msg Id Logical Order Get by Correlid Complete Msg Set Iden Context All Avail Set All Context Convert Alternate User Id
Exit

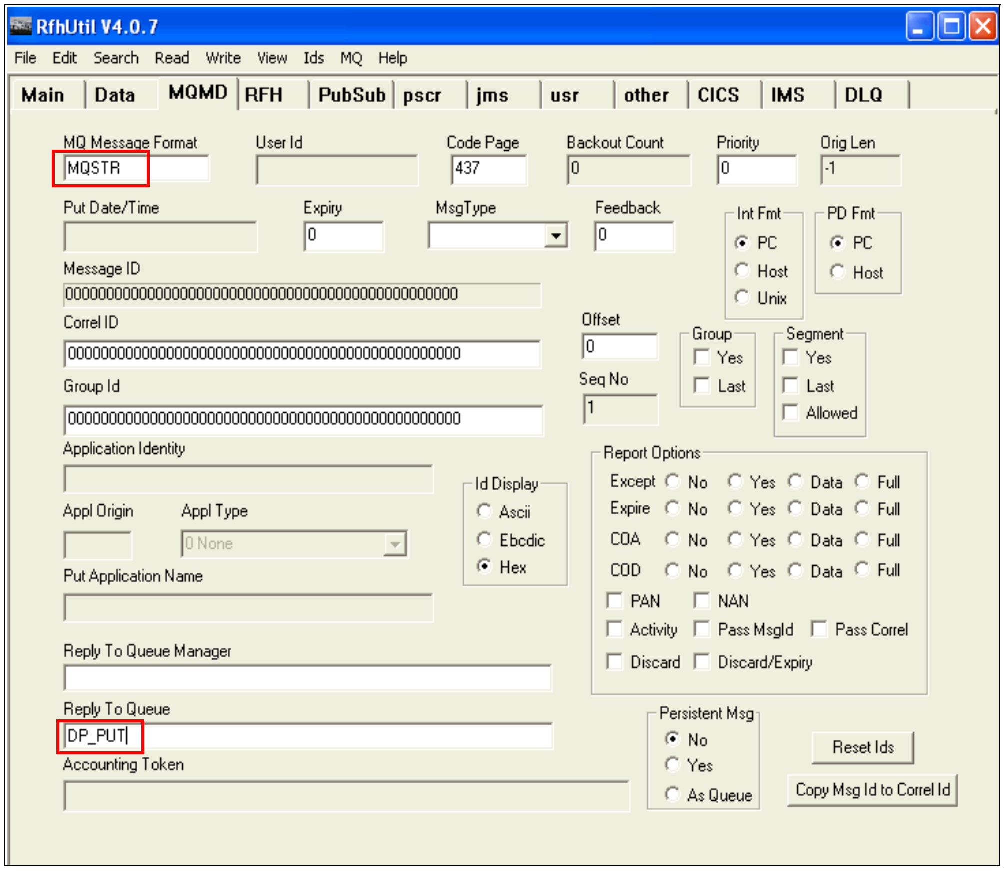
Queue Manager Name (to connect to): WBRK6_DEFAULT_QUEUE_MANAGER
Queue Name: DP_GET
Remote Queue Manager Name (remote queues only):
Queue Type:
Queue depth: 0
File Name: C:\ITSO_DP\Data\TestRegularSOAP_IN.xml
Data Size: 428
Cluster Open: [✓ As Queue] Bind Open Not Fixed
COBOL Copy Book File Name:

```
00.58.01 428 bytes read from file C:\ITSO_DP\Data\TestRegularSOAP_IN.xml
```

Figure 2-106 RFHUtil

      iv. Click the **MQMD** tab and set the MQ Message Format to `MQSTR` and the Replottted to `DP_PUT` as shown in Figure 2-107 on page 94.

      The Rivulet is used to put a request SOAP message to the DP_GET queue.
      v. Back to the **Main** tab, click **Write Q** to put the message in the DP_GET queue.


RfhUtil V4.0.7
File Edit Search Read Write View Ids MQ Help
Main Data MQMD RFH PubSub pscr jms usr other CICS IMS DLQ
Reset Ids
Copy Msg Id to Correl Id

MQ Message Format: MQSTR
User Id:
Code Page: 437
Backout Count: 0
Priority: 0
Orig Len: -1
Put Date/Time:
Expiry: 0
MsgType:
Feedback: 0
Int Fmt: [✓ PC] Host Unix
PD Fmt: [✓ PC] Host
Message ID: 000000000000000000000000000000000000000000000000
Correl ID: 000000000000000000000000000000000000000000000000
Group Id: 000000000000000000000000000000000000000000000000
Offset: 0
Seq No: 1
Group: Yes Last
Segment: Yes Last Allowed
Application Identity:
Id Display: Ascii Ebcdic [✓ Hex]
Appl Origin:
Appl Type: 0 None
Put Application Name:
Report Options:
Except: No Yes Data Full
Expire: No Yes Data Full
COA: No Yes Data Full
COD: No Yes Data Full
PAN NAN
Activity Pass MsgId Pass Correl
Discard Discard/Expiry
Reply To Queue Manager:
Reply To Queue: DP_PUT
Accounting Token:
Persistent Msg: [✓ No] Yes As Queue

Figure 2-107 RFHUtil: Set MQMD

2. Verify that the message is processed:

   a. Use WebSphere MQ Explorer console to check the queue depth of the DP_PUT queue. You will see a reply message in the DP_PUT queue if it works. Otherwise, the message is in the DP_GET (if DataPower did not pick it up), CATCH_ALL_ERRORS, MQGET_ERROR, or Dead Letter queue.

   b. Use RFHUtil to read the response message as shown in Figure 2-108 on page 95 by following these steps:

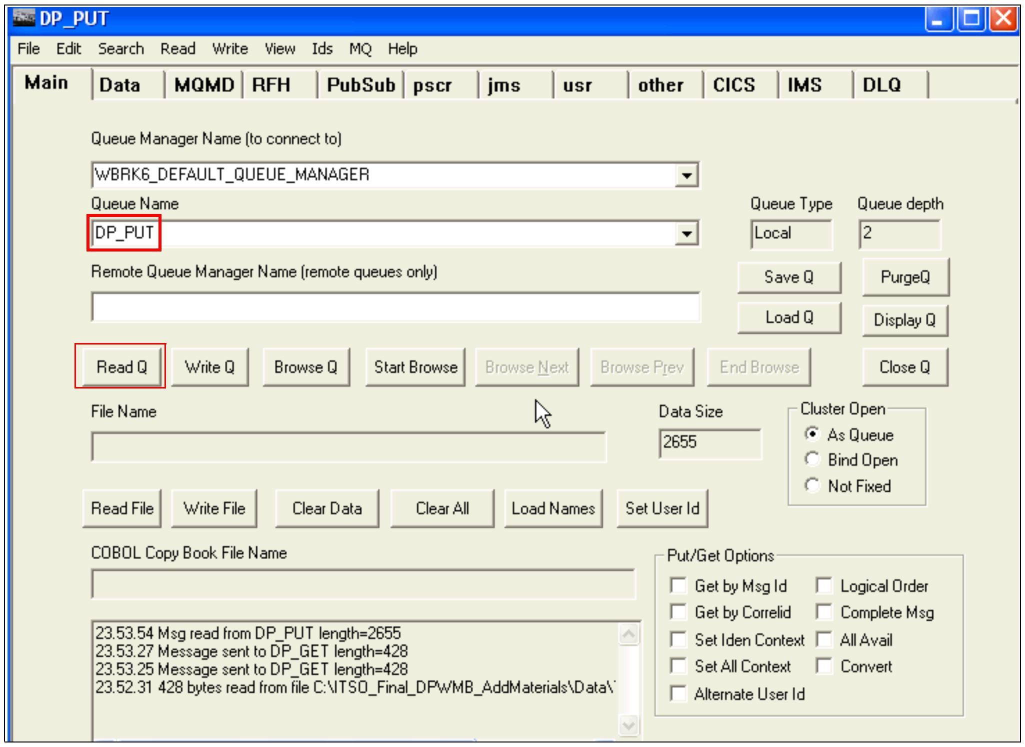
      i. Set the Queue Name to `DP_PUT` (select from the drop-down list).

      ii. Click **Read Q**.

   c. Verify that the message is encrypted:

      i. Click the **Data** tab.

      ii. Click **XML** and verify that the data is encrypted as illustrated in Figure 2-109 on page 95.


DP_PUT
File Edit Search Read Write View Ids MQ Help
Main Data MQMD RFH PubSub pscr jms usr other CICS IMS DLQ
Save Q PurgeQ Load Q Display Q
Read Q Write Q Browse Q Start Browse Browse Next Browse Prev End Browse Close Q
Read File Write File Clear Data Clear All Load Names Set User Id

Queue Manager Name (to connect to): WBRK6_DEFAULT_QUEUE_MANAGER

Queue Name: DP_PUT

Queue Type: Local

Queue depth: 2

Remote Queue Manager Name (remote queues only):

File Name:

Data Size: 2655

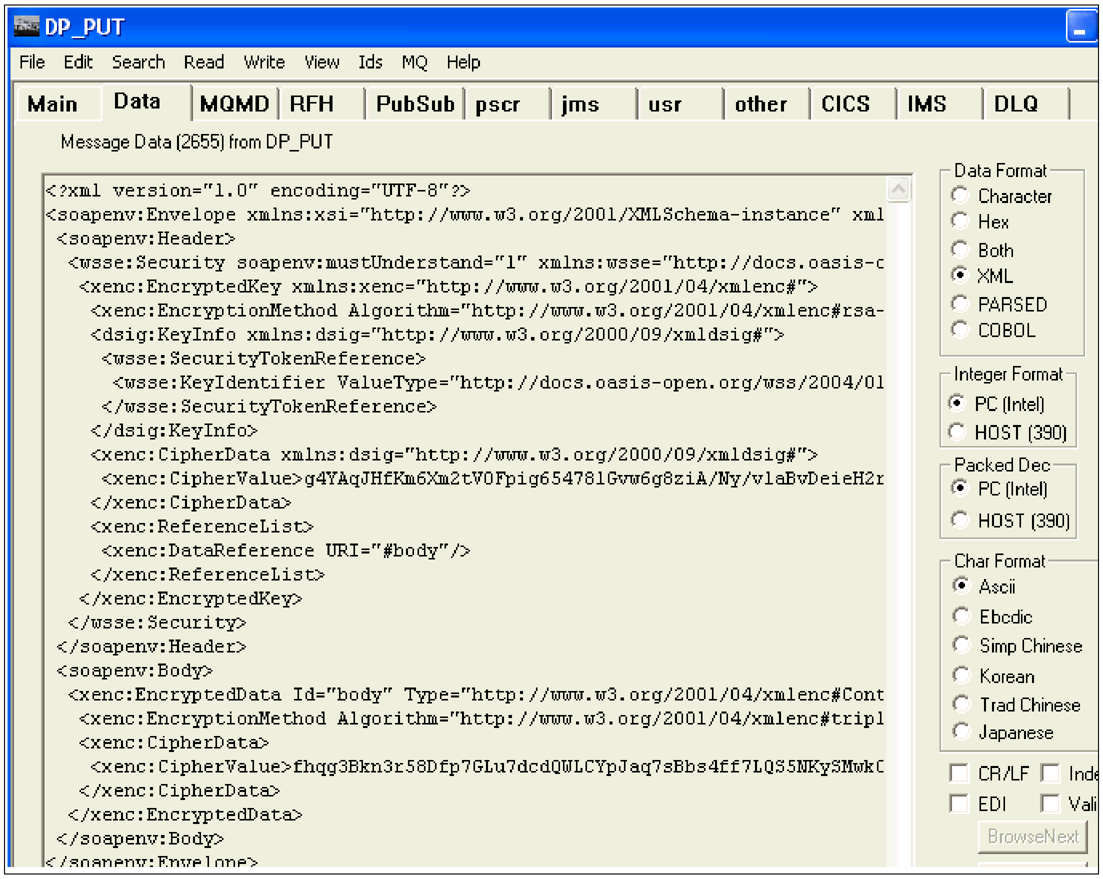
Cluster Open: [✓ As Queue] Bind Open Not Fixed

COBOL Copy Book File Name:

Put/Get Options: Get by Msg Id, Logical Order, Get by Correlid, Complete Msg, Set Iden Context, All Avail, Set All Context, Convert, Alternate User Id

```
23.53.54 Msg read from DP_PUT length=2655
23.53.27 Message sent to DP_GET length=428
23.53.25 Message sent to DP_GET length=428
23.52.31 428 bytes read from file C:\ITSO_Final_DPWMB_AddMaterials\Data\
```

Figure 2-108   RFHUtil: Read message


DP_PUT
File Edit Search Read Write View Ids MQ Help
Main Data MQMD RFH PubSub pscr jms usr other CICS IMS DLQ
BrowseNext

Message Data (2655) from DP_PUT

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xml
 <soapenv:Header>
  <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-c
   <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-
    <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
     <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/2004/01
     </wsse:SecurityTokenReference>
    </dsig:KeyInfo>
    <xenc:CipherData xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
     <xenc:CipherValue>g4YAqJHfKm6Xm2tV0Fpig654781Gvw6g8ziA/Ny/vlaBvDeieH2r
    </xenc:CipherData>
    <xenc:ReferenceList>
     <xenc:DataReference URI="#body"/>
    </xenc:ReferenceList>
   </xenc:EncryptedKey>
  </wsse:Security>
 </soapenv:Header>
 <soapenv:Body>
  <xenc:EncryptedData Id="body" Type="http://www.w3.org/2001/04/xmlenc#Cont
   <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripl
   <xenc:CipherData>
    <xenc:CipherValue>fhqg3Bkn3r58Dfp7GLu7dcdQWLCYpJaq7sBbs4ff7LQS5NKySMwkC
   </xenc:CipherData>
  </xenc:EncryptedData>
 </soapenv:Body>
</soapenv:Envelope>
```

Data Format: Character, Hex, Both, [✓ XML], PARSED, COBOL

Integer Format: [✓ PC (Intel)], HOST (390)

Packed Dec: [✓ PC (Intel)], HOST (390)

Char Format: [✓ Ascii], Ebcdic, Simp Chinese, Korean, Trad Chinese, Japanese

CR/LF, Inde, EDI, Vali

Figure 2-109   Encrypted data

Summary

The IBM DataPower for SOA Appliances is well known for its security features. DataPower has opened the door for clients of WebSphere Message Broker to expose the services provided by the back-end application in a secured way to their customers over the Internet and external business partners. Together, WebSphere Message Broker and DataPower offer an extremely powerful solution to integrate SOA and existing applications where the front-end security is provided by the best SOA appliance and the back-end connectivity is provided by the best integration product.

The primary purpose of this chapter was to explain how to configure the IBM WebSphere DataPower SOA Appliances (DataPower) to serve as a front-end XML and Web service security gateway to offload the security processing from the IBM WebSphere Message Brokers (WMB). It also showed how to configure WebSphere Message Broker to integrate with DataPower via the HTTP and HTTPS protocols.

You have learned how to configure DataPower to provide the front-end XML firewall security gateway for the message broker using the security wizard within the message broker. This security wizard automates the configuration of the DataPower XML security gateways and its processing policies and rules for the HTTP and HTTPS connection with its clients and the message broker. You have also learned all the steps to configure a Multi-Protocol Gateway within DataPower to provide support for MQ connectivity to its clients and to the message broker.

You also learned how to test the scenarios using various testing tools. You learned how to turn on the DataPower Probe, which is the message broker trace for problem determination.

For your convenience, we provided the prebuilt message broker artifacts for ease of testing. Optionally, you also learned how to build the message broker artifacts if you choose to build them yourself. We explained how to build these artifacts in Appendix B, “Building message flows in WebSphere Message Broker” on page 119. This chapter gives you the fundamental knowledge to build and configure end-to-end connectivity to the back-end services that are exposed as Web Services or MQ applications.

3

XML threat protection in DataPower

XML is the new prime target for malicious hackers wanting to harm enterprises. For example:

- It is the common currency of data exchange.
- It passes through firewalls riding on SOAP/HTTP.
- It is the first line of processing and transformation in Enterprise Service Buses (ESBs).
- It is used in plenty of default or non-hardened installations due to lack of experience.

In this chapter, we explore the parameters and properties offered by DataPower for XML threat protection. These are properties common to all forms of service, and several of the properties are turned on by default when the service is created.

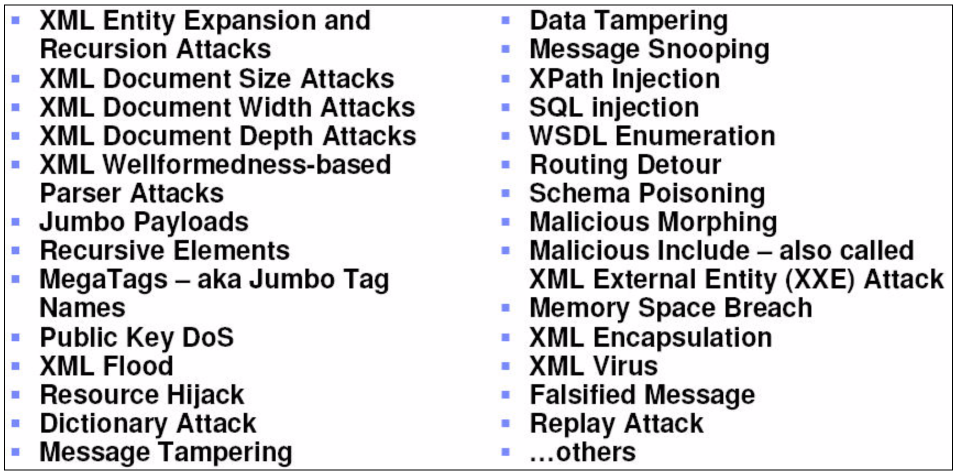
Note: This chapter is based of the developerWorks article at:

http://www.ibm.com/developerworks/websphere/techjournal/0603_col_hines/0603_col_hines.html

See the original article for more up-to-date information.

3.1 XML as the new carrier for attacks

A list of possible XML attacks is shown in Figure 3-1.



- XML Entity Expansion and Recursion Attacks
- XML Document Size Attacks
- XML Document Width Attacks
- XML Document Depth Attacks
- XML Wellformedness-based Parser Attacks
- Jumbo Payloads
- Recursive Elements
- MegaTags – aka Jumbo Tag Names
- Public Key DoS
- XML Flood
- Resource Hijack
- Dictionary Attack
- Message Tampering
- Data Tampering
- Message Snooping
- XPath Injection
- SQL injection
- WSDL Enumeration
- Routing Detour
- Schema Poisoning
- Malicious Morphing
- Malicious Include – also called XML External Entity (XXE) Attack
- Memory Space Breach
- XML Encapsulation
- XML Virus
- Falsified Message
- Replay Attack
- ...others

Figure 3-1 A non-exhaustive list of XML-based attacks

In this chapter, we illustrate how DataPower provides protection against many of these attacks. XML attacks can be divided into four categories:

- XML Denial of Service (xDoS): Slowing down or disabling a Web Service so that valid service requests are hampered or denied
- Unauthorized Access: Gaining unauthorized access to a Web Service or its data
- Data Integrity and Confidentiality: Attacks that strike at the data integrity of Web Service responses, requests, or underlying databases
- System Compromise: Corrupting the Web Service itself or the servers that host it, attacks that gain control of your system.

3.1.1 XML Denial of Service (XDoS)

XML Denial of Service (XDoS) can be further divided into Single and Multiple XDoS.

Single XDoS

Examples of single XDoS attacks include:

- Jumbo Payloads: Sending a very large XML message to exhaust memory and CPU on the target system
- Recursive Elements: XML messages that can be used to force recursive entity expansion or other repeated processing to exhaust server resources
- MegaTags: Otherwise valid XML messages with excessively long element names that can lead to buffer overruns
- Coercive Parsing: XML messages specially constructed to be difficult to parse to consume the resources of the machine
- Public Key DoS: Utilizing the asymmetric nature of public key operations to force resource exhaustion on the recipient by transmitting a message with a large number of long key length, computationally expensive digital signatures

Example 3-1 on page 99 shows a coercive parsing and recursive element.

Example 3-1 Single XDoS coercive parsing and recursive element

```
<?xml version="1.0"?>
<!DOCTYPE billion [
<!ELEMENT billion (#PCDATA)>
<!ENTITY laugh0 "ha! ">
<!ENTITY laugh1 "&laugh0;&laugh0;">
<!ENTITY laugh2 "&laugh1;&laugh1;">
...
<!ENTITY laugh127 "&laugh126;&laugh125;">
]> <billion>&laugh127;</billion>
```

This example is a completely valid, well-formed XML document, which when submitted to the parser, quickly exhausts memory and CPU.

```
<billion>
ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha!
ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha!
ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha!
ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha!
ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha! ha!
....
</billion>
```

Multiple-message XDoS attacks

Examples of multiple-message XDoS attacks include:

- XML Flood: Sending thousands of otherwise benign messages per second to tie up a Web Service. Often combined with the Replay Attack to bypass authentication and Single Message XDoS to increase its impact
- Resource Hijack: Sending messages that lock or reserve resources on the target server as part of a never-completed transaction. For example, messages that intentionally force lock contention on resources or similar situations

3.1.2 Unauthorized access attacks

Examples of unauthorized access attacks are:

- Dictionary Attack: Guessing the password of a valid user using a brute force search through dictionary words
- Falsified Message: Faking that a message is from a valid user, such as by using Man-in-the-Middle to gain a valid message and modifying it to send a different message
- Replay Attack: Resending a previously valid message for malicious effect, possibly where only parts of the message, such as the security token, are replayed

3.1.3 Data integrity and confidentiality attacks

Examples of data integrity and confidentiality attacks are:

- Message Tampering: Modifying parts of a request or response in flight, most dangerous when undetected; less commonly known as *Message Alteration*

- Data Tampering: Exploiting weakness in the access control mechanism that permits the attacker to make unauthorized calls to the Web Service to alter data
- Message Snooping: A direct attack on data privacy by examining all or part of the content of a message. This can happen to messages that are being transmitted in the clear, transmitted encrypted but stored in the clear, or that are decrypted due to a stolen key or crypto-analysis.
- SQL Injection: Modifying SQL in XML to obtain additional data than what the service was designed to return. For example, if XML contains:

  ```
  SELECT USER_NAME FROM USERS WHERE USER_ID = '?1'
  ```

  the `?1` parameter can be replaced with any of the following statements where *user01 is* any valid name:

  - *user01 OR 1=1*
  - *user01;DROP TABLE USERS*
  - *user01*1; INSERT INTO USERS ('user_id', 'passwd', 'is_admin') VALUES ('hines01', 'mypwd', true)
- XPath/XSLT/XQuery Injection: Injection of expressions into the application logic. Newer modifications include Blind XPath Injection, which reduces the knowledge required to mount the attack. No access control built into XPath as there is in SQL, but XQuery can use XACML.
- WSDL Enumeration: Examining the services listed in the Web Service Definition Language (WSDL) to guess and gain access to unlisted services. Request a WSDL by adding *?WSDL* to the end of the URL.

  Note: WSDL is used for defining Web services.

- Routing Detour: Using the SOAP routing header to access to internal Web services

System compromise attacks

Examples of attacks that compromise the system include:

- Malicious Include: Causing a Web service to include invalid external data in output or return privileged files from the server file system. For example, using embedded "file:" URLs to return UNIX® password files or other privileged data to the attacker
- Memory Space Breach: Accomplished via Stack Overflow, Buffer Overrun, or Heap Error. Allows execution of arbitrary code supplied by the attacker with permissions of the host process
- XML Encapsulation: Embedding a system command in the XML payload, such as through the CDATA tag

  Note: CDATA is a way to include non-legal characters in XML, including script.

- XML Virus (X-Virus): Using SOAP with attachments or other attachment mechanisms to transmit malicious executables, such as viruses or worms

3.1.4 Summary

As you can see, SOA implementations, which use XML, are prone to several types of attacks. Traditional systems either offer little by way of warding off these attacks or the offering is difficult to use.

DataPower offers substantial support for protection from malicious XML. Most of these functions are driven by a property or a parameter and easy to enable.

3.2 XML threat protection in DataPower

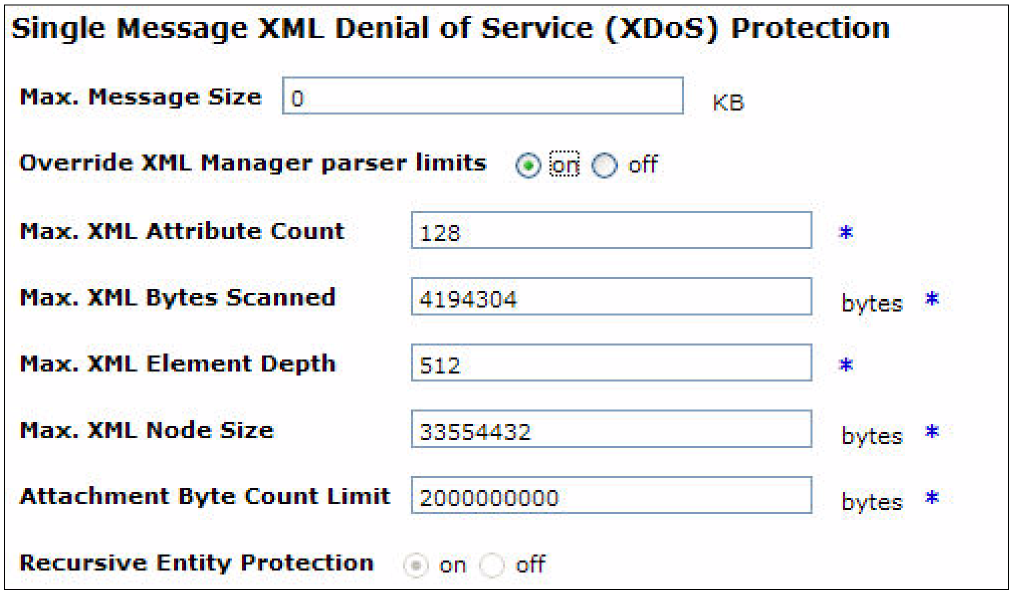
XML threat protection parameters are available for every type of firewall. Six categories of threat protection parameters exist:

- Single Message XML Denial of Service (XDoS) Protection
- Multiple Message XML Denial of Service (XDoS) Protection
- Message Tampering Protection
- SQL Injection Protection
- XML Virus Protection
- Dictionary Attack Protection

In the rest of this section, we describe these categories in detail.

3.2.1 Single message XML Denial of Service protection

Figure 3-2 presents the list of parameters in this category along with their default values.


Single Message XML Denial of Service (XDoS) Protection

Max. Message Size	0	KB	
Override XML Manager parser limits	[✓ on] off		
Max. XML Attribute Count	128		*
Max. XML Bytes Scanned	4194304	bytes	*
Max. XML Element Depth	512		*
Max. XML Node Size	33554432	bytes	*
Attachment Byte Count Limit	2000000000	bytes	*
Recursive Entity Protection	[✓ on] off		

Figure 3-2 Singe Message XDoS parameters

The parameters are:

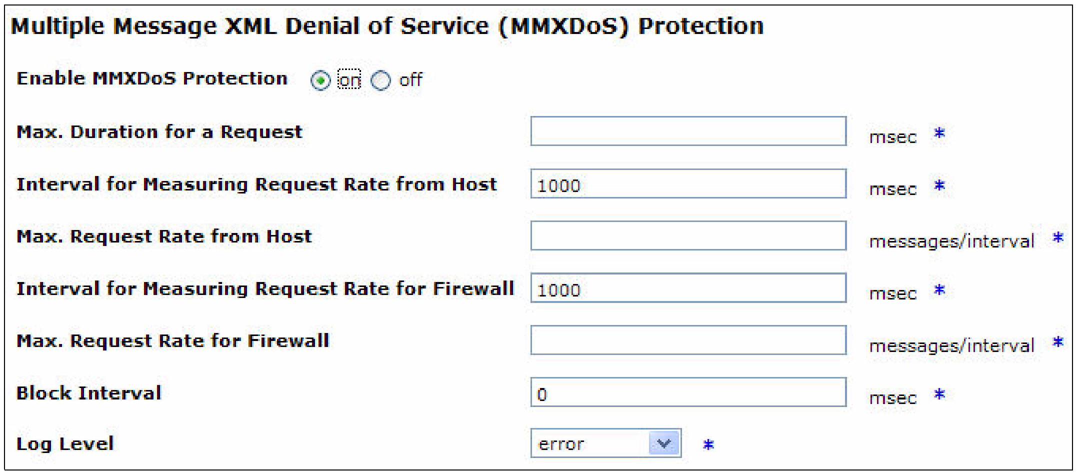
- *Max. Message Size* controls the total size of the XML document in kilobytes. A value of 0 permits documents of any size.
- *Max. XML Attribute Count* controls the total number of attributes allowed on any XML element.
- *Max. XML Bytes Scanned* controls the maximum number of bytes the XML parser will scan for a XML token (for example, a string value, an element name, and so on).
- *Max. XML Element Depth* controls the nesting of XML elements allowed.

- *Max. XML Node Size* controls the total number of bytes allowed in a node (including attributes and nested nodes).
- *Attachment Byte Count Limit* controls the maximum size of any single attachment.

3.2.2 Multiple message XML Denial of Service protection

The parameters for this type of protection are shown in Figure 3-3:

- *Max. Duration for a Request* controls the maximum allowed turnaround time for a request (the responsiveness of the firewall under load conditions).
- *Interval for Measuring Request Rate from Host* controls the time interval for DataPower to measure the *Max. Request Rate from Host* parameter.
- *Max. Request Rate from Host* parameter controls the rate at which requests can be received into the firewall from one IP address.
- *Interval for Measuring Request Rate for Firewall* controls the time interval for DataPower to measure the *Max. Request Rate for Firewall* parameter.
- *Max. Request Rate for Firewall* parameter controls the rate at which requests can be received into the firewall from all IP addresses.
- *Block Interval* controls how long requests will be blocked from the firewall when any of these parameters evaluate to true (indicating a possible DoS attack).



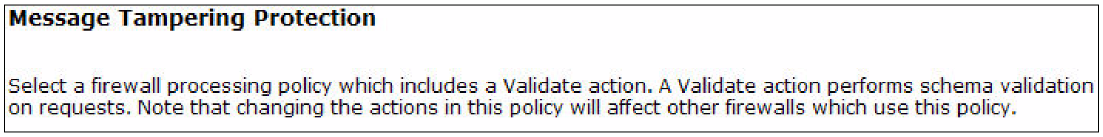
Multiple Message XML Denial of Service (MMXDoS) Protection

Enable MMXDoS Protection	[✓ on] off	
Max. Duration for a Request		msec *
Interval for Measuring Request Rate from Host	1000	msec *
Max. Request Rate from Host		messages/interval *
Interval for Measuring Request Rate for Firewall	1000	msec *
Max. Request Rate for Firewall		messages/interval *
Block Interval	0	msec *
Log Level	error	*

Figure 3-3 Multiple Message XML Denial of Service (MMXDoS) parameters

3.2.3 Message tampering protection

Message tampering protection (Figure 3-4) requires that a *Validate* action is included in the schema to validate the request against a schema. The core application that we used contains an example of a validate action.



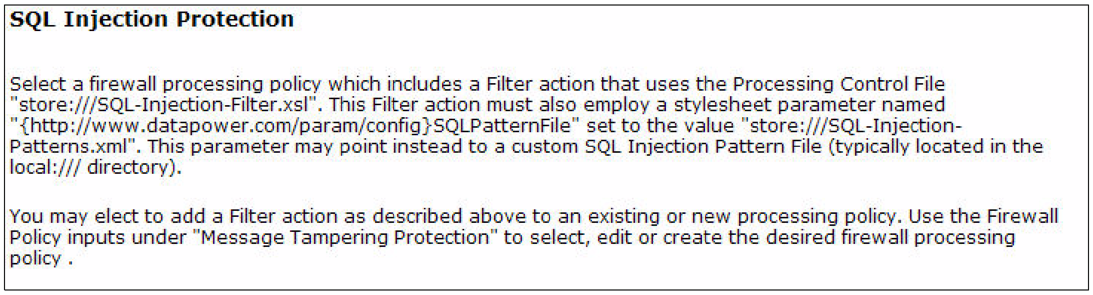
Message Tampering Protection

Select a firewall processing policy which includes a Validate action. A Validate action performs schema validation on requests. Note that changing the actions in this policy will affect other firewalls which use this policy.

Figure 3-4 Message Tampering Protection

3.2.4 SQL injection protection

SQL injection protection (Figure 3-5) is checking SQL injection by using a *Filter* action in the policy. DataPower provides *XSL Transformations (XSLTs)*, which is a language for transforming XML documents into other XML documents, which can provide default checking that can be used in the Filter action. Alternatively, you can use these files as the basis for creating custom protection files suitable to the request.



SQL Injection Protection

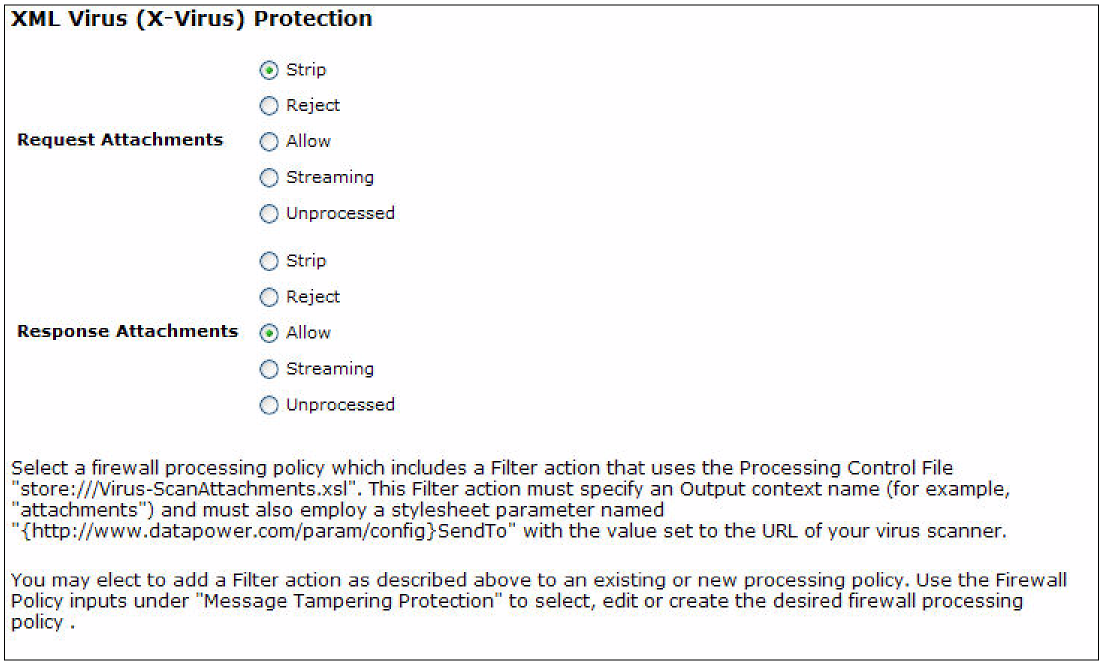
Select a firewall processing policy which includes a Filter action that uses the Processing Control File "store:///SQL-Injection-Filter.xsl". This Filter action must also employ a stylesheet parameter named "{http://www.datapower.com/param/config}SQLPatternFile" set to the value "store:///SQL-Injection-Patterns.xml". This parameter may point instead to a custom SQL Injection Pattern File (typically located in the local:/// directory).

You may elect to add a Filter action as described above to an existing or new processing policy. Use the Firewall Policy inputs under "Message Tampering Protection" to select, edit or create the desired firewall processing policy .

Figure 3-5 SQL injection protection

3.2.5 XML virus protection

Viruses in XML are injected using attachments (Figure 3-6). There are two controls to handle virus threats. At the higher level, you can choose from a range of options from allowing attachments to pass through fully to completely stripping away attachments. If attachments are allowed to be processed by the policy, you can further scrutinize the attachment using a Filter action along with XSLTs supplied by DataPower for that purpose. Of course, you can customize the XSLTs appropriately as well.



XML Virus (X-Virus) Protection

Request Attachments [✓ Strip] Reject Allow Streaming Unprocessed

Response Attachments Strip Reject [✓ Allow] Streaming Unprocessed

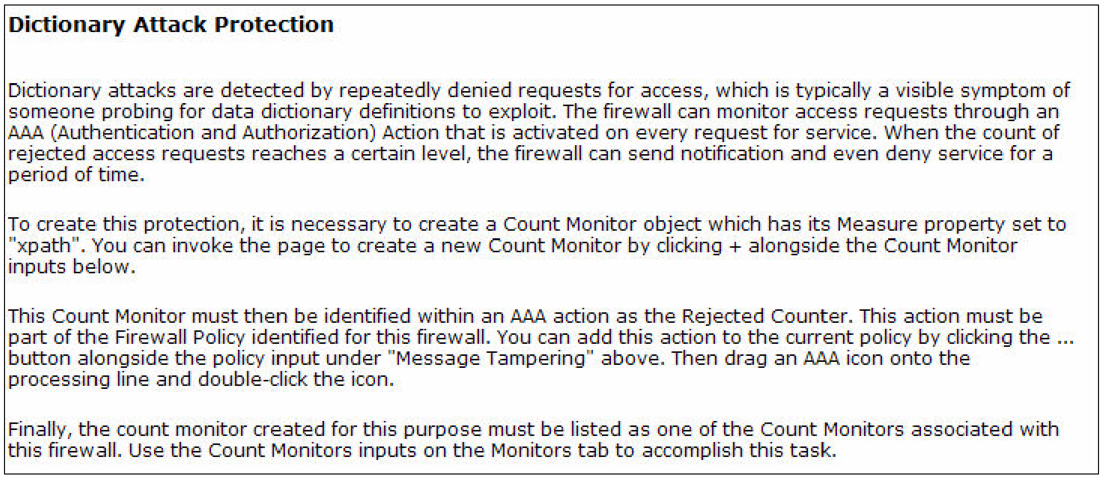
Select a firewall processing policy which includes a Filter action that uses the Processing Control File "store:///Virus-ScanAttachments.xsl". This Filter action must specify an Output context name (for example, "attachments") and must also employ a stylesheet parameter named "{http://www.datapower.com/param/config}SendTo" with the value set to the URL of your virus scanner.

You may elect to add a Filter action as described above to an existing or new processing policy. Use the Firewall Policy inputs under "Message Tampering Protection" to select, edit or create the desired firewall processing policy .

Figure 3-6 XML virus protection

3.2.6 Dictionary attack protection

This is the most complex protection of all. Dictionary attack protection needs the maximum development effort of all of the categories of XML threat protection that we have discussed. The accompanying documentation shown in Figure 3-7 provides the details.



Dictionary Attack Protection

Dictionary attacks are detected by repeatedly denied requests for access, which is typically a visible symptom of someone probing for data dictionary definitions to exploit. The firewall can monitor access requests through an AAA (Authentication and Authorization) Action that is activated on every request for service. When the count of rejected access requests reaches a certain level, the firewall can send notification and even deny service for a period of time.

To create this protection, it is necessary to create a Count Monitor object which has its Measure property set to "xpath". You can invoke the page to create a new Count Monitor by clicking + alongside the Count Monitor inputs below.

This Count Monitor must then be identified within an AAA action as the Rejected Counter. This action must be part of the Firewall Policy identified for this firewall. You can add this action to the current policy by clicking the ... button alongside the policy input under "Message Tampering" above. Then drag an AAA icon onto the processing line and double-click the icon.

Finally, the count monitor created for this purpose must be listed as one of the Count Monitors associated with this firewall. Use the Count Monitors inputs on the Monitors tab to accomplish this task.

Figure 3-7 Dictionary attack protection

3.3 Summary

With the increasing adoption of service-oriented architecture (SOA) comes an increase in the XML traffic, giving rise to newer and more serious kinds of security issues, which can bring down the IT infrastructure of the entire company in an extremely short time. Without adequate means to protect against these attacks, the only viable option is to close the firewall ports that allow XML traffic. This is not an option, because those ports need to be open in order to implement SOA successfully. To truly harden a system using Web Services, you must perform several important security steps. We saw several of these actions in earlier chapters:

- Inspecting messages for well-formedness
- Validating schema
- Verifying digital signatures
- Signing messages
- Implementing service virtualization to mask internal resources via XML transformation and routing
- Encrypting data at the field level

Even before these steps can be performed in a policy, you need a first line of protection right at the edge of the XML firewall (of all types). DataPower provides these types of monitoring functions for the types of attacks right at the front end, as well as configurations to ensure compliance with service level agreements to protect data, thus preventing expensive back-end processing.

A

Web Services security issues

In this appendix, we discuss the usage of the command line tool called *Curl* for transferring files with URL syntax supporting FTP, FTPS, HTTP, and HTTPS protocols. We also describe in detail the usage of *Probe* in the scenario that we used in Appendix 1, “Web Services security improvements” on page 1 and trace the messages before and after each action in the processing rules (Request and Response) using probe.

Curl Tool

You can use the Curl tool as a Web services client. You can download this tool from the Web site:

http://curl.haxx.se/download.html

You can use the following commands for sending the encrypted messages to the `ITSO_MPGW` Multi-Protocol Gateway:

- `curl --data-binary @C:\IBM\Security_Gateway_Artefacts\itso_encry_msg.xml http://datapower.itso.ral.ibm.com:4000` for sending the encrypted messages
- `curl -k -E itsodp-sscert.pem:itsopass --key itsodp-privkey.pem --cacert itsodp-sscert.pem --data-binary @C:\IBM\Security_Gateway_Artefacts\itso_encry_msg.xml https://datapower.itso.ral.ibm.com:4003`

Enable Probe for an encrypted request message

The *MultiStep Probe* displays the contents of contexts and the value of variables at each step of a document processing rule, which can contain one or many steps. This is an invaluable tool for developing service applications and a great troubleshooting tool.

You can enable the MultiStep Probe for any service supported by the Probe by clicking the Probe button on the configuration page of the service. Next, we describe the steps to enable the probe for the `ITSO_MPGW` Multi-Protocol Gateway. Select troubleshooting as shown in Figure A-1.

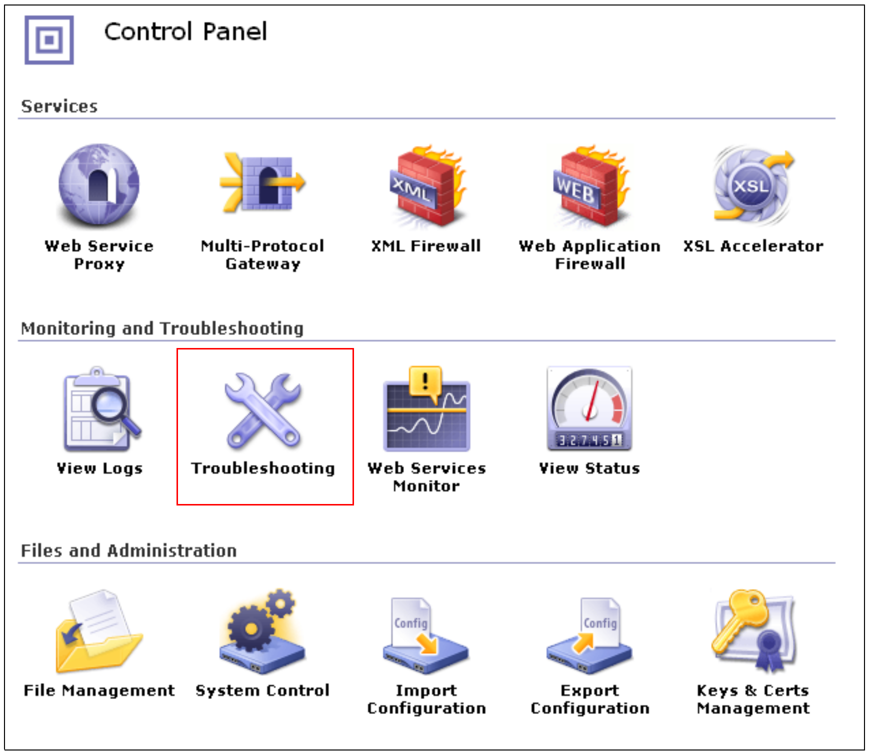


Figure A-1 Control Panel

In Figure A-2, click the **Probe** tab.

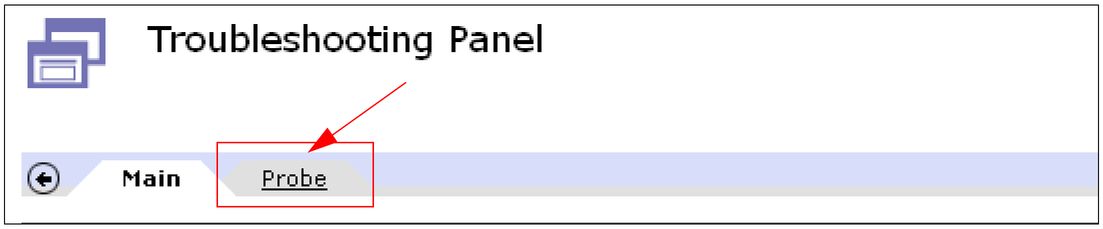


Figure A-2 Probe tab in Troubleshooting Panel

The steps are:

1. Click **Add Probe** in Figure A-3. The probe will be enabled only for the Multi-Protocol Gateway named `ITSO_MPGW`. If you have a number of multi-protocol gateways, you can choose to enable Probe on a particular Multi-Protocol Gateway or you can enable the Probe on multiple gateways by repeating the process.

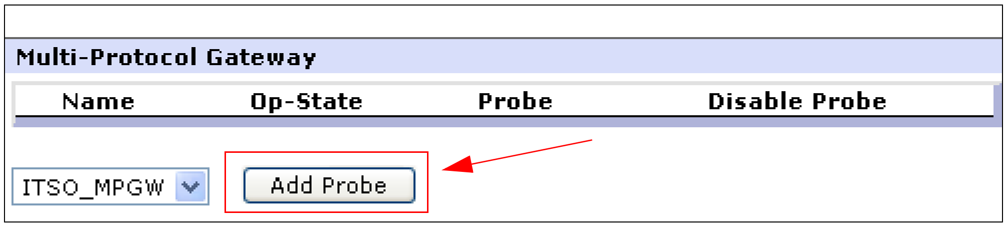
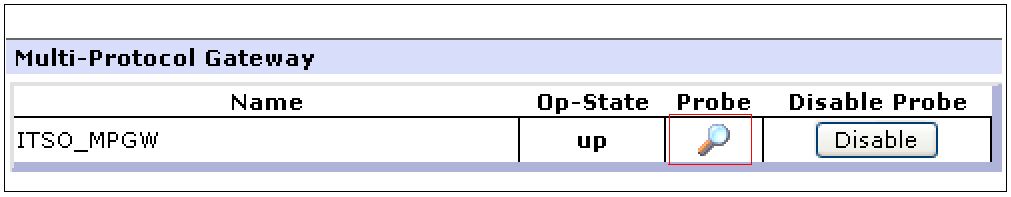


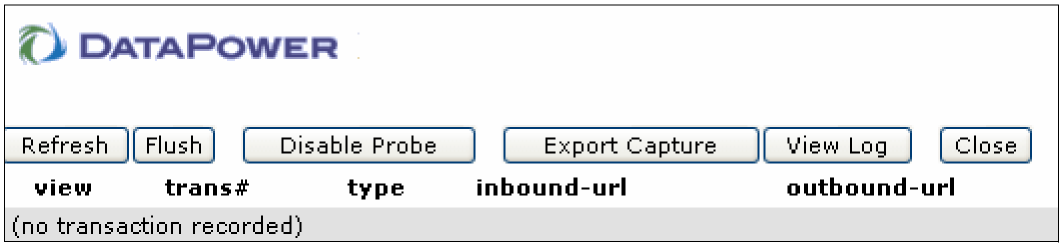
Figure A-3 Add Probe in a Multi-Protocol Gateway

2. Click the magnifying glass icon under Probe in Figure A-4 to open the transaction record panel, which is shown in Figure A-5.



Name	Op-State	Probe	Disable Probe
ITSO_MPGW	up	🔍	Disable

Figure A-4 Probe enabled for the ITSO_MPGW gateway



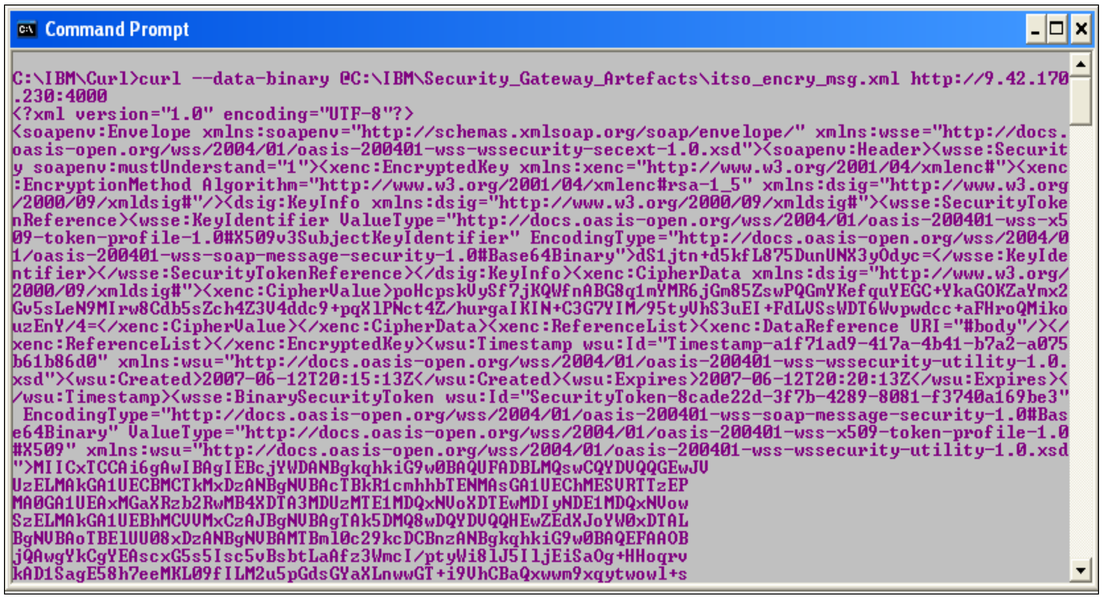
view	trans#	type	inbound-url	outbound-url
(no transaction recorded)				

Figure A-5 Transaction record

3. The transactions get recorded when the message is sent to the Multi-Protocol Gateway. We use the **`curl`** tool to send a SOAP message using the following command:

```
curl --data-binary @C:\IBM\Security_Gateway_Artefacts\itso_encry_msg.xml
http://9.42.170.230:4000
```

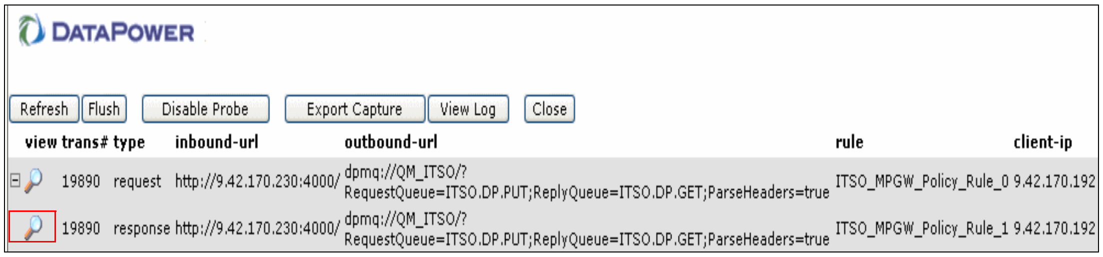
4. The command line panel is shown in Figure A-6 on page 108.



```
C:\IBM\Curl>curl --data-binary @C:\IBM\Security_Gateway_Artefacts\itso_encry_msg.xml http://9.42.170
.230:4000
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsse="http://docs.
oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"><soapenv:Header><wsse:Securit
y soapenv:mustUnderstand="1"><xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc
:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" xmlns:dsig="http://www.w3.org
/2000/09/xmldsig#"/><dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"><wsse:SecurityToke
nReference><wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x5
09-token-profile-1.0#X509v3SubjectKeyIdentifier" EncodingType="http://docs.oasis-open.org/wss/2004/0
1/oasis-200401-wss-soap-message-security-1.0#Base64Binary">dS1jtn+d5kfL875DunUNX3yOdyc=</wsse:KeyIde
ntifier></wsse:SecurityTokenReference></dsig:KeyInfo><xenc:CipherData xmlns:dsig="http://www.w3.org/
2000/09/xmldsig#"><xenc:CipherValue>poHcpskVySf7jKQWfnABG8q1mYMR6jGm85ZswPQGmYKefquYEGC+YkaGOKZaYmx2
Gv5sLeN9MIrw8Cdb5sZch4Z3V4ddc9+pqXlPNct4Z/hurgaIKIN+C3G7YIM/95tyVhS3uEI+FdLVSsWDT6Wvpwdcc+aFHroQMiko
uzEnY/4=</xenc:CipherValue></xenc:CipherData><xenc:ReferenceList><xenc:DataReference URI="#body"/></
xenc:ReferenceList></xenc:EncryptedKey><wsu:Timestamp wsu:Id="Timestamp-a1f71ad9-417a-4b41-b7a2-a075
b61b86d0" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.
xsd"><wsu:Created>2007-06-12T20:15:13Z</wsu:Created><wsu:Expires>2007-06-12T20:20:13Z</wsu:Expires><
/wsu:Timestamp><wsse:BinarySecurityToken wsu:Id="SecurityToken-8cade22d-3f7b-4289-8081-f3740a169be3"
 EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Bas
e64Binary" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0
#X509" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
">MIICxTCCAi6gAwIBAgIEBcjYWDANBgkqhkiG9w0BAQUFADBLMQswCQYDVQQGEwJV
UzELMAkGA1UECBMCTkMxDzANBgNVBAcTBkR1cmhhbTENMAsGA1UEChMESVRTTzEP
MA0GA1UEAxMGaXRzb2RwMB4XDTA3MDUzMTE1MDQxNVoXDTEwMDIyNDE1MDQxNVow
SzELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAk5DMQ8wDQYDVQQHEwZEdXJoYW0xDTAL
BgNVBAoTBElUU08xDzANBgNVBAMTBml0c29kcDCBnzANBgkqhkiG9w0BAQEFAAOB
jQAwgYkCgYEAscxG5s5Isc5vBsbtLaAfz3WmcI/ptyWi8lJ5IljEiSaOg+HHoqrv
kAD1SagE58h7eeMKL09fILM2u5pGdsGYaXLnwwGT+i9VhCBaQxwwm9xqytwowl+s
```

Figure A-6 Command line invocation of Curl tool

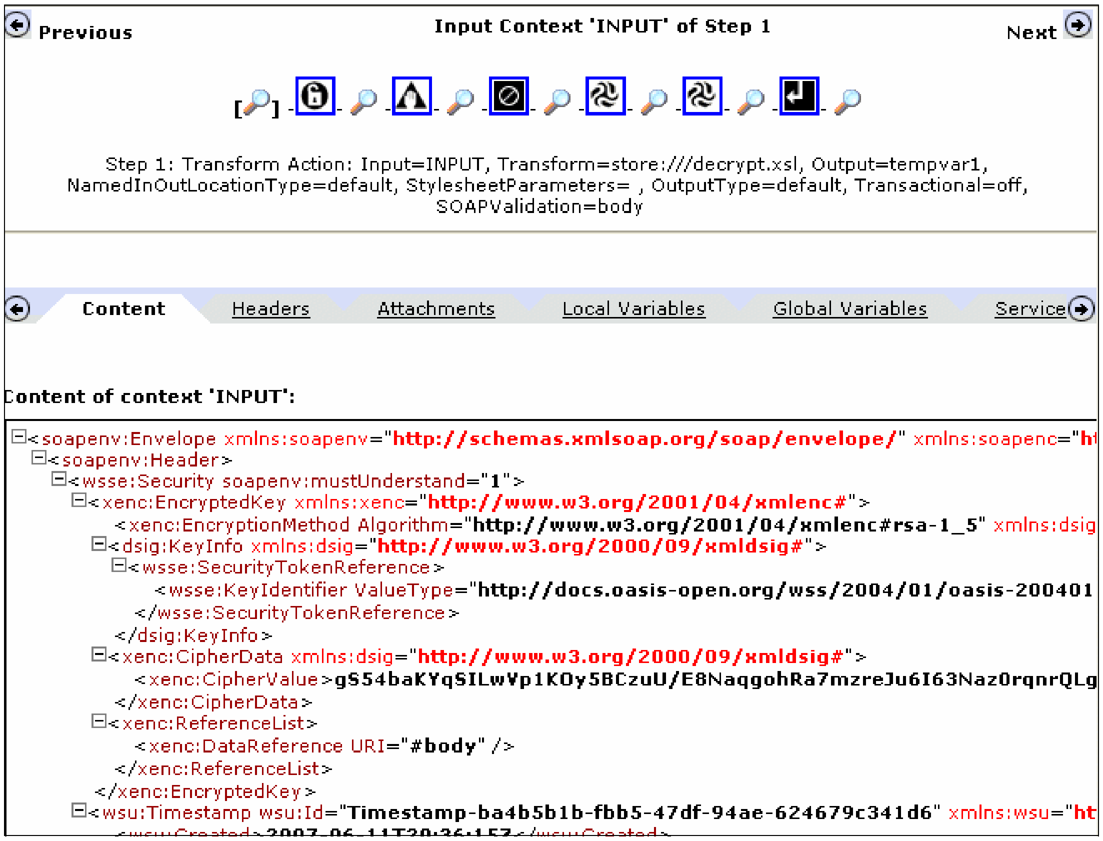
5. After the request message is sent, you receive the response message in SOAP response format in the same command line panel. Because we have now processed a request and response message successfully, the multi-step probe has detected and captured inputs and outputs of all the processing rules (request rule and response rule) on `ITSO_MPGW`, as shown in Figure A-7.



view	trans#	type	inbound-url	outbound-url	rule	client-ip
⊟ 🔍	19890	request	http://9.42.170.230:4000/	dpmq://QM_ITSO/?RequestQueue=ITSO.DP.PUT;ReplyQueue=ITSO.DP.GET;ParseHeaders=true	ITSO_MPGW_Policy_Rule_0	9.42.170.192
🔍	19890	response	http://9.42.170.230:4000/	dpmq://QM_ITSO/?RequestQueue=ITSO.DP.PUT;ReplyQueue=ITSO.DP.GET;ParseHeaders=true	ITSO_MPGW_Policy_Rule_1	9.42.170.192

Figure A-7 Multi-probe transaction detail

6. Clicking the magnifying glass icon shows us input and output messages at each step. The following six panels, beginning with Figure A-8 on page 109, show us the details. Figure A-8 on page 109 shows us Step 1, which shows the incoming encrypted message and subsequent mini-actions that are performed in that decrypt action.


Previous   Input Context 'INPUT' of Step 1   Next

Step 1: Transform Action: Input=INPUT, Transform=store:///decrypt.xsl, Output=tempvar1, NamedInOutLocationType=default, StylesheetParameters= , OutputType=default, Transactional=off, SOAPValidation=body

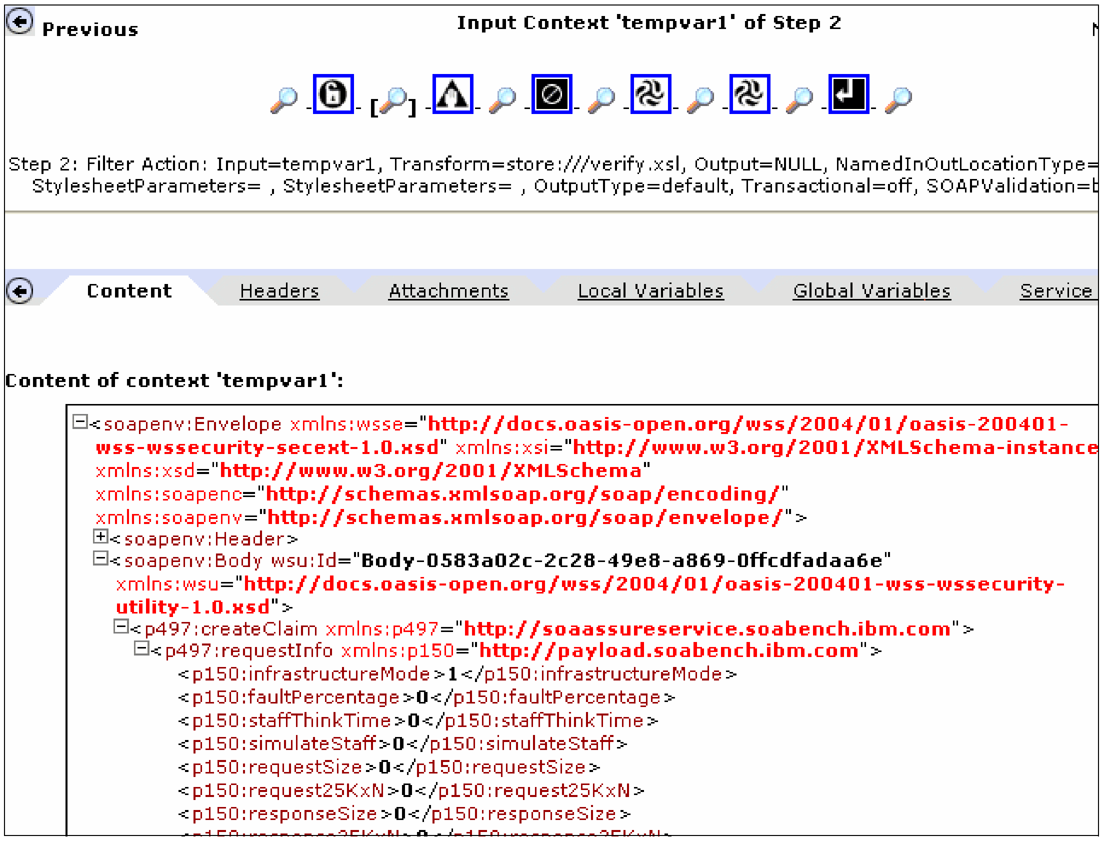
Content   Headers   Attachments   Local Variables   Global Variables   Service

Content of context 'INPUT':

```
⊟<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:soapenc="ht
  ⊟<soapenv:Header>
    ⊟<wsse:Security soapenv:mustUnderstand="1">
      ⊟<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
          <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" xmlns:dsig
        ⊟<dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
          ⊟<wsse:SecurityTokenReference>
              <wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401
            </wsse:SecurityTokenReference>
          </dsig:KeyInfo>
        ⊟<xenc:CipherData xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
            <xenc:CipherValue>gS54baKYqSILwVp1KOy5BCzuU/E8NaqgohRa7mzreJu6I63Naz0rqnrQLg
          </xenc:CipherData>
        ⊟<xenc:ReferenceList>
            <xenc:DataReference URI="#body" />
          </xenc:ReferenceList>
        </xenc:EncryptedKey>
      ⊟<wsu:Timestamp wsu:Id="Timestamp-ba4b5b1b-fbb5-47df-94ae-624679c341d6" xmlns:wsu="ht
```

Figure A-8 Request Processing Rule (Step one of six)

7. In the Figure A-9 on page 110, you can see that the message is successfully decrypted and the payload is in clear text.



Previous

Input Context 'tempvar1' of Step 2

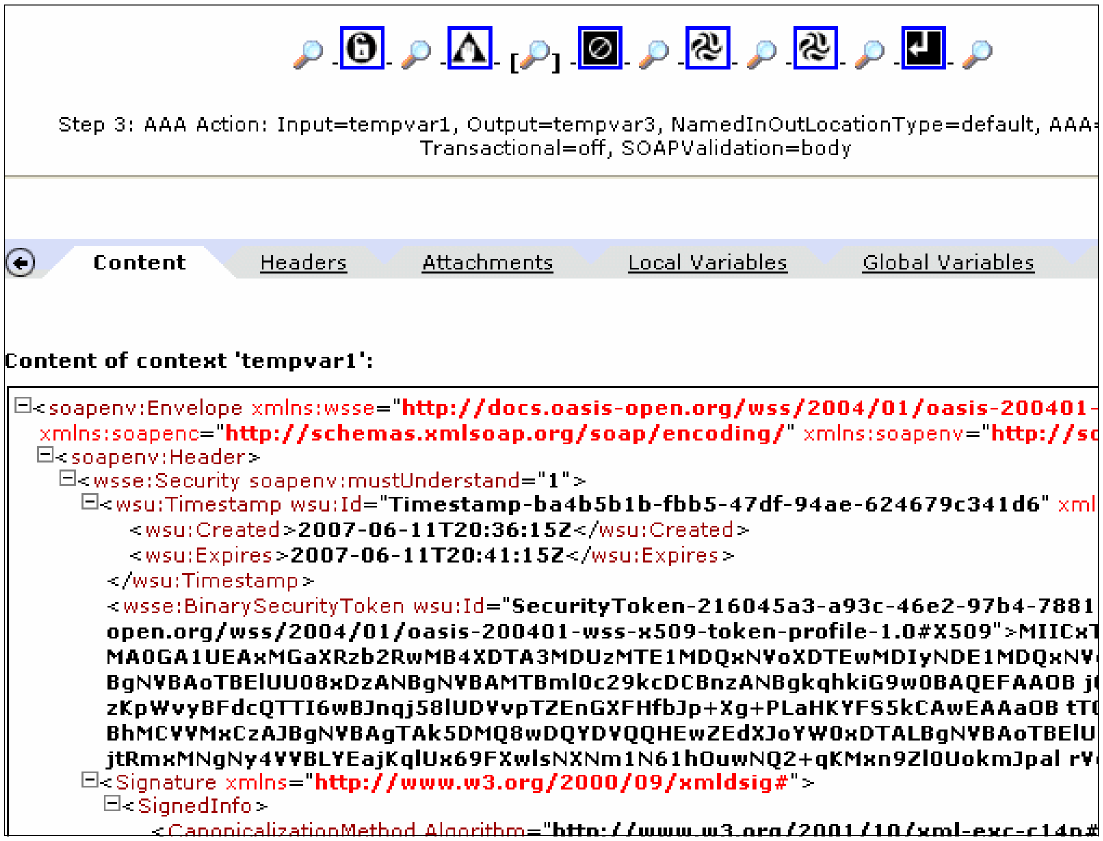
Step 2: Filter Action: Input=tempvar1, Transform=store:///verify.xsl, Output=NULL, NamedInOutLocationType=
StylesheetParameters= , StylesheetParameters= , OutputType=default, Transactional=off, SOAPValidation=

Content of context 'tempvar1':

```
⊟<soapenv:Envelope xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
  wss-wssecurity-secext-1.0.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  ⊞<soapenv:Header>
  ⊟<soapenv:Body wsu:Id="Body-0583a02c-2c28-49e8-a869-0ffcdfadaa6e"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
    utility-1.0.xsd">
    ⊟<p497:createClaim xmlns:p497="http://soaassureservice.soabench.ibm.com">
      ⊟<p497:requestInfo xmlns:p150="http://payload.soabench.ibm.com">
          <p150:infrastructureMode>1</p150:infrastructureMode>
          <p150:faultPercentage>0</p150:faultPercentage>
          <p150:staffThinkTime>0</p150:staffThinkTime>
          <p150:simulateStaff>0</p150:simulateStaff>
          <p150:requestSize>0</p150:requestSize>
          <p150:request25KxN>0</p150:request25KxN>
          <p150:responseSize>0</p150:responseSize>
```

Figure A-9 Decrypted message (Step two of six)

8. Figure A-10 on page 111 shows the decrypted and successfully verified message.



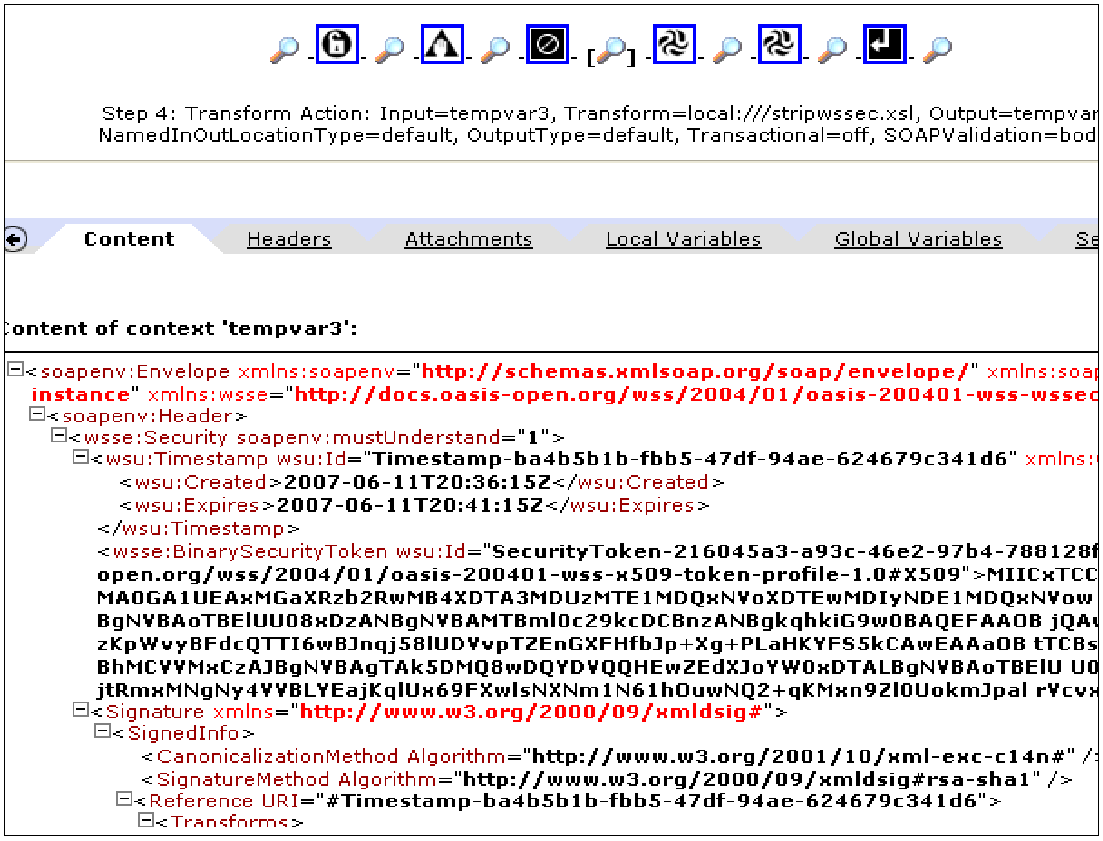
Step 3: AAA Action: Input=tempvar1, Output=tempvar3, NamedInOutLocationType=default, AAA=
Transactional=off, SOAPValidation=body

Content of context 'tempvar1':

```
⊟<soapenv:Envelope xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:soapenv="http://sc
  ⊟<soapenv:Header>
    ⊟<wsse:Security soapenv:mustUnderstand="1">
      ⊟<wsu:Timestamp wsu:Id="Timestamp-ba4b5b1b-fbb5-47df-94ae-624679c341d6" xml
          <wsu:Created>2007-06-11T20:36:15Z</wsu:Created>
          <wsu:Expires>2007-06-11T20:41:15Z</wsu:Expires>
        </wsu:Timestamp>
        <wsse:BinarySecurityToken wsu:Id="SecurityToken-216045a3-a93c-46e2-97b4-7881
        open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509">MIICxT
        MA0GA1UEAxMGaXRzb2RwMB4XDTA3MDUzMTE1MDQxNVoXDTEwMDIyNDE1MDQxNV
        BgNVBAoTBElUU08xDzANBgNVBAMTBml0c29kcDCBnzANBgkqhkiG9w0BAQEFAAOB j
        zKpWvyBFdcQTTI6wBJnqj58lUDVvpTZEnGXFHfbJp+Xg+PLaHKYFS5kCAwEAAaOB tT
        BhMCVVMxCzAJBgNVBAgTAk5DMQ8wDQYDVQQHEwZEdXJoYW0xDTALBgNVBAoTBElU
        jtRmxMNgNy4VVBLYEajKqlUx69FXwlsNXNm1N61hOuwNQ2+qKMxn9Zl0UokmJpal rV
      ⊟<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        ⊟<SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#
```

Figure A-10   Message successfully verified (Step three of six)

9. Figure A-11 on page 112 shows the message successfully authenticated.

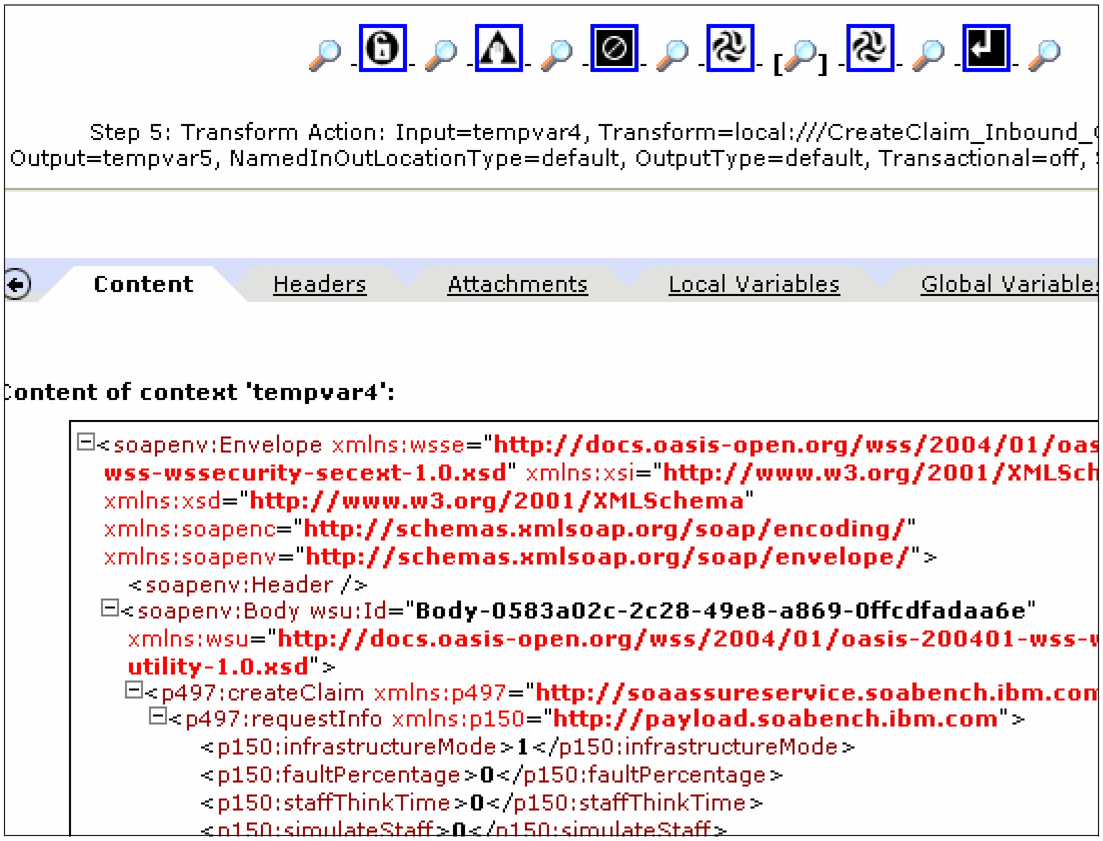

Step 4: Transform Action: Input=tempvar3, Transform=local:///stripwssec.xsl, Output=tempvar
NamedInOutLocationType=default, OutputType=default, Transactional=off, SOAPValidation=bod

ontent of context 'tempvar3':

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:soa
 instance" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsse
  <soapenv:Header>
   <wsse:Security soapenv:mustUnderstand="1">
    <wsu:Timestamp wsu:Id="Timestamp-ba4b5b1b-fbb5-47df-94ae-624679c341d6" xmlns:
       <wsu:Created>2007-06-11T20:36:15Z</wsu:Created>
       <wsu:Expires>2007-06-11T20:41:15Z</wsu:Expires>
     </wsu:Timestamp>
     <wsse:BinarySecurityToken wsu:Id="SecurityToken-216045a3-a93c-46e2-97b4-788128f
     open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509">MIICxTCC
     MA0GA1UEAxMGaXRzb2RwMB4XDTA3MDUzMTE1MDQxNVoXDTEwMDIyNDE1MDQxNVow
     BgNVBAoTBElUU08xDzANBgNVBAMTBml0c29kcDCBnzANBgkqhkiG9w0BAQEFAAOB jQA
     zKpWvyBFdcQTTI6wBJnqj58lUDVvpTZEnGXFHfbJp+Xg+PLaHKYFS5kCAwEAAaOB tTCBs
     BhMCVVMxCzAJBgNVBAgTAk5DMQ8wDQYDVQQHEwZEdXJoYW0xDTALBgNVBAoTBElU U0
     jtRmxMNgNy4VVBLYEajKqlUx69FXwlsNXNm1N61hOuwNQ2+qKMxn9Zl0UokmJpal rVcvx
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
     <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" /
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
      <Reference URI="#Timestamp-ba4b5b1b-fbb5-47df-94ae-624679c341d6">
       <Transforms>
```

Figure A-11   Message successfully authenticated (Step four of six)

10. In Figure A-12 on page 113, the ws-sec header is successfully stripped, and now the entire message is in clear text.



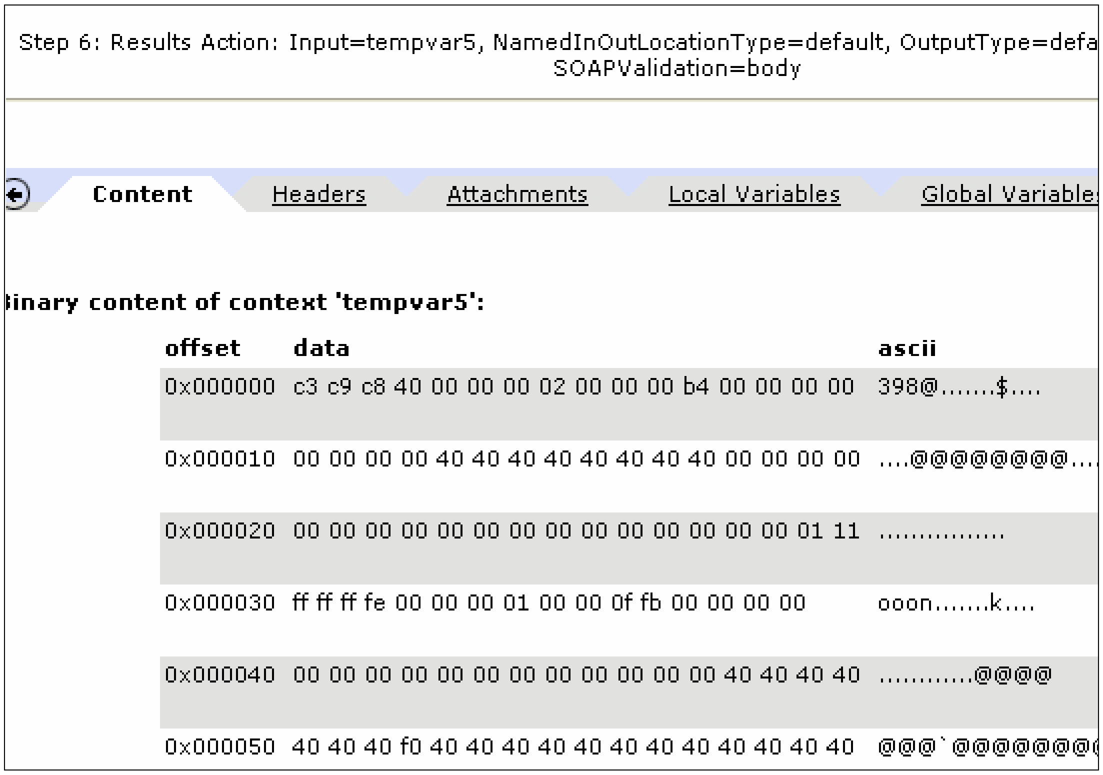
Step 5: Transform Action: Input=tempvar4, Transform=local:///CreateClaim_Inbound_
Output=tempvar5, NamedInOutLocationType=default, OutputType=default, Transactional=off,

ontent of context 'tempvar4':

```
⊟<soapenv:Envelope xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oas
 wss-wssecurity-secext-1.0.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSch
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header />
 ⊟<soapenv:Body wsu:Id="Body-0583a02c-2c28-49e8-a869-0ffcdfadaa6e"
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
   utility-1.0.xsd">
   ⊟<p497:createClaim xmlns:p497="http://soaassureservice.soabench.ibm.con
     ⊟<p497:requestInfo xmlns:p150="http://payload.soabench.ibm.com">
         <p150:infrastructureMode>1</p150:infrastructureMode>
         <p150:faultPercentage>0</p150:faultPercentage>
         <p150:staffThinkTime>0</p150:staffThinkTime>
         <p150:simulateStaff>0</p150:simulateStaff>
```

Figure A-12 Ws-sec headers removed, resulting in all clear text (Step five of six)

11. In Figure A-13 on page 114, you see that the message is converted into binary format for the back-end application.



```
Step 6: Results Action: Input=tempvar5, NamedInOutLocationType=default, OutputType=defa
                                   SOAPValidation=body
```

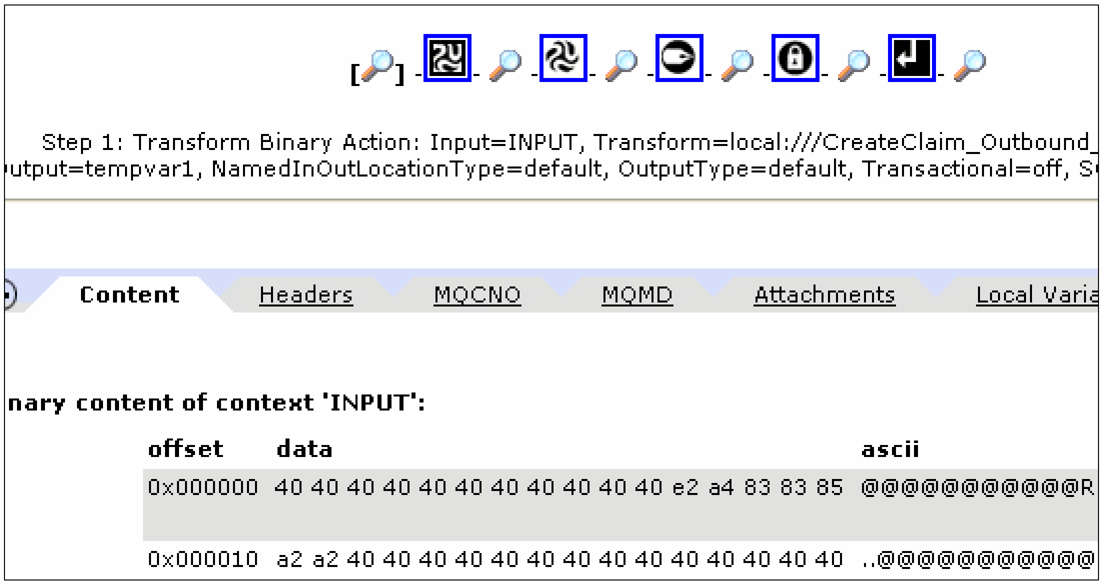
```
inary content of context 'tempvar5':
         offset    data                                              ascii
         0x000000  c3 c9 c8 40 00 00 00 02 00 00 00 b4 00 00 00 00   398@.......$....
         0x000010  00 00 00 00 40 40 40 40 40 40 40 40 00 00 00 00   ....@@@@@@@@...
         0x000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 11   ...............
         0x000030  ff ff ff fe 00 00 00 01 00 00 0f fb 00 00 00 00   ooon.......k....
         0x000040  00 00 00 00 00 00 00 00 00 00 00 00 40 40 40 40   ............@@@@
         0x000050  40 40 40 f0 40 40 40 40 40 40 40 40 40 40 40 40   @@@`@@@@@@@@
```

Figure A-13   Message is transformed and sent to the back-end application (Step six of six)

Response rule

After we receive the message from the back-end application, we need to sign and encrypt the message. We can see the transformation at each step, because we have enabled the Probe:

1. In Figure A-14, we see the incoming message is in binary format, which was received from the back-end application via WebSphere MQ.

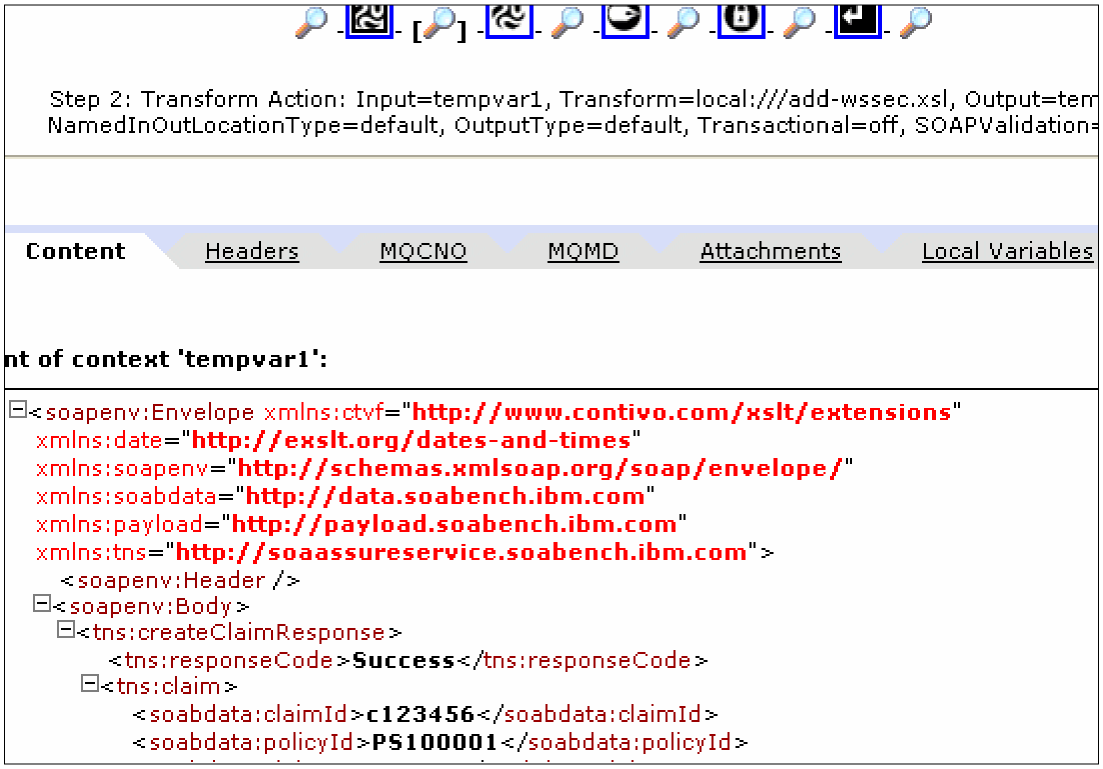


```
 Step 1: Transform Binary Action: Input=INPUT, Transform=local:///CreateClaim_Outbound_
utput=tempvar1, NamedInOutLocationType=default, OutputType=default, Transactional=off, S
```

```
nary content of context 'INPUT':
         offset    data                                              ascii
         0x000000  40 40 40 40 40 40 40 40 40 40 40 e2 a4 83 83 85   @@@@@@@@@@@R
         0x000010  a2 a2 40 40 40 40 40 40 40 40 40 40 40 40 40 40   ..@@@@@@@@@@@@
```

Figure A-14   Binary Message received from the back-end application (Step one of six)

2. Figure A-15 shows that the message has been successfully converted into clear text. Also, notice that there is no ws-sec header present in the payload.



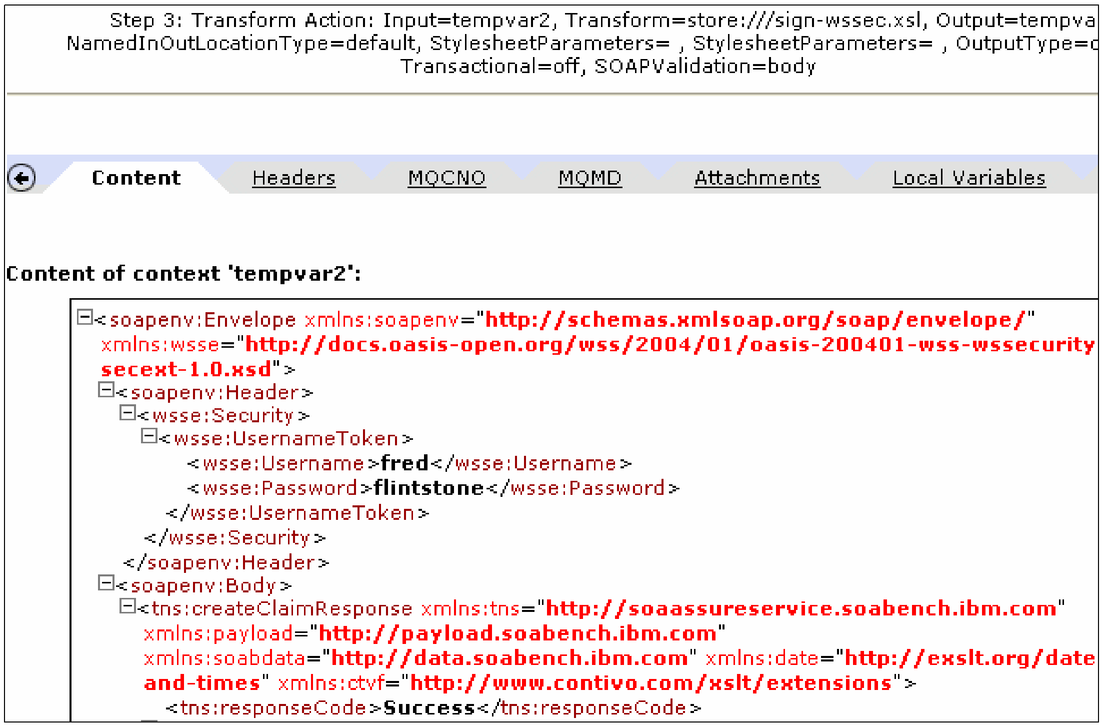
Step 2: Transform Action: Input=tempvar1, Transform=local:///add-wssec.xsl, Output=tem
NamedInOutLocationType=default, OutputType=default, Transactional=off, SOAPValidation=

nt of context 'tempvar1':

```
⊟<soapenv:Envelope xmlns:ctvf="http://www.contivo.com/xslt/extensions"
  xmlns:date="http://exslt.org/dates-and-times"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soabdata="http://data.soabench.ibm.com"
  xmlns:payload="http://payload.soabench.ibm.com"
  xmlns:tns="http://soaassureservice.soabench.ibm.com">
    <soapenv:Header />
  ⊟<soapenv:Body>
    ⊟<tns:createClaimResponse>
        <tns:responseCode>Success</tns:responseCode>
      ⊟<tns:claim>
          <soabdata:claimId>c123456</soabdata:claimId>
          <soabdata:policyId>PS100001</soabdata:policyId>
```

Figure A-15   Message in clear text and no ws-sec header (Step two of six)

3. This step adds the ws-sec header to the message, as shown in Figure A-16.



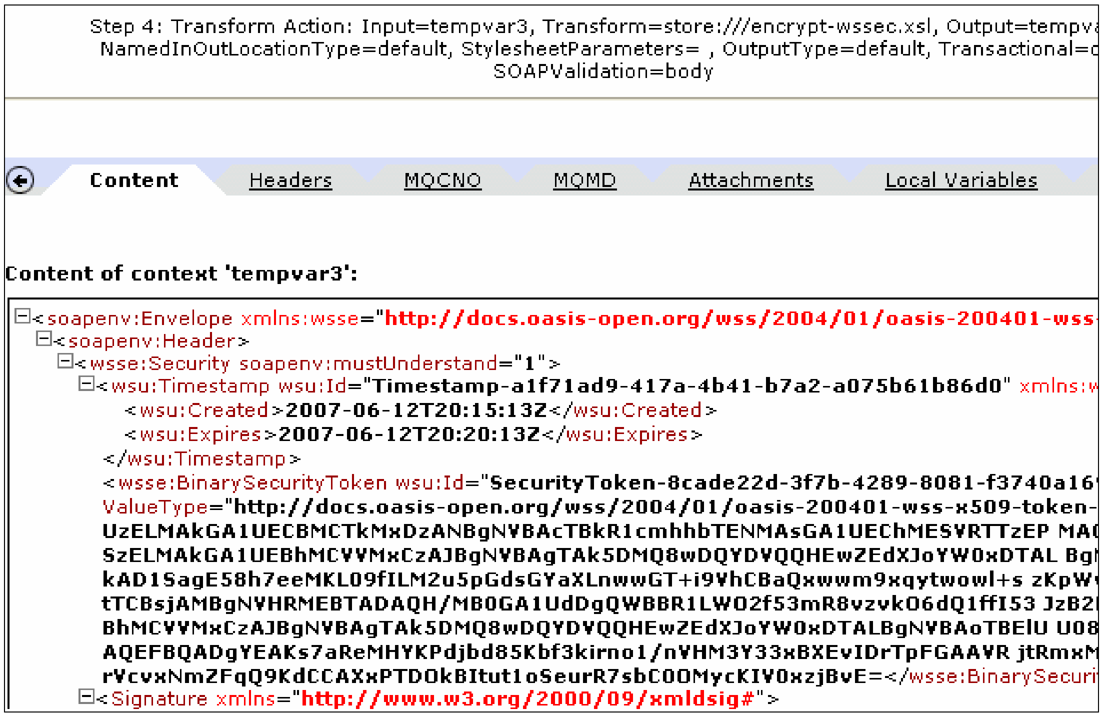
Step 3: Transform Action: Input=tempvar2, Transform=store:///sign-wssec.xsl, Output=tempva
NamedInOutLocationType=default, StylesheetParameters= , StylesheetParameters= , OutputType=d
Transactional=off, SOAPValidation=body

Content of context 'tempvar2':

```
⊟<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
  secext-1.0.xsd">
 ⊟<soapenv:Header>
   ⊟<wsse:Security>
     ⊟<wsse:UsernameToken>
         <wsse:Username>fred</wsse:Username>
         <wsse:Password>flintstone</wsse:Password>
       </wsse:UsernameToken>
     </wsse:Security>
   </soapenv:Header>
 ⊟<soapenv:Body>
   ⊟<tns:createClaimResponse xmlns:tns="http://soaassureservice.soabench.ibm.com"
      xmlns:payload="http://payload.soabench.ibm.com"
      xmlns:soabdata="http://data.soabench.ibm.com" xmlns:date="http://exslt.org/date
      and-times" xmlns:ctvf="http://www.contivo.com/xslt/extensions">
       <tns:responseCode>Success</tns:responseCode>
```

Figure A-16   Ws-sec headers are added using the XSL stylesheet (Step three of six)

4. Figure A-17 shows that the message is signed using the key certs in the sign action.



Step 4: Transform Action: Input=tempvar3, Transform=store:///encrypt-wssec.xsl, Output=tempv
NamedInOutLocationType=default, StylesheetParameters= , OutputType=default, Transactional=
SOAPValidation=body

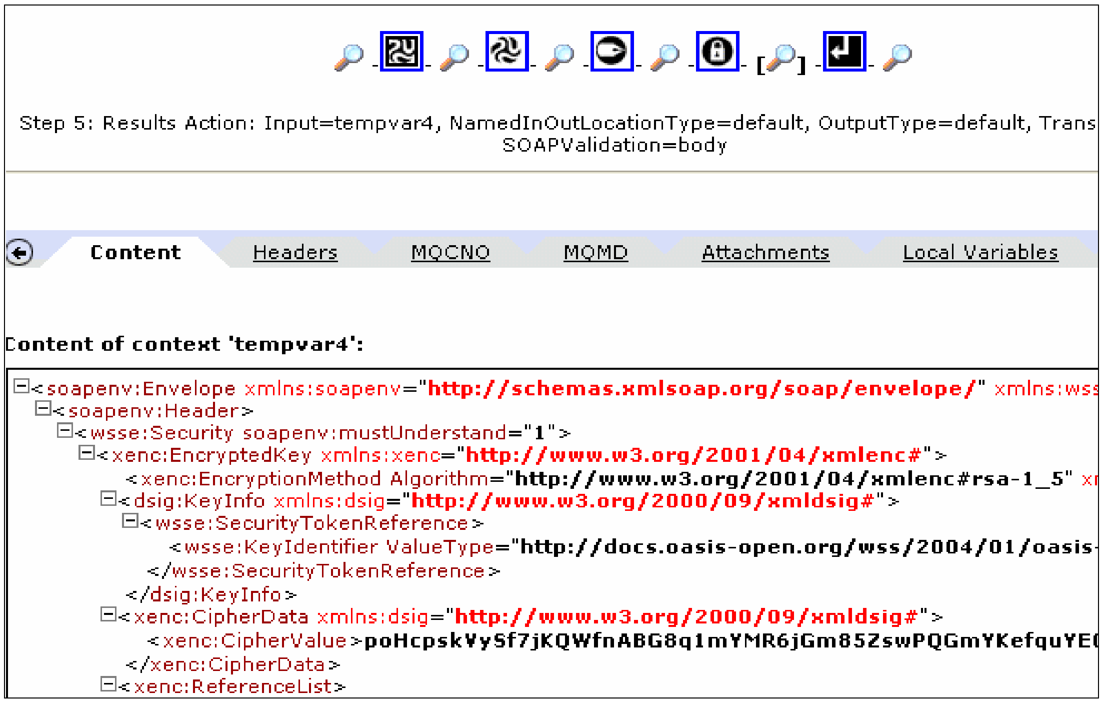
Content | Headers | MQCNO | MQMD | Attachments | Local Variables

Content of context 'tempvar3':

```
⊟<soapenv:Envelope xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss
 ⊟<soapenv:Header>
  ⊟<wsse:Security soapenv:mustUnderstand="1">
   ⊟<wsu:Timestamp wsu:Id="Timestamp-a1f71ad9-417a-4b41-b7a2-a075b61b86d0" xmlns:w
      <wsu:Created>2007-06-12T20:15:13Z</wsu:Created>
      <wsu:Expires>2007-06-12T20:20:13Z</wsu:Expires>
    </wsu:Timestamp>
    <wsse:BinarySecurityToken wsu:Id="SecurityToken-8cade22d-3f7b-4289-8081-f3740a16
    ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
    UzELMAkGA1UECBMCTkMxDzANBgNVBAcTBkR1cmhhbTENMAsGA1UEChMESVRTTzEP MA
    SzELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAk5DMQ8wDQYDVQQHEwZEdXJoYW0xDTAL Bg
    kAD1SagE58h7eeMKL09fILM2u5pGdsGYaXLnwwGT+i9VhCBaQxwwm9xqytwowl+s zKpW
    tTCBsjAMBgNVHRMEBTADAQH/MB0GA1UdDgQWBBR1LW02f53mR8vzvk06dQ1ffI53 JzB2
    BhMCVVMxCzAJBgNVBAgTAk5DMQ8wDQYDVQQHEwZEdXJoYW0xDTALBgNVBAoTBElU U08
    AQEFBQADgYEAKs7aReMHYKPdjbd85Kbf3kirno1/nVHM3Y33xBXEvIDrTpFGAAVR jtRmx
    rVcvxNmZFqQ9KdCCAXxPTD0kBItut1oSeurR7sbC0OMycKIV0xzjBvE=</wsse:BinarySecur
   ⊟<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
```

Figure A-17 Signed message (Step four of six)

5. The message is successfully encrypted with security credentials that were provided in the sign action, as shown in the Figure A-18.



Step 5: Results Action: Input=tempvar4, NamedInOutLocationType=default, OutputType=default, Trans
SOAPValidation=body

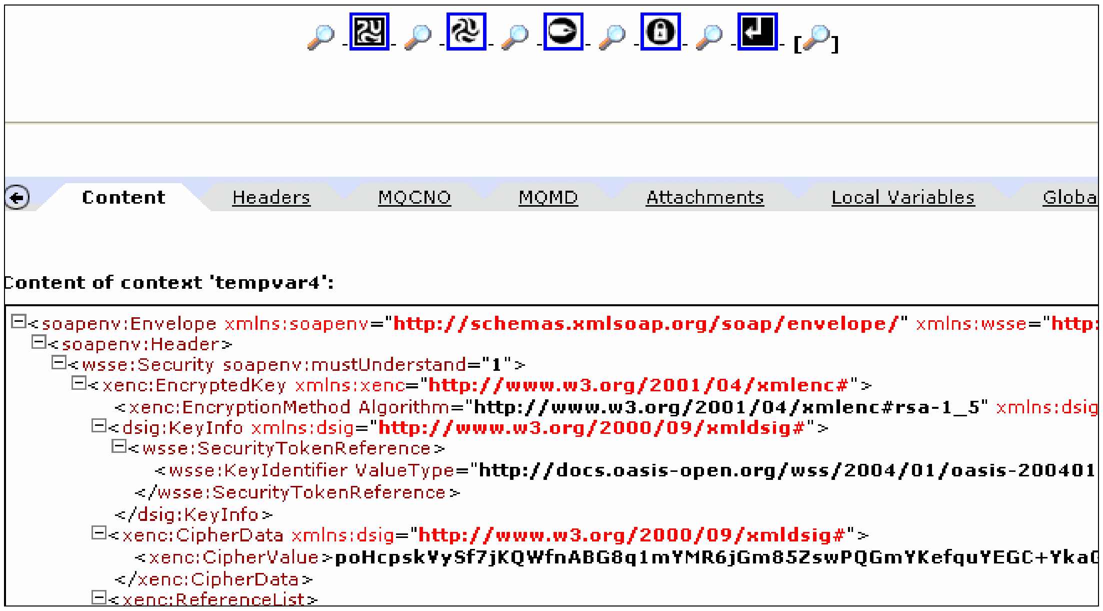
Content | Headers | MQCNO | MQMD | Attachments | Local Variables

Content of context 'tempvar4':

```
⊟<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wss
 ⊟<soapenv:Header>
  ⊟<wsse:Security soapenv:mustUnderstand="1">
   ⊟<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" x
    ⊟<dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
     ⊟<wsse:SecurityTokenReference>
        <wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/2004/01/oasis
       </wsse:SecurityTokenReference>
      </dsig:KeyInfo>
    ⊟<xenc:CipherData xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
       <xenc:CipherValue>poHcpskVy5f7jKQWfnABG8q1mYMR6jGm85ZswPQGmYKefquYE
      </xenc:CipherData>
    ⊟<xenc:ReferenceList>
```

Figure A-18 Encrypted message (Step five of six)

6. Figure A-19 shows the successfully signed and encrypted message that has been sent to the Web-service client. This is the same response that is shown on the command line panel.



Content of context 'tempvar4':

```
⊟<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsse="http:
  ⊟<soapenv:Header>
    ⊟<wsse:Security soapenv:mustUnderstand="1">
      ⊟<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
          <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" xmlns:dsig
        ⊟<dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
          ⊟<wsse:SecurityTokenReference>
              <wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401
            </wsse:SecurityTokenReference>
          </dsig:KeyInfo>
        ⊟<xenc:CipherData xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
            <xenc:CipherValue>poHcpskVySf7jKQWfnABG8q1mYMR6jGm85ZswPQGmYKefquYEGC+Ykа
          </xenc:CipherData>
        ⊟<xenc:ReferenceList>
```

Figure A-19 Final message sent to client (Step six of six)

B

Building message flows in WebSphere Message Broker

In this appendix, we explain how to build message flows. It is not our intention to provide a detailed explanation about how to build message sets and message flows, because the primary objective of this Redpaper is to explain DataPower Integration with WebSphere Message Broker (WMB). We expect you to have basic knowledge of message broker development skills using the IBM WebSphere Message Brokers Toolkit. You can get detailed information about the WMB Toolkit from the WebSphere Message Broker Information center at this Web site:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp

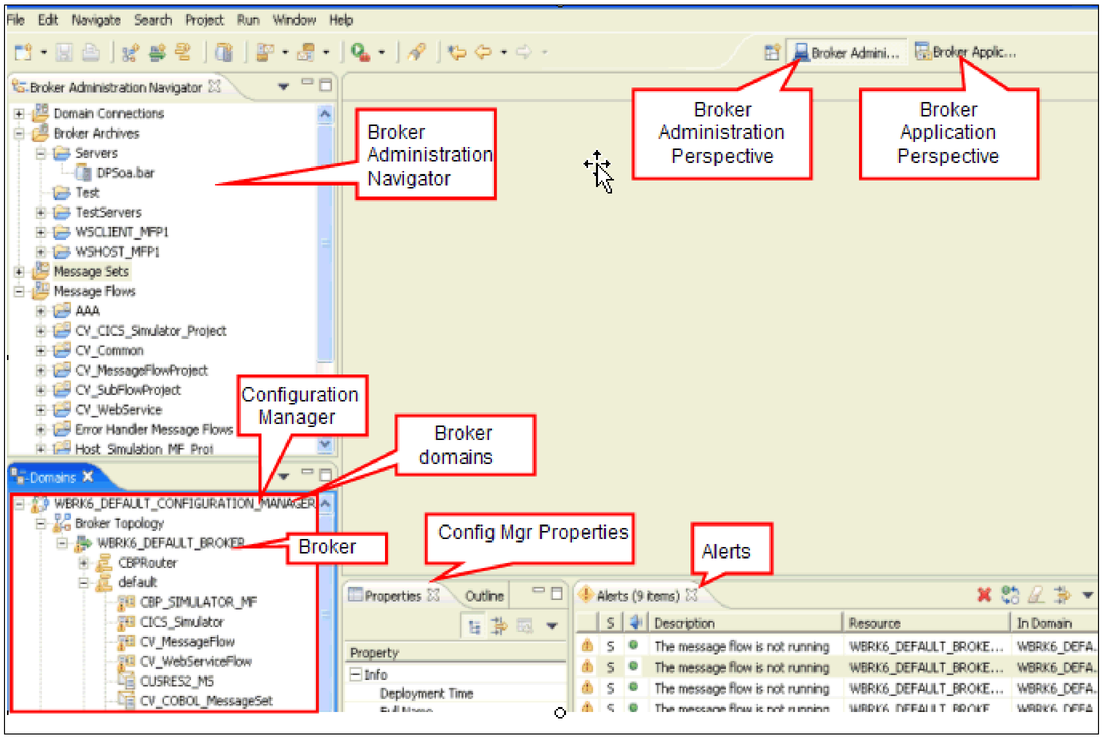
In this section, we explain how to complete the following tasks:

- Create message definition files used by both scenarios
- Build ErrorHandler Flow
- Build Host Simulation Flow
- Build message flows for scenario one
- Build message flows for scenario two
- Deploy message sets and the message flows

Introduction to WMB Toolkit

Based on the Eclipse platform, the WMB Toolkit provides a flexible and rich development environment for developers to build application integration. It provides two major perspectives: *Broker Administration* and *Broker Application Development*. You use the Broker Application Development perspective to develop message broker artifacts and the Broker Administration perspective to deploy code and administer brokers in the domains. You switch between these perspectives. It is important that you work with the correct perspective, as instructed.

Before we start, it is essential for you to know where to find the parts of WMB Toolkit. Figure B-1 shows an overview of the WMB Toolkit V6.0.2. We refer to these parts throughout this appendix.



S	Description	Resource	In Domain
S	The message flow is not running	WBRK6_DEFAULT_BROKE...	WBRK6_DEFA
S	The message flow is not running	WBRK6_DEFAULT_BROKE...	WBRK6_DEFA
S	The message flow is not running	WBRK6_DEFAULT_BROKE...	WBRK6_DEFA
S	The message flow is not running	WBRK6_DEFAULT_BROKE	WBRK6_DEFA

Figure B-1 WebSphere Message Brokers Toolkit

You must first create the message definition files that define the data structures that are used by the message flows in the first step.

Building the message definition files

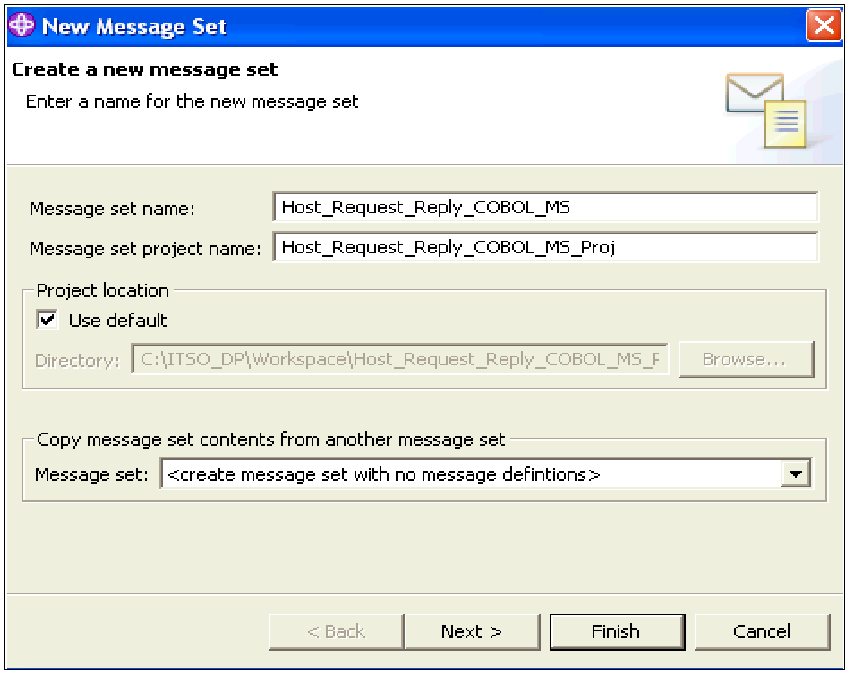
The WMB Toolkit allows you to define the message definition from scratch or to import existing COBOL copy books or XML schemas to define the message definition files that are used for transformation. We use the easier path to import an existing COBOL copy book, an XML schema, and a Web Services Description Language (WSDL) to create message definition files for these scenarios.

You must download and unzip artifacts into the directory C:\ITSO_DP\Artifacts as described in Appendix C, "Additional material" on page 147. The COBOL copy book, the XML schema, and the WSDL needed to build the message definition files are in that directory.

The steps to build the message definition files are:

1. Create a message set from the COBOL copy book:
   a. Click **Start** → **Program Files** → **WebSphere Message Brokers Toolkit**.
   b. Select the **Broker Application Development** perspective as shown in Figure B-1 on page 120.
   c. Click **File** → **New** → **Message Set**.

      Set the Message set name to `Host_Request_Reply_COBOL_MS` and the Message Set project name to `Host_Request_Reply_COBOL_MS_Proj` as shown in Figure B-2.



Message set name: Host_Request_Reply_COBOL_MS

Message set project name: Host_Request_Reply_COBOL_MS_Proj

Project location

☑ Use default

Directory: C:\ITSO_DP\Workspace\Host_Request_Reply_COBOL_MS_F

Copy message set contents from another message set

Message set: <create message set with no message defintions>

Figure B-2 Create Message Set project

   d. Select **Binary Data** and click **Finish.** You can click **OK** on the Tip pop-up menu.
   e. Import the COBOL copy book `Customer1.cpy` from C:\ITSO_DP\DataStruc\Customer1.cpy into the project:
      i. Right-click **Host_Request_Reply_COBOL_MS_Proj**, and then click **Import** → **File System** → **Next**.
      ii. Click **Browse** to select **C:\ITSO_DP\Artifacts\Customer1.cpy**, and click **Finish**.
   f. Create a message definition file from the COBOL copy book (Figure B-3 on page 122):
      i. Right-click **Customer1.cpy** and click **New** → **Message Definition File From**.
      ii. **COBOL File** is preselected. Select **Customer1.cpy** and click **Next**.

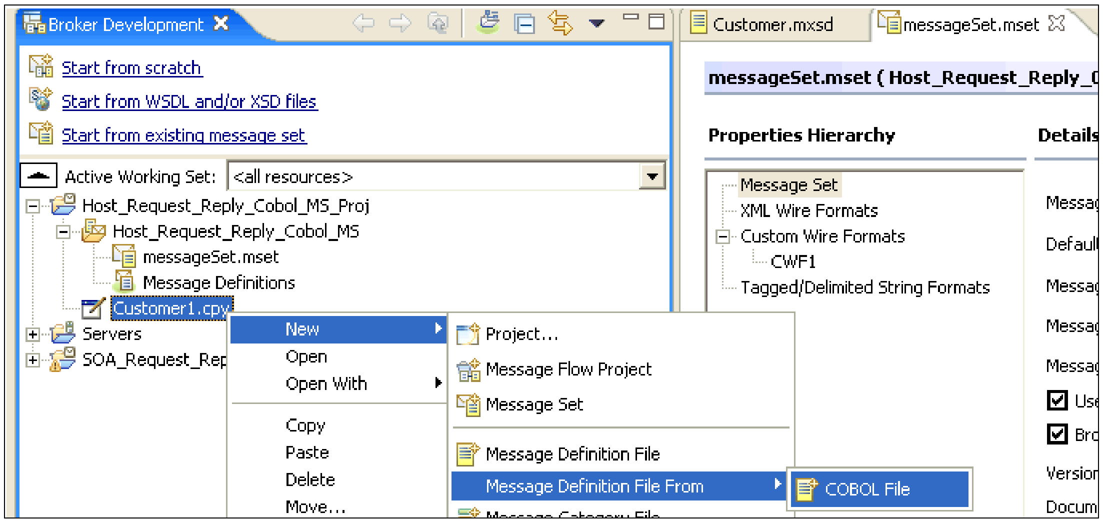


Figure B-3 Import COBOL copy book

iii. In the New Message Definition File panel (Figure B-4), select **CUSTOMER** under the Source structures section and click the right arrow symbol (**>**), which is highlighted in red, to move it to the Imported structures section. Click **Next.**

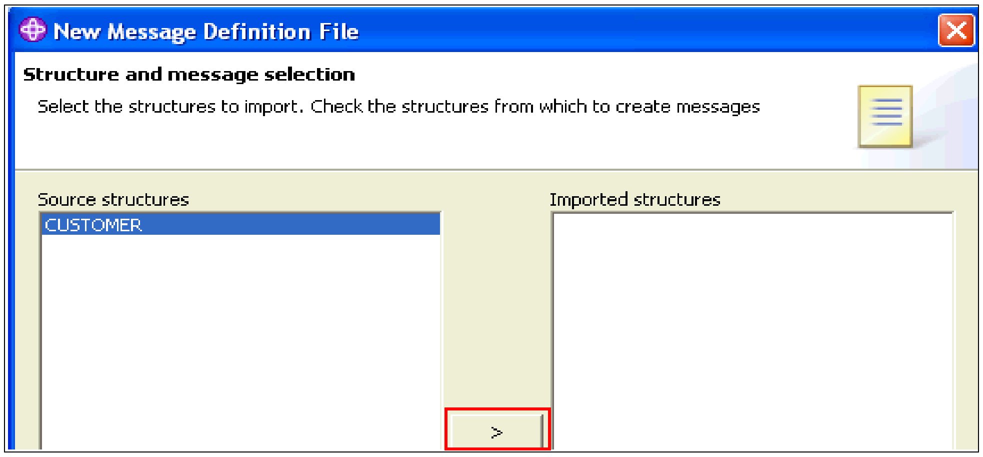
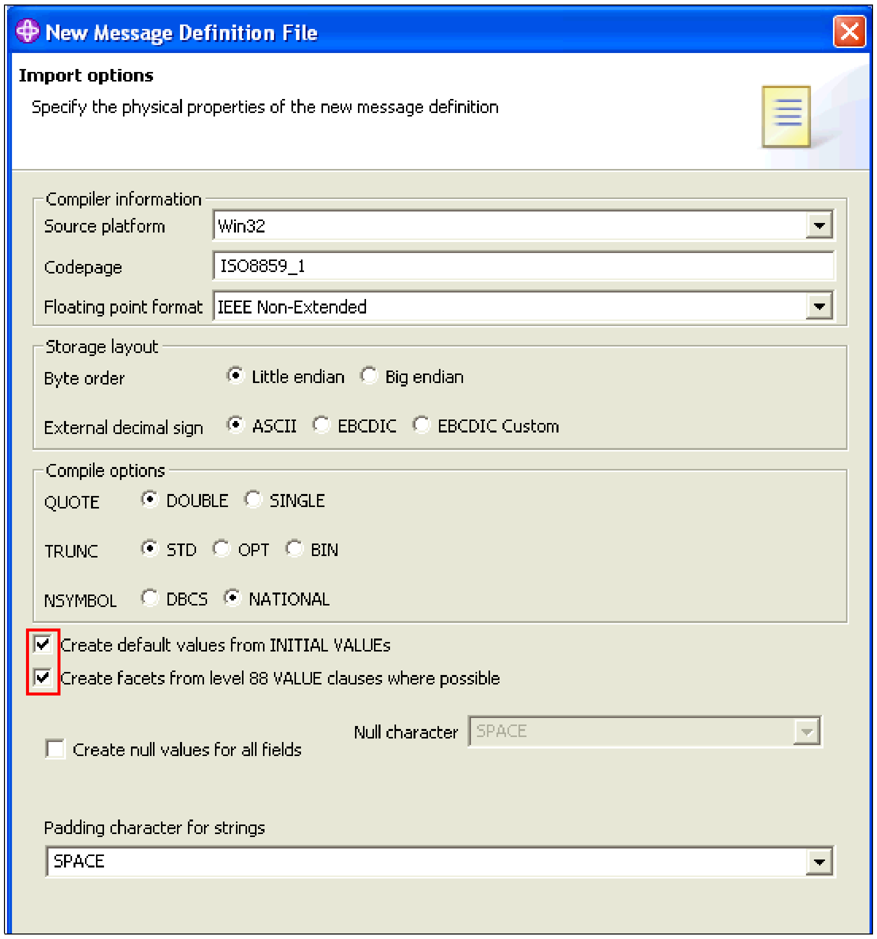

New Message Definition File
Structure and message selection
Select the structures to import. Check the structures from which to create messages
Source structures
CUSTOMER
Imported structures
>

Figure B-4 Import COBOL copy book continued

iv. In Figure B-5 on page 123, select **Create default values from INITIAL VALUEs** and **Create facets from level 88 VALUE clauses where possible**.


New Message Definition File

Import options

Specify the physical properties of the new message definition

Compiler information
- Source platform: Win32
- Codepage: ISO8859_1
- Floating point format: IEEE Non-Extended

Storage layout
- Byte order: [✓ Little endian] Big endian
- External decimal sign: [✓ ASCII] EBCDIC EBCDIC Custom

Compile options
- QUOTE: [✓ DOUBLE] SINGLE
- TRUNC: [✓ STD] OPT BIN
- NSYMBOL: DBCS [✓ NATIONAL]

- [✓] Create default values from INITIAL VALUEs
- [✓] Create facets from level 88 VALUE clauses where possible
- [ ] Create null values for all fields

Null character: SPACE

Padding character for strings: SPACE

Figure B-5 Import COBOL copy book continued

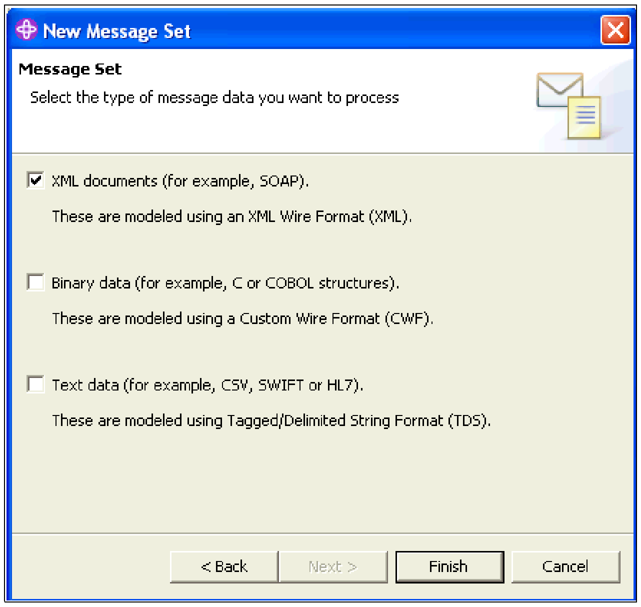
v. Accept all default values and click **Finish** to complete.

2. Create the SOAP message definition:

   a. Click **File** → **New** → **Message Set**.

   b. Set the Message set name to `SOA_Request_Reply_XML_MS` and the Message Set project name to `SOA_Request_Reply_XML_MS_Proj`.

   c. In Figure B-6 on page 124, select **XML documents** and click **Finish.** You can click **OK** on the Tip pop-up menu.


New Message Set

Message Set

Select the type of message data you want to process

☑ XML documents (for example, SOAP).

These are modeled using an XML Wire Format (XML).

☐ Binary data (for example, C or COBOL structures).

These are modeled using a Custom Wire Format (CWF).

☐ Text data (for example, CSV, SWIFT or HL7).

These are modeled using Tagged/Delimited String Format (TDS).

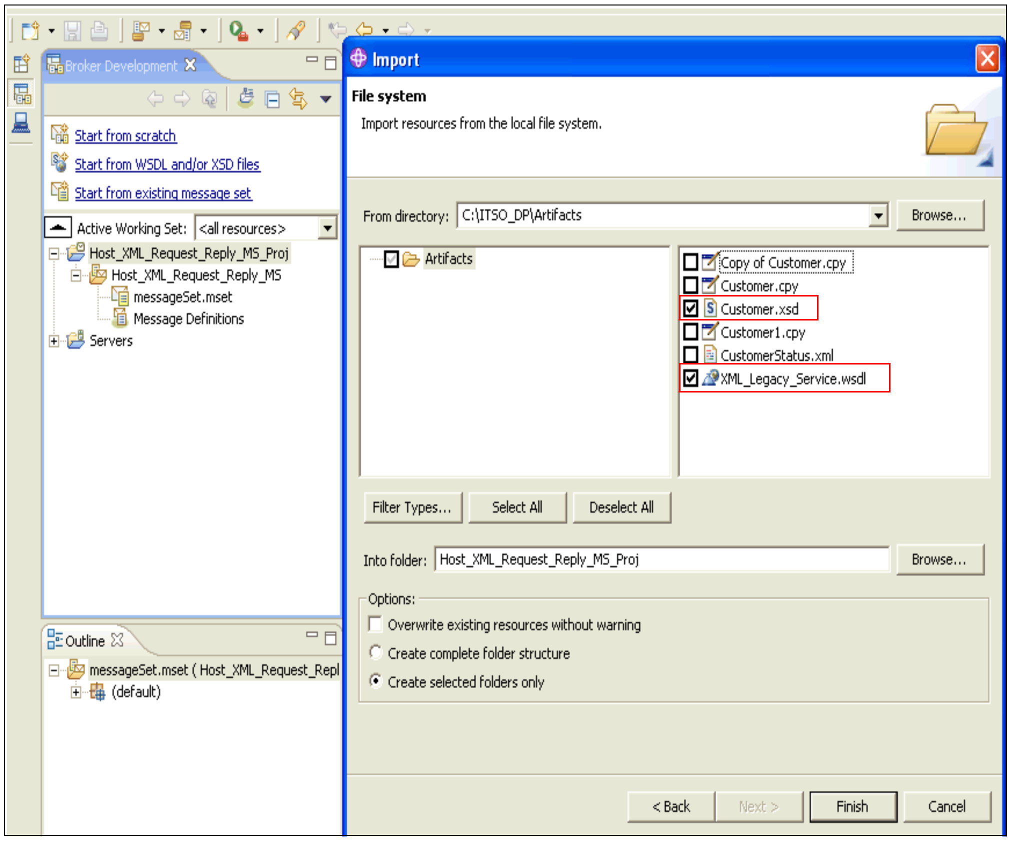
Figure B-6 Create the new message set project for XML and WSDL documents

d. Import `Customer.xsd` and `XML_Legacy_Service.wsdl`:

  i. Right-click **Host_Request_Reply_XML_MS_Proj**, and then click **Import** → **File System** → **Next**.

  ii. Click **Browse** to select the **C:\ITSO_DP\Artifacts** folder and select **Customer.xsd** and **XML_Legacy_Service.wsdl** as shown in Figure B-7 on page 125.

  iii. Click **Finish** and save (press Ctrl + S keys).


Broker Development
Start from scratch
Start from WSDL and/or XSD files
Start from existing message set
Active Working Set: <all resources>
Host_XML_Request_Reply_MS_Proj
Host_XML_Request_Reply_MS
messageSet.mset
Message Definitions
Servers
Outline
messageSet.mset (Host_XML_Request_Repl
(default)
Import
Back
Next >
Finish
Cancel

File system

Import resources from the local file system.

From directory: C:\ITSO_DP\Artifacts Browse...

Artifacts

- [ ] Copy of Customer.cpy
- [ ] Customer.cpy
- [x] Customer.xsd
- [ ] Customer1.cpy
- [ ] CustomerStatus.xml
- [x] XML_Legacy_Service.wsdl

Filter Types... Select All Deselect All

Into folder: Host_XML_Request_Reply_MS_Proj Browse...

Options:

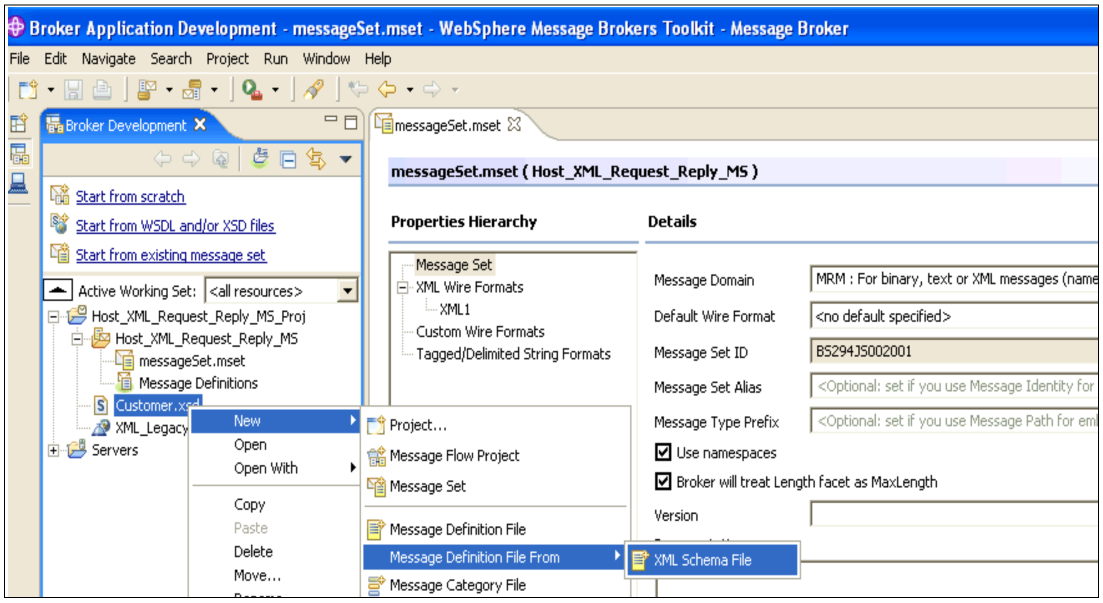
- [ ] Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

Figure B-7 Import Customer.xsd and XML_Legacy_Service.wsdl

e. To create the SOAP message definition file from the WSDL, you have to create message definitions for `Customer.xsd` (first) and `XML_Legacy_Service.wsdl` (second):

i. In Figure B-8, right-click **Customer1.xsd** and click **New** → **Message Definition File From**.

ii. **XML Schema File** is preselected. Click **Next**.


Broker Application Development - messageSet.mset - WebSphere Message Brokers Toolkit - Message Broker
File Edit Navigate Search Project Run Window Help
Broker Development
Start from scratch
Start from WSDL and/or XSD files
Start from existing message set
Active Working Set: <all resources>
Host_XML_Request_Reply_MS_Proj
Host_XML_Request_Reply_MS
messageSet.mset
Message Definitions
Customer.xsd
XML_Legacy
Servers
New
Open
Open With
Copy
Paste
Delete
Move...
Project...
Message Flow Project
Message Set
Message Definition File
Message Definition File From
Message Category File
XML Schema File
messageSet.mset

messageSet.mset (Host_XML_Request_Reply_MS)

Properties Hierarchy

- Message Set
  - XML Wire Formats
    - XML1
  - Custom Wire Formats
  - Tagged/Delimited String Formats

Details

Message Domain	MRM : For binary, text or XML messages (name
Default Wire Format	<no default specified>
Message Set ID	BS294JS002001
Message Set Alias	<Optional: set if you use Message Identity for
Message Type Prefix	<Optional: set if you use Message Path for em

- [x] Use namespaces
- [x] Broker will treat Length facet as MaxLength

Version

Figure B-8 Create message definition for Customer.xsd

iii. In Figure B-9, click **Select All** to select everything listed in the Global elements box and click **Finish**.

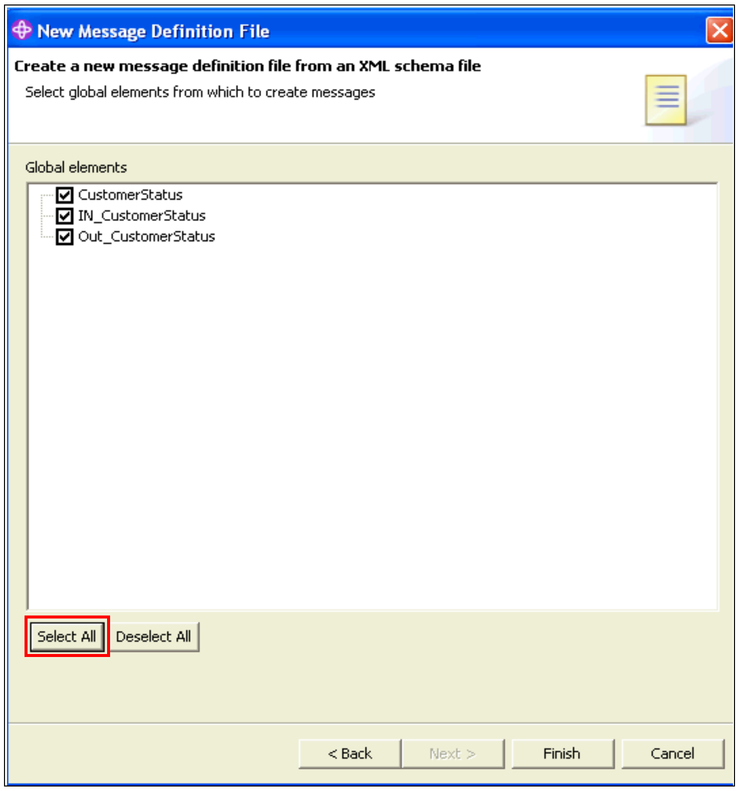

New Message Definition File
Create a new message definition file from an XML schema file
Select global elements from which to create messages
Global elements
CustomerStatus
IN_CustomerStatus
Out_CustomerStatus
Select All
Deselect All
< Back
Next >
Finish
Cancel

Figure B-9 Create message definition for customer.xsd (continued)

i. In Figure B-7 on page 125, right-click **XML_Legacy_Service.wsdl** and click **New → Message Definition File From**.

ii. **WSDL File** is preselected. Click **XML_Legacy_Service.wsdl** on the next panel, click **Next**, click **Next** again, and click **Finish**.

iii. Press the Ctrl + S keys to save the definitions.

You have created all the message definitions that will be used by the message flows.

Note: You will see many warnings in the Problems view of the WMB Toolkit. You can ignore these warnings as long as there are no errors in red.

Build the ErrorHandler subflow

This message flow performs error handling processing. This common flow is used by both scenarios.

In the WMB Toolkit, make sure that you are in the Broker Application Development perspective:

1. Click **File → New → Message Flow Project**.
2. Set Project name to `Error Handler Message Flow` and click **Finish**.
3. Drag and drop the nodes from the Palette to the Message Flow canvas and rename the nodes to the names listed under Node name. Table B-1 on page 127 shows you where to

find each node by its type from the Palette folder name. Right-click the node and select **rename** and enter the new name.

Table B-1 List of node type, Palette folder name, and node name

Node type	Palette folder name	Node name (rename to)
Input	Construction	Start Subflow
Compute	Transformation	ExceptionList
Flow Order	Construction	FlowOrder
Trace	Construction	Trace
MQOutput	WebSphere MQ	CATCH_ALL_ERRORS

4. Connect the nodes in the subflow as shown in Table B-2. The name of the node terminal is in parentheses.

Table B-2 Instructions for wiring the nodes in the ErrorHandler subflow

Node and terminal name	Node and terminal name
Input (out)	ExceptionList (in)
ExceptionList (out)	FlowOrder (in)
FlowOrder (first)	CATCH_ALL_ERRORS (in)
FlowOrder (second)	Trace (in)
Trace (in)	Throw (in)

The subflow you built needs to look like the one in Figure B-10.

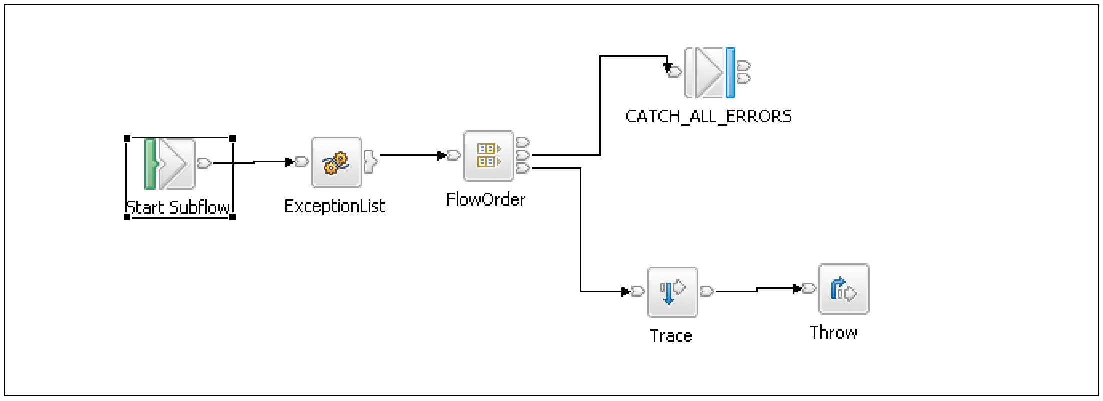


Figure B-10 ErrorHandler subflow

5. Right-click the **ExceptionList** node and select **Open ESQL**. Replace the existing code in the BEGIN and END block with the sample code in Example B-1 on page 128. This code parses the exception list to retrieve the last exception number and error message.

Example: B-1 ExceptionList ESQL code

```
CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
      DECLARE Path CHAR;
      DECLARE ExpLstPTR REFERENCE TO InputExceptionList.*[1];
      SET Path = 'Environment.Variables.BrokerData';
      WHILE ExpLstPTR.Number IS NOT NULL do
         SET Environment.Variables.BrokerData.LastError.Label =
ExpLstPTR.Label;
         SET Environment.Variables.BrokerData.LastError.Number =
         cast(ExpLstPTR.Number as char);
         SET Environment.Variables.BrokerData.LastError.Text = ExpLstPTR.Text;
         SET Path = Path || '.*[<]';
         -- Move start to the last child of the field to which it
         -- currently points
         MOVE ExpLstPTR LASTCHILD;
      END WHILE;
      SET Path = 'Environment.Variables.BrokerData';
      CALL CopyMessageHeaders();
      SET OutputRoot.XML.Message.Environment.Variables.UserData =
Environment.Variables.UserData;
      SET OutputRoot.XML.Message.Environment.Variables.BrokerData =
      Environment.Variables.BrokerData;
      RETURN TRUE;
   END;
```

6. Set the properties of the CATCH_ALL_ERRORS node:
   a. Right-click the **CATCH_ALL_ERRORS** node and click **Properties**.
   b. Click the **Basic** tab and fill in the Queue name `CATCH_ALL_ERRORS`.

Building the host simulation flow

For the convenience of testing, we provide a host simulation flow for you to test the scenarios without needing CICS/COBOL running in your environment. The steps to build this message flow are:

1. In the Message Brokers Toolkit, switch to the **Broker Application Development** perspective.
2. Click **File** → **New** → **Message Flow Project**.
3. Set Project name to `Host_Simulation_MF` and click **Finish**.
4. Drag and drop the nodes from the Palette onto the Message Flow canvas and rename the nodes to the names listed under Node name in Table B-3 on page 129.

Table B-3 List of node type, Palette folder name, and node name

Node type	Palette folder name	Node name (rename to)
MQInput	WebSphere MQ	HOST_REQUEST
Compute	Transformation	Set_Customer_Info_Status
TryCatch	Construction	TryCatch
MQReply	Construction	MQReply
Subflow	N/A	ErrorHandler

5. Connect the nodes as described in Table B-4.

Table B-4 Instructions for wiring the nodes in Host_Simulation_MF flow

Input node and terminal name	Output node and terminal name
HOST_REQUEST (out)	TryCatch (in)
TryCatch (try)	Set_Customer_Info_Status (in)
TryCatch (catch)	ErrorHandler subflow
Set_Customer_Info_Status (out)	MQReply

6. Set the properties of the HOST_REQUEST node:
   a. Right-click the **HOST_REQUEST** node and click **Properties**.
   b. Click the **Basic** tab and set the Queue Name to `HOST_REQUEST`.
   c. Select the **Input Message Parsing** tab.
   d. Select **MRM** from the Message Domain drop-down list.
   e. Select **Host_Request_Reply_Cobol_MS** from the Message Set drop-down list.
   f. Select **msg_CUSTOMER** from the Message Type drop-down list.
   g. Select **CWF1** for the Message Format from the drop-down list.

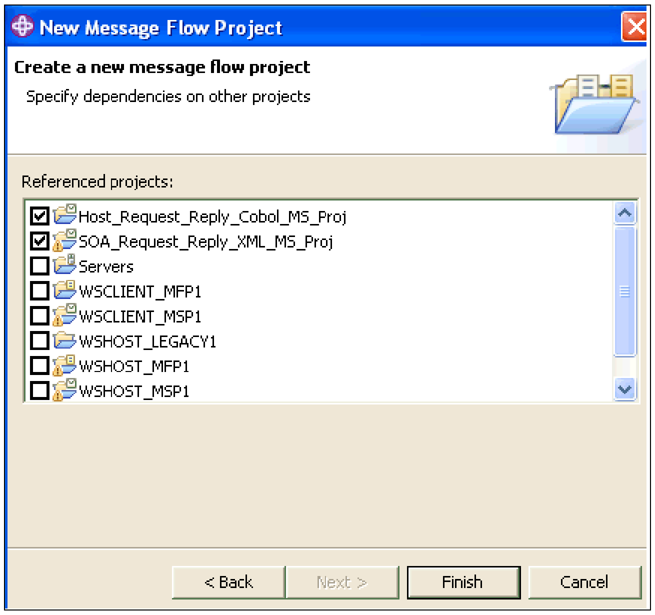
Building message flows for scenario one

Follow these steps:

1. Build the message flow `WS_To_Legacy_MF`.

   This message flow has two HTTPInput nodes for receiving request messages from DataPower via secured or non-secured HTTP connections. It invokes the CICS application via MQ, gets the response from CICS, and transforms it from COBOL to XML. A reply message is sent to DataPower via the HTTP Reply node:

   a. Click **File** → **New** → **Message Flow Project**.
   b. Set Project name to `WS_To_Legacy_MF_project` and then click **Finish**.
   c. As shown in Figure B-11 on page 130, select **SOA_Request_Reply_XML_MS_Proj** and **Host_Request_Reply_Cobol_MS_Proj** in the Referenced project pane. Click **Finish**.



Create a new message flow project

Specify dependencies on other projects

Referenced projects:

- [✓] Host_Request_Reply_Cobol_MS_Proj
- [✓] SOA_Request_Reply_XML_MS_Proj
- [ ] Servers
- [ ] WSCLIENT_MFP1
- [ ] WSCLIENT_MSP1
- [ ] WSHOST_LEGACY1
- [ ] WSHOST_MFP1
- [ ] WSHOST_MSP1

Figure B-11 Set referenced projects

d. Right-click **WS_To_Legacy_MF_Proj**, and then click **New** → **Message Flow**.

e. Set Message flow name to `WS_To_Legacy_MF`, and then click **Finish**.

f. Drag and drop the following nodes from the palette onto the Message Flow canvas:

- Two HTTPInput nodes
- Three Compute nodes
- One TryCatch node
- Four Output nodes
- One MQGET node
- One HTTP Reply node

g. Rename the nodes according to Table B-5. To rename a node, right-click the node and then click **Rename**. We used the names listed in the table. Optionally, you can use other names.

Table B-5 List of node type, Palette folder name, and node name

Node type	Palette folder name	Node name
HTTPInput Node	HTTP	HTTP No Security
HTTPInput Node Security	HTTP	HTTP Security
Compute Node 1	Transformation	XML_To_COBOL
Compute Node 2	Transformation	COBOL_To_XML
Compute Node 3	Transformation	Set_HTTP_Id
Output Node 1	WebSphere MQ	ERROR
Output Node 2	WebSphere MQ	HTTP_STATUS_SAVE
Output Node 3	WebSphere MQ	HOST_REQUEST
Output Node 4	WebSphere MQ	HOST_REPLY

h. Connect the nodes and terminals according to Table B-6. The name of the node terminal is in parentheses.

Table B-6 Instructions for wiring the nodes in WS_To_Legacy_MF message flow

Node and terminal name (from)	Node and terminal name (to)
HTTP No Security (in)	TryCatch (in)
HTTP Security (in)	TryCatch (in)
TryCatch (try)	XML_To_COBOL (in)
TryCatch (catch)	Error_Handler subflow (in)
XML_To_COBOL (out)	HOST_REQUEST (in)
HOST_REQUEST (out)	Set_HTTP_Id (in)
Set_HTTP_Id (out)	HTTP_STATEQ (in)
HOST_REPLY (out)	TryCatch (in)
TryCatch (try)	MQGET (in)
TryCatch (catch)	Error_Handler (in)
MQGET (out)	COBOL_To_XML
MQGET (warning)	MQGET_ERROR
MQGET (no message)	MQGET_ERROR
COBOL_To_XML (out)	HTTP Reply

i. After you have connected all nodes, the message flow needs to look like the message flow shown in Figure B-12 on page 131.

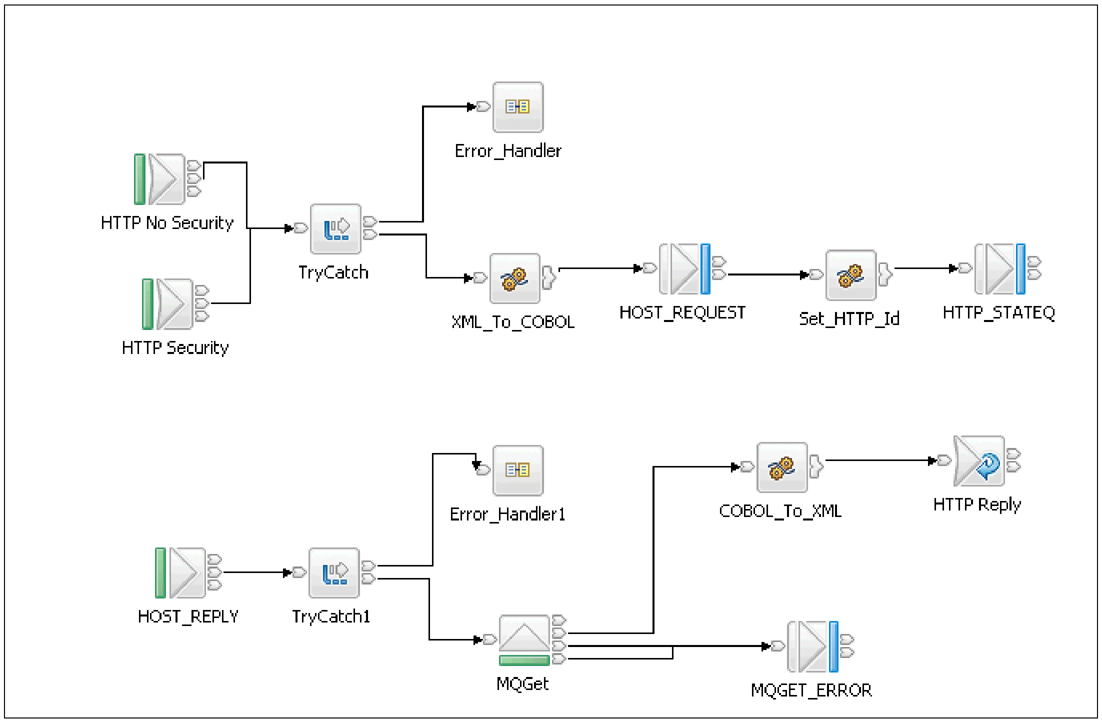


Figure B-12 Completed WS_To_Legacy_MF message flow

j. Set the properties of the HTTP No Security Input Node:
   i. Click the **Basic** tab and set URL Selector to **/HTTPNoSec/AccountStatus**.
   ii. Click the **Input Message Parsing** tab.
   iii. Select **MRM** from the Message domain drop-down list.
   iv. Select **SOA_Request_Reply_XML_MS** from the Message Set drop-down list.
   v. Select **Envelope** from the Message Type drop-down list.
   vi. Select **XML1** from the Message Format drop-down list.
   vii. Click **OK.**

k. Set the properties of the HTTP Security Input node:
   i. Click the **Basic** tab and set URL Selector to **/HTTPNoSec/AccountStatus**.
   ii. Select **Use HTTPS**.
   iii. Click the **Input Message Parsing** tab.
   iv. Select **MRM** from the Message domain drop-down list.
   v. Select **SOA_Request_Reply_XML_MS** from the Message Set drop-down list.
   vi. Select **Envelope** from the Message Type drop-down list.
   vii. Select **XML1** from the Message Format drop-down list.
   viii. Click **OK**.

l. Right-click the **XML_TO_COBOL** compute node and select **Open ESQL**. Replace the existing code in the BEGIN and END block with the sample code in Example B-2 on page 132.

Example: B-2 Create compute module XML to COBOL

```
CREATE COMPUTE MODULE XML_To_COBOL
   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
      -- Declare namespace after the BEGIN statement
      DECLARE tns NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
      DECLARE ns  NAMESPACE 'http://www.itso.lab.com';

      CALL CopyMessageHeaders();
      -- Remove HTTP headers - Output is and MQ Message
       SET OutputRoot.HTTPInputHeader = null;

      -- Output is legacy, so set wire format and message name
      SET OutputRoot.Properties.MessageSet = 'Host_Request_Reply_Cobol_MS';
      SET OutputRoot.Properties.MessageType = 'msg_CUSTOMER';
      SET OutputRoot.Properties.MessageFormat = 'CWF1';

      -- Add an MQMD for output message
      CREATE NEXTSIBLING OF OutputRoot.Properties DOMAIN 'MQMD';
      SET OutputRoot.MQMD.Version = 2;
      SET OutputRoot.MQMD.ReplyToQ = 'HOST_REPLY';
      SET OutputRoot.MQMD.PutApplName = 'WMB';
      SET OutputRoot.MQMD.PutDate = CURRENT_DATE;

      DECLARE CustIn_Ref REFERENCE TO
InputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo;
--        DECLARE CustOut_Ref REFERENCE TO OutputRoot.MRM.CUSTOMER_INFO;
```

```
      DECLARE CustOut_Ref REFERENCE TO OutputRoot.MRM;

      -- Transform the message form XML to Cobol to be sent to legacy
        SET OutputRoot.MRM.ACCOUNT_NUMBER =
InputRoot.MRM.tns:Body.ns:IN_CustomerStatus.accountNumber;
   RETURN TRUE;
   END;
```

m. Set the properties of the HOST_REQUEST node:
   i. Right-click the **HOST_REQUEST** node and click **Properties**.
   ii. Click the **Basic** tab and set Queue Name to **HOST_REQUEST**.
   iii. Select the **Input Message Parsing** tab.
   iv. Select **MRM** from the Message Domain drop-down list.
   v. Select **Host_Request_Reply_Cobol_MS** from the Message Set drop-down list.
   vi. Select **msg_CUSTOMER** from the Message Type drop-down list.
   vii. Select **CWF1** from the Message Format drop-down list.
   viii.Click the **Advanced** tab and set Transaction Mode to **Yes**.
   ix. Set Message Context to **Default**.
   x. Select the **Request** tab and select the check box for **Request**.
   xi. Set Reply-to Queue to **HOST_REPLY** and then click **OK**.
n. Right-click the **Set_HTTP_Id** compute node and select **Open ESQL**. Replace the existing code in the BEGIN and END block with the sample code in Example B-3 on page 133.

Example: B-3 Create compute module Set_HTTP_Id

```
CREATE COMPUTE MODULE Set_HTTP_Id
   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
-- The output message built in this node stores the HTTP context.
-- Any format may be suitable, but we are using a
-- self-defined field in a SOAP Header to do this
-- So the wire format is XML
-- Create the target message set properties
   DECLARE tns NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
   DECLARE ns  NAMESPACE 'http://www.itso.lab.com';
   SET OutputRoot.Properties.MessageSet = 'SOA_Request_Reply_XML_MS';
   SET OutputRoot.Properties.MessageType = 'Envelope';
   SET OutputRoot.Properties.MessageFormat = 'XML1';

-- Create the MQMD and set the CorrelId
-- The CorrelId needs to be set to the MsgId of the message sent
-- to the legacy application.  When the reply from the legacy
-- application is received, we will MQGet from the store
-- queue with correlid equal to the correlid of the legacy reply

   SET OutputRoot.MQMD.CorrelId
= InputLocalEnvironment.WrittenDestination.MQ.DestinationData.msgId;

-- Build message to store HTTP context
```

```
SET OutputRoot.MRM.tns:Header.HTTP.RequestIdentifier
 = InputLocalEnvironment.Destination.HTTP.RequestIdentifier;

RETURN TRUE;
END;
```

o. Set the properties of the HTTP_STATEQ node:
   i. Right-click the **HTTP_STATEQ** node and then click **Properties**.
   ii. Click the **Basic** tab and set the Queue Name to **HTTP_STATEQ.**
   iii. Click the **Advanced** tab and set Transaction Mode to **Yes** if you want to execute the message flow under a transaction.
   iv. Set Message Context to **Default**.
   v. Click **OK**.
p. Set the properties of the HOST_REPLY input node:
   i. Right-click the **HOST_REPLY** input node, and then click **Properties**.
   ii. Select **Basic** and set the Queue Name to **HOST_REPLY**.
q. Set the properties for the MQGet (get saved HTTP state from HTTP_STATEQ) node:
   iii. Right-click the **MQGet** node and click **Properties**.
   iv. Select the **Basic** tab and set the Queue Name to **HTTP_STATEQ**.
   v. Select the **Input Message Parsing** tab.
   vi. Select **MRM** from the Message Domain drop-down list.
   vii. Select **SOA_Request_Reply_XML_MS** from the Message Set drop-down list.
   viii.Select **Envelope** from the Message Type drop-down list.
   ix. Select **XML1** for Message Format from the drop-down list.
   x. Click the **Advanced** tab.
   xi. Select **Message and LocalEnvironment** from the drop-down list.
   xii. Select **Copy Entire Message** from the Copy Message drop-down list.
   xiii.Select **Request**, and select **Get by Correlation ID**.
   xiv.Select **Result** and set Output Data Location to **OutputLocalEnvironment**.
   xv. Click **OK**.
r. Set up the COBOL_TO_XML compute node:
   i. Right-click the **COBOL_TO_XML** compute node and click **Properties**.
   ii. Selectthe **Basic** tab and set Compute Mode to **LocalEnvironment and Message**.
   iii. Click **OK** (to allow HTTP context to be passed to HTTPReply node).
   iv. Right-click **COBOL_TO_XML** compute node and select **Open ESQL**. Replace the existing code in the BEGIN and END block with the sample code in Example B-4.

Example: B-4 Create compute module COBOL to XML

```
CREATE COMPUTE MODULE COBOL_To_XML
   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
   -- Declare namespace after the BEGIN statement
   DECLARE tns NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
   DECLARE ns  NAMESPACE 'http://www.itso.lab.com';
```

```
-- Input is CWF, output is SOAP/XML
   SET OutputRoot.Properties = InputRoot.Properties;
   SET OutputRoot.Properties.MessageSet = 'SOA_Request_Reply_XML_MS';
   SET OutputRoot.Properties.MessageType = 'Envelope';
   SET OutputRoot.Properties.MessageFormat = 'XML1';
   DECLARE CustOut_Ref REFERENCE TO
OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus;
   DECLARE CustIn_Ref REFERENCE TO InputRoot.MRM.CUSTOMER_INFO;

   SET OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.accountNumber =
InputRoot.MRM.ACCOUNT_NUMBER;
   SET OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.status =
InputRoot.MRM.CUSTOMER_INFO.CUSTOMER_STATUS;
   SET
OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.firstName =
InputRoot.MRM.CUSTOMER_INFO.FIRST_NAME;
   SET OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.lastName
= InputRoot.MRM.CUSTOMER_INFO.LAST_NAME;
   SET OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.street =
InputRoot.MRM.CUSTOMER_INFO.STREET;
   SET OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.city =
InputRoot.MRM.CUSTOMER_INFO.CITY;
   SET OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.state =
InputRoot.MRM.CUSTOMER_INFO.STATE;
   SET OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.country
= InputRoot.MRM.CUSTOMER_INFO.COUNTRY ;
   SET OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.zipCode
= InputRoot.MRM.CUSTOMER_INFO.ZIPCODE;
   SET OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.phone =
InputRoot.MRM.CUSTOMER_INFO.PHONE;

-- Set the http reply identifier
   SET OutputLocalEnvironment.Destination.HTTP.RequestIdentifier =
 CAST(InputLocalEnvironment.MRM.tns:Header.HTTP.RequestIdentifier AS
BLOB);

   RETURN TRUE;
   END;
```

s. Set the properties of the MQGET_ERROR node:
   i. Right-click the **MQGET_ERROR** node and click **Properties**.
   ii. Click the **Basic** tab and set Queue Name to **MQGET_ERROR**.

t. Set the properties of the HTTP Reply node:
   iii. Right-click the **HTTP Reply** node and click **Properties**.
   iv. Click the **Basic** tab. Click **Ignore transport failures** and **Generate default HTTP headers from reply or response**.

Build the message flows for scenario two

In the second scenario, DataPower and the message broker communicate via MQ. DataPower sends a request to the SOA_REQUEST queue, which is monitored by the message broker for incoming requests. The message broker transforms the incoming request and sends it to CICS for customer information. CICS processes the request and sends the data to the message broker. The message broker transforms the data, builds the reply message and sends it to DataPower, which in turn, sends it to the original requester.

Now, you will learn how to build this message flow as just described:

1. Create the message flow project `MQ_To_Legacy_MF_Proj` to process the request and reply message via MQ:
   a. Click **File** → **New** → **Message Flow Project**.
   b. Set Project name to `MQ_To_Legacy_MF_Proj`, and then click **Finish**.
   c. Select **SOA_Request_Reply_XML_MS_Proj** and **Host_Request_Reply_Cobol_MS_Proj** in the Referenced project pane.
2. Right-click **MQ_To_Legacy_MF_Proj**, and then click **New** → **Message Flow**:
   a. Set Message flow name to **StoreOriginalMQMD.msgflow**, and then click **Finish**.
   b. Drag and drop the following nodes from the palette onto the Message Flow canvas and rename them as shown in Table B-7 on page 136.

Table B-7 List of node type, Palette folder name, and node name

Node type	Palette folder name	Node name (rename)
Input	Construction	
FlowOrder	Construction	
Compute Node	Transformation	Save_Request_MQMD
MQ Output Node	WebSphere MQ	MQMD_STOREQ
Output	Construction	

   c. Connect the nodes and terminals according to Table B-8.

Table B-8 Instructions for wiring the nodes

Node and terminal name (from)	Node and terminal name (to)
Input (out)	FlowOrder (in)
FlowOrder (first)	Save_Request_MQMD (in)
FlowOrder (second)	Output (in)
Save_Request_MQMD (out)	MQMD_STOREQ (in)

   d. Set the properties of the nodes in StoreOriginalMQMD.msgflow:

      Set the properties of the MQMD_STOREQ node:

      i. Right-click the node **MQMD_STOREQ** and click **Properties**.
      ii. Click the **Basic** tab and set Queue Name to **MQMD_STOREQ**.
   e. Right-click the **Save_Request_MQMD** compute node and select **Open ESQL**. Replace the existing code in the BEGIN and END block with the sample code in Example B-5.

Example: B-5 ESQL code for Save_Request_MQMD compute node

```
CREATE COMPUTE MODULE Save_Request_MQMD
   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN

      -- Copy the original MQMD header
      CALL CopyMessageHeaders();

      -- Copy MsgId to CorrelId per MQ usual programming for request/reply 
pattern
      -- This allows the MQGet node in the Reply flow to retrieve the
      -- corresponding message stored in the MQMD_STOREQ by CorrelId.
      -- It also set the ReplyToQ to 'SOA_REPLY' for the Reply flow to know 
where to
      -- send the reply message.  DataPower is configured to pick up the 
reply 
message
      -- from the 'SOA_REPLY' queue.

      -- Note: We don't need to store the message body because the MQGET 
node only 
needs
      -- the correlation ID to get the store message.

      SET OutputRoot.MQMD.CorrelId = InputRoot.MQMD.MsgId;
      SET OutputRoot.MQMD.ReplyToQ = 'SOA_REPLY';

      RETURN TRUE;
   END;
```

3. Right-click **MQ_To_Legacy_MF_Proj** and then click **New** → **Message Flow**:
   a. Set Message flow name to `RestoreOriginalMQMD.msgflow` and then click **Finish**.
   b. Drag and drop the nodes from the Palette onto the Message Flow canvas and rename them as shown in Table B-9.

Table B-9 Node types

Node type	Palette folder name	Node name (rename)
Input	Construction	
MQGET	WebSphere MQ	Get_Request_MQMD
Compute Node 1	Transformation	Error:MQGET with warning
Compute node 2	WebSphere MQ	Error:MQGET no message
Output	Construction	

   c. Connect the nodes and terminals according to the table 4-6.

Table B-10 Connect these nodes and terminals

Node and terminal name	Node and terminal name
Input (Out)	Get_Request_MQMD (In)
Get_Request_MQMD (Out)	Output (In)
Get_Request_MQMD (Warning)	ErrorHandling:MQGET with Warning (In)
Get_Request_MQMD (No Message)	ErrorHandling:MQGET No Message (In)

d. Set the properties of the nodes in the RestoreOriginalMQMD.msgflow:

   Set the properties of the MQMD_STOREQ node:

   i. Right-click the node **MQMD_STOREQ** and click **Properties**.

   ii. Click the **Basic** tab and set Queue Name to **MQMD_STOREQ**.

e. Build the ESQL for the “ErrorHandling:MQGET with Warning” compute node as shown in Example B-6 on page 138.

Example: B-6 ESQL Code for MQGET with warning

```
CREATE COMPUTE MODULE RestoreOriginalReplyToQ_HandleMQGetWarning
   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
      -- The MQGet Node received a MQ Warning
      -- The Tree will contain just a MQMD and any message content
received
      -- in aa BLOB, no parsing will be performed.
      -- This could occur if the get options were set to accept truncated
      -- messages. We have not set that option here.
      -- Throw a user exception
      DECLARE reasonStr CHAR;
      SET reasonStr = 'MQGet returned warning';
      THROW USER EXCEPTION VALUES(reasonStr);
      RETURN TRUE;
   END;
```

f. Build the ESQL for the “ErrorHandling:MQGET with No Message” compute node as shown in Example B-7.

Example: B-7 ESQL Code for MQGET with no message

```
CREATE COMPUTE MODULE RestoreOriginalReplyToQ_HandleNoStoreMessage
   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
      -- The MQGet Node failed to get a message from the Store Queue
      -- Throw a user exception
      DECLARE reasonStr CHAR;
      SET reasonStr = 'MQGet no store queue message retrieved';
      THROW USER EXCEPTION VALUES(reasonStr);
      RETURN TRUE;
   END;

END
MODULE
```

4. Right-click **MQ_To_Legacy_MF_Proj** and then click **New** → **Message Flow**:

   a. Set the Message flow name to `Host_Request.msgflow` and then click **Finish**.

   b. Drag and drop the following nodes from the Palette onto the Message Flow canvas and rename them according to Table B-11.

Table B-11 List of node type, Palette folder name, and node name

Node type	Palette folder name	Node name
MQInput	WebSphere MQ	SOA_REQUEST
TryCatch	Construction	

Node type	Palette folder name	Node name
Compute Node	Transformation	XML_To_COBOL
Output Node	WebSphere MQ	HOST_REQUEST

c. Connect the nodes and terminals according to Table B-12 on page 139.

Table B-12 Instructions for wiring the nodes in HOST_Request.msgflow

Node and terminal name (from)	Node and terminal name (to)
SOA_REQUEST (out)	TryCatch (in)
TryCatch (catch)	Error_Handler (in)
TryCatch (try)	StoreOriginalMQMD_Sub (in)
StoreOriginalMQMD_Sub (out)	XML_To_COBOL (in)
XML_To_COBOL (out)	HOST_REQUEST (in)

d. Verify that your message flow looks like the one in Figure B-13.

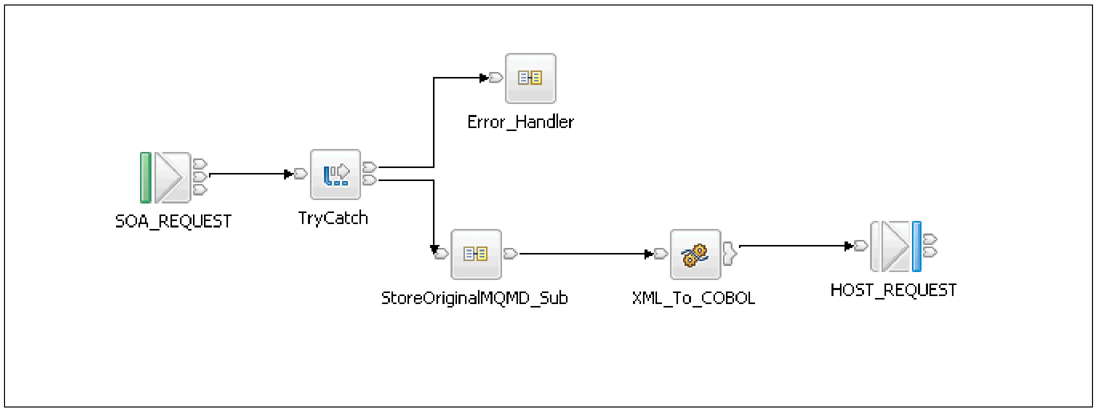


Figure B-13 Host_Request message flow

e. Set the properties of the nodes in Host_Request.msgflow:

Set the properties of the SOA_REQUEST node:

i. Right-click the **SOA_REQUEST** node and click **Properties**.

ii. Click the **Basic** tab and set Queue Name to **SOA_REQUEST**.

iii. Select the **Input Message Parsing** tab.

iv. Select **MRM** from the Message Domain drop-down list.

v. Select **SOA_Request_Reply_XML_MS** from the Message Set drop-down list.

vi. Select **Envelope** for the Message Type drop-down list.

vii. Select **XML1** for Message Format from the drop-down list.

viii.Click the **Advanced** tab and set Transaction Mode to **Automatic**.

f. Build the ESQL for the XML_To_COBOL compute node as shown in Example B-8 on page 140.

Example: B-8 ESQL code for XML_To_COBOL compute node

```
        CREATE FUNCTION Main() RETURNS BOOLEAN
           BEGIN
           -- Declare namespace after the BEGIN statement
              DECLARE tns NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
              DECLARE ns  NAMESPACE 'http://www.itso.lab.com';
            -- Copy Message Headers
              CALL CopyMessageHeaders();

             DECLARE CustIn_Ref REFERENCE TO
        InputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo;
              DECLARE CustOut_Ref REFERENCE TO OutputRoot.MRM;

           -- Transform the message from XML/SOAP to Cobol/fixed format before
        sending it to legacy
                SET OutputRoot.MRM.ACCOUNT_NUMBER =
        InputRoot.MRM.tns:Body.ns:IN_CustomerStatus.accountNumber;

           -- Output is legacy, so set wire format and message name
              SET OutputRoot.Properties.MessageSet =
        'Host_Request_Reply_Cobol_MS';
              SET OutputRoot.Properties.MessageType = 'msg_CUSTOMER';
              SET OutputRoot.Properties.MessageFormat = 'CWF1';
           -- Set ReplyToQ for the Host_Simulation message flow. This flow sends
        the reply to two different
           -- queues as follows:
           -- - HOST_REPLY if the request comes from the message flow
        WS_To_Legacy_MF (first scenario)
  -- - HOST_REPLY2 if the request comes from the message flow
        MQ_To_Legacy_MF (second scenario)
              SET OutputRoot.MQMD.CorrelID = InputRoot.MQMD.MsgId;
              SET OutputRoot.MQMD.ReplyToQ = 'HOST_REPLY2';

              RETURN TRUE;
           END;
```

5. Right-click **MQ_To_Legacy_MF_Proj** and then click **New** → **Message Flow**:
   a. Set Message flow name to `Host_Reply.msgflow` and then click **Finish**.
   b. Drag and drop the following nodes from the Palette onto the Message Flow canvas and rename them according to Table B-13.

   Table B-13 List of node type, Palette folder name, and node name

Node type	Palette folder name	Node name (rename)
MQInput	WebSphere MQ	HOST_REPLY2
TryCatch	Construction	
Compute Node	Transformation	XML_To_COBOL
Output Node	WebSphere MQ	HOST_REQUEST

   c. Connect the nodes and terminals according to Table B-14 on page 141.

Table B-14 Instructions for wiring nodes in Host_Reply message flow

Node and terminal name (from)	Node and terminal name (to)
SOA_REQUEST (out)	TryCatch (in)
TryCatch (catch)	Error_Handler (in)
TryCatch (try)	COBOL_To_XML (in)
COBOL_To_XML (out)	RestoreOriginalMQ_Sub (in)
RestoreOriginalMQ_Sub (out)	ReplyToQ

d. Verify that your message flow looks like the one shown in Figure B-14.

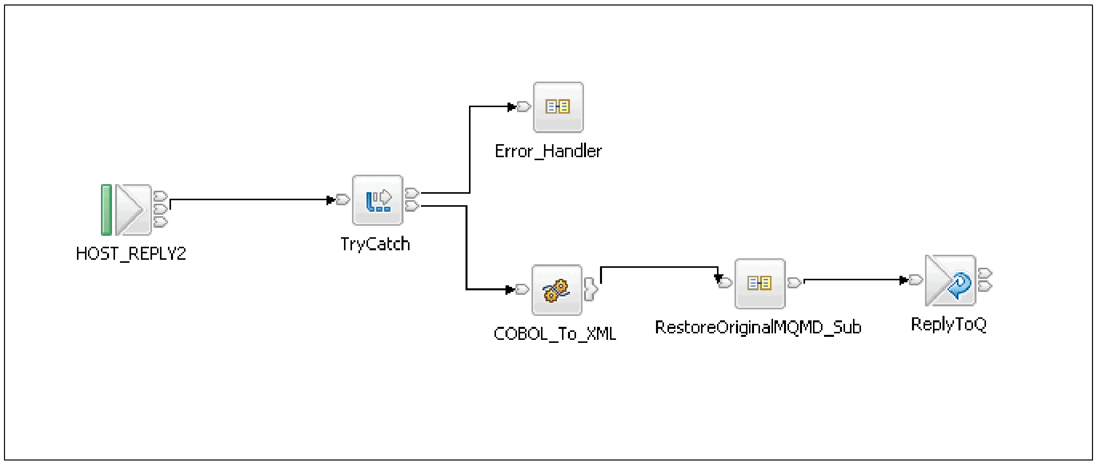


Figure B-14 Host_Reply message flow

e. Set the properties of the nodes in Host_Reply.msgflow:

Set the properties of the HOST_REPLY2 node:

i. Right-click the **HOST_REPLY2** node and click **Properties**.

ii. Click the **Basic** tab and set Queue Name to `HOST_REPLY2`.

iii. Select the **Input Message Parsing** tab.

iv. Select **MRM** from the Message Domain drop-down list.

v. Select **Host_Request_Reply_Cobol_MS** from the Message Set drop-down list.

vi. Select **msg_CUSTOMER** for the Message Type drop-down list.

vii. Select **CWF1** for the Message Format from the drop-down list.

viii.Click the **Advanced** tab and set Transaction Mode to **Automatic**.

f. Build the ESQL for the COBOL_To_XML compute node as shown in Example B-9 on page 142.

Example: B-9   ESQL for COBOL_To_XML compute node

```
CREATE COMPUTE MODULE COBOL_To_XML
   CREATE FUNCTION Main() RETURNS BOOLEAN
      BEGIN
   -- Declare namespace after the BEGIN statement
   DECLARE tns NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
   DECLARE ns  NAMESPACE 'http://www.itso.lab.com';
-- Input is CWF, output is SOAP/XML
    CALL CopyMessageHeaders();
   SET OutputRoot.Properties = InputRoot.Properties;
   SET OutputRoot.Properties.MessageSet = 'SOA_Request_Reply_XML_MS';
   SET OutputRoot.Properties.MessageType = 'Envelope';
   SET OutputRoot.Properties.MessageFormat = 'XML1';
   SET OutputRoot.MQMD.ReplyToQ = 'SOA_REPLY';
   SET OutputRoot.MQMD.ReplyToQMgr = 'WBRK6_DEFAULT_QUEUE_MANAGER';
   DECLARE CustOut_Ref REFERENCE TO
OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus;
   DECLARE CustIn_Ref REFERENCE TO InputRoot.MRM.CUSTOMER_INFO;

   SET OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.accountNumber =
InputRoot.MRM.ACCOUNT_NUMBER;
      SET
OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.status =
InputRoot.MRM.CUSTOMER_INFO.CUSTOMER_STATUS;
      SET
OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.firstName =
InputRoot.MRM.CUSTOMER_INFO.FIRST_NAME;
      SET
OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.lastName =
InputRoot.MRM.CUSTOMER_INFO.LAST_NAME;
      SET
OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.street =
InputRoot.MRM.CUSTOMER_INFO.STREET;
      SET OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.city
=
InputRoot.MRM.CUSTOMER_INFO.CITY;
      SET OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.state
=
InputRoot.MRM.CUSTOMER_INFO.STATE;
      SET
OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.country =
InputRoot.MRM.CUSTOMER_INFO.COUNTRY ;
      SET
OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.zipCode =
InputRoot.MRM.CUSTOMER_INFO.ZIPCODE;
      SET OutputRoot.MRM.tns:Body.ns:IN_CustomerStatus.customerInfo.phone
=
InputRoot.MRM.CUSTOMER_INFO.PHONE;

      END;
```

Deploying the message sets and message flows

We have completed building the message flows for testing both scenarios. We must compile and deploy them to a message broker where they will be instantiated and executed. You will learn how to deploy the message sets and message flows next:

1. Create a Broker Archive File (bar) to include all of the message sets and flows for both scenarios. The bar file contains the compile code to be executed in a broker:

   a. Click **Broker Administration Navigator** perspective (Figure B-15).

   b. Click **Server** → **new** → **Message Broker Archive**.

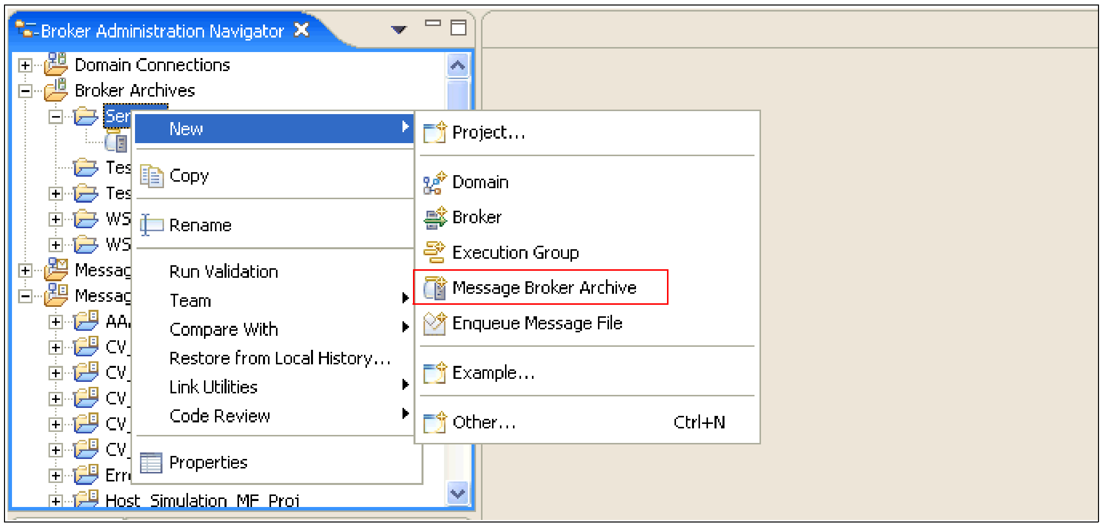
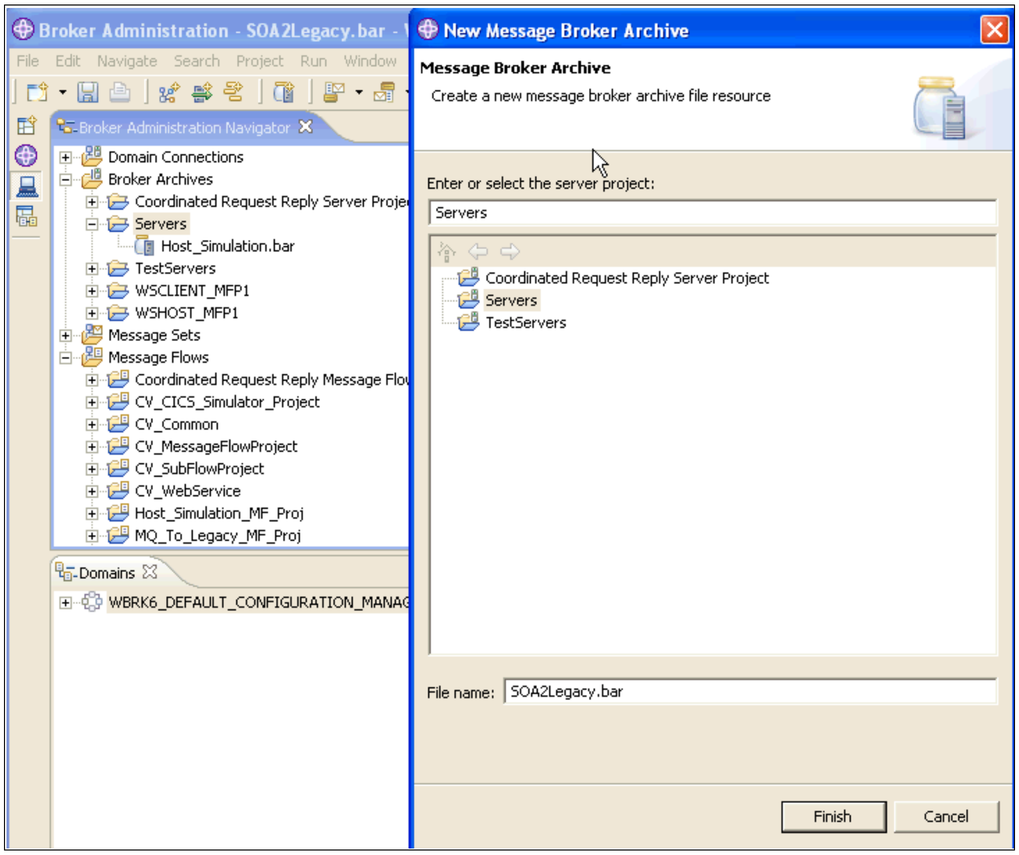


Figure B-15 Create a new broker archive file

   c. Set the name of the bar file to `SOA2Legacy.bar` as shown in Figure B-16 on page 144.


Broker Administration - SOA2Legacy.bar
File Edit Navigate Search Project Run Window
Broker Administration Navigator
Domain Connections
Broker Archives
Coordinated Request Reply Server Proje
Servers
Host_Simulation.bar
TestServers
WSCLIENT_MFP1
WSHOST_MFP1
Message Sets
Message Flows
Coordinated Request Reply Message Flo
CV_CICS_Simulator_Project
CV_Common
CV_MessageFlowProject
CV_SubFlowProject
CV_WebService
Host_Simulation_MF_Proj
MQ_To_Legacy_MF_Proj
Domains
WBRK6_DEFAULT_CONFIGURATION_MANAG
New Message Broker Archive
Coordinated Request Reply Server Project
Servers
TestServers
Finish
Cancel

Message Broker Archive

Create a new message broker archive file resource

Enter or select the server project: Servers

File name: SOA2Legacy.bar

Figure B-16 Create SOA2Legacy bar file

d. Another editor is open to allow you to **add** the message sets and message flows to the bar file as shown in Figure B-17 on page 145. Click on the plus sign (**+**) to add the following items:

i. Host_Request_Reply_COBOL_MS

ii. SOA_Request_Reply_XML_MS

iii. Host_Simulation_MF

iv. WS_To_Legacy_MF

v. MQ_To_Legacy_MF

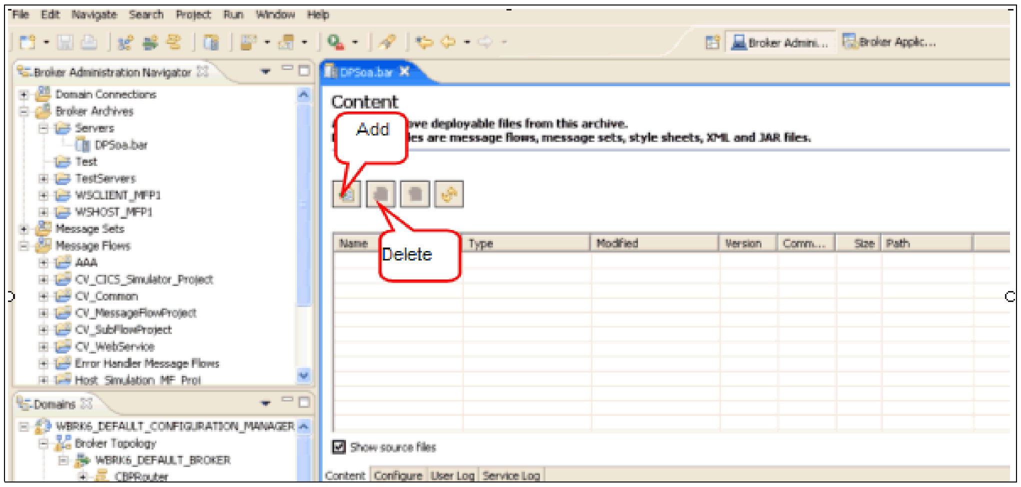
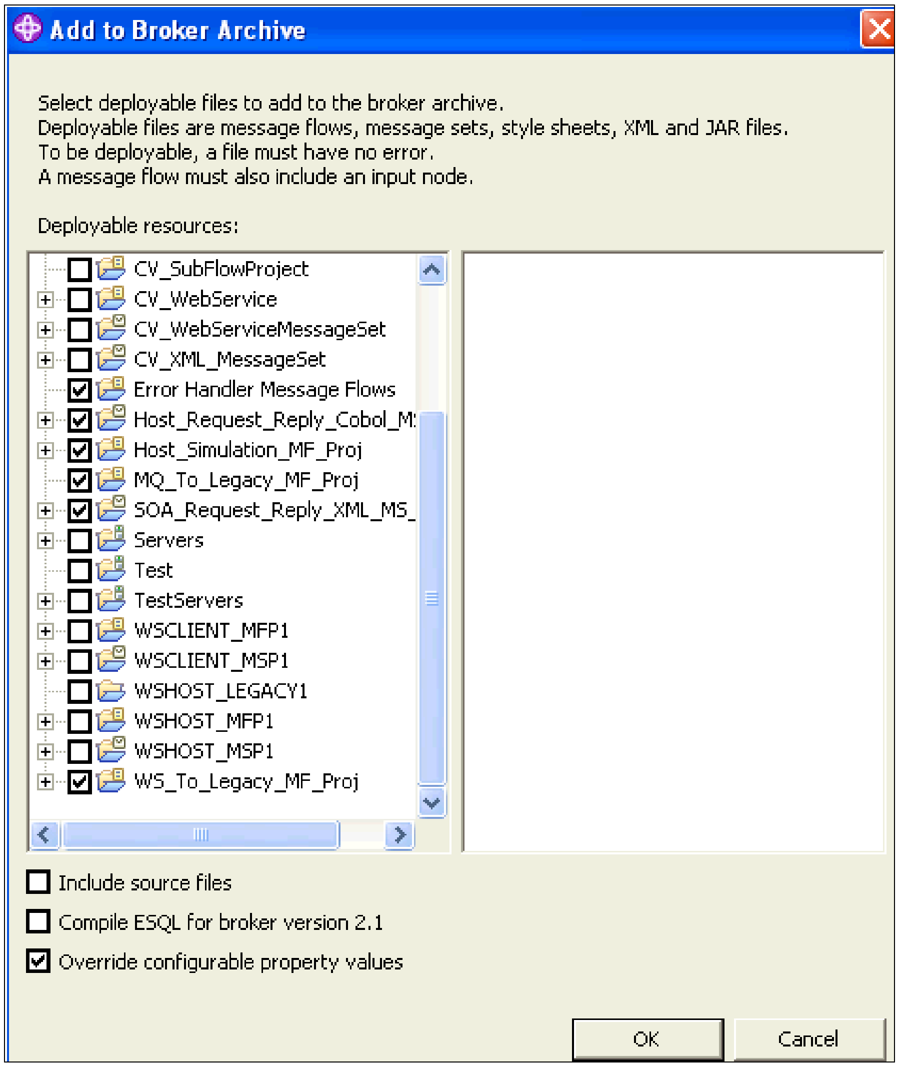


Figure B-17 Broker archive editor

e. Click **OK**. See Figure B-18.



Select deployable files to add to the broker archive.
Deployable files are message flows, message sets, style sheets, XML and JAR files.
To be deployable, a file must have no error.
A message flow must also include an input node.

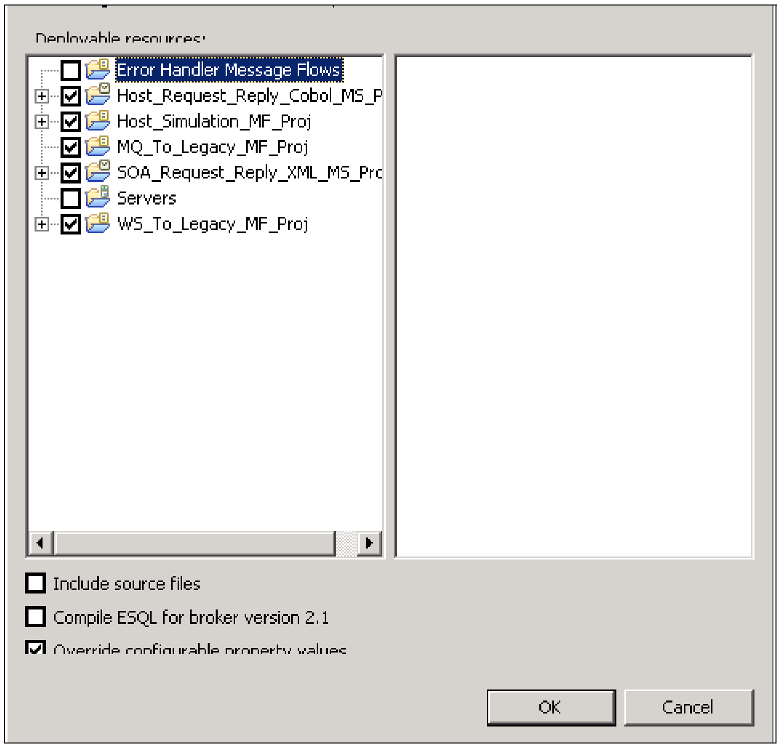
Deployable resources:

- [ ] CV_SubFlowProject
- [ ] CV_WebService
- [ ] CV_WebServiceMessageSet
- [ ] CV_XML_MessageSet
- [x] Error Handler Message Flows
- [x] Host_Request_Reply_Cobol_M:
- [x] Host_Simulation_MF_Proj
- [x] MQ_To_Legacy_MF_Proj
- [x] SOA_Request_Reply_XML_MS_
- [ ] Servers
- [ ] Test
- [ ] TestServers
- [ ] WSCLIENT_MFP1
- [ ] WSCLIENT_MSP1
- [ ] WSHOST_LEGACY1
- [ ] WSHOST_MFP1
- [ ] WSHOST_MSP1
- [x] WS_To_Legacy_MF_Proj

- [ ] Include source files
- [ ] Compile ESQL for broker version 2.1
- [x] Override configurable property values

Figure B-18 Build SOA2Legacy.bar

f. Click **OK**. See Figure B-19.



Deployable resources:

- [ ] Error Handler Message Flows
- [✓] Host_Request_Reply_Cobol_MS_P
- [✓] Host_Simulation_MF_Proj
- [✓] MQ_To_Legacy_MF_Proj
- [✓] SOA_Request_Reply_XML_MS_Prc
- [ ] Servers
- [✓] WS_To_Legacy_MF_Proj

- [ ] Include source files
- [ ] Compile ESQL for broker version 2.1
- [✓] Override configurable property values

Figure B-19 Soa2Legacy.bar

g. Press Ctrl + S to save the bar file.

h. Next, you must create a new Execution Group called DPSoa and deploy the Soa2Legacy.bar to DPSoa. Refer to “Deploying the message sets and message flows” on page 143 starting at step 2 to deploy the Soa2Legacy.bar.

C

Additional material

This paper refers to additional material that can be downloaded from the Internet as described.

Locating the Web material

The Web material associated with this paper is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

`ftp://www.redbooks.ibm.com/redbooks/REDP4365`

Alternatively, you can go to the IBM Redbooks Web site at:

`http://www.redbooks.ibm.com`

Select the **Additional materials** and open the directory that corresponds with the IBM Redpaper form number, REDP4365.

Using the Web material

The additional Web material that accompanies this paper includes the following files:

File name	*Description*
redp4365.zip	Zipped Code Samples

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	40 MB minimum
Operating System:	Windows
Processor:	1 GH or higher
Memory:	1 GB or higher

How to use the Web material

Create a subdirectory (folder) on your workstation and unzip the contents of the Web material zipped file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 150. Note that some of the documents referenced here might be available in softcopy only:

- *Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB*, SG24-7369
- *WebSphere Message Broker Basics*, SG24-7137
- *Enabling SOA Using WebSphere Messaging,* SG24-7163
- *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started,* REDP-4327
- *IBM WebSphere DataPower SOA Appliances Part II: Authentication and Authorization,* REDP-4364
- *IBM WebSphere DataPower SOA Appliances Part IV: Management and Governance,* REDP-4366

Other publications

These publications are also relevant as further information sources:

- *WebSphere Business Integration Message Broker V5 Overview and Architecture,* REDP-3867

Online resources

These Web sites are also relevant as further information sources:

- Integrating WebSphere DataPower SOA Appliances with WebSphere MQ

  http://www.ibm.com/developerworks/websphere/library/techarticles/0703_crocker/0703_crocker.html
- Integrating WebSphere DataPower XML Security Gateway XS40 with WebSphere Message Broker

  http://www.ibm.com/developerworks/websphere/library/techarticles/0710_crocker/0710_crocker.html
- Integrating DataPower with WebSphere Message Broker using the Broker Explorer

  http://www.ibm.com/developerworks/websphere/library/techarticles/0707_storey/0707_storey.html

How to get IBM Redbooks publications

You can search for, view, or download IBM Redbooks publications, Redpapers, Technotes, draft publications, and Additional materials, as well as order hardcopy IBM Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM WebSphere DataPower SOA Appliances
Part III: XML Security Guide

Redpaper™

Secure and improve your XML and Web Services setup and deployment

Integrate DataPower appliances with WebSphere Message Broker

Provide XML threat protection in DataPower

IBM® WebSphere® DataPower® SOA Appliances represent an important element in the holistic approach of IBM to service-oriented architecture (SOA). IBM SOA appliances are purpose-built, easy-to-deploy network devices that simplify, help secure, and accelerate your XML and Web services deployments while extending your SOA infrastructure. These appliances offer an innovative, pragmatic approach to harness the power of SOA. By using them, you can simultaneously use the value of your existing application, security, and networking infrastructure investments.

This series of IBM Redpaper publications is written for architects and administrators who need to understand the implemented architecture in WebSphere DataPower appliances to successfully deploy it as a secure and efficient enterprise service bus (ESB) product. Part three of the series, this paper, describes how to use the DataPower appliance to secure incoming Web Services within an SOA environment. It also describes how to integrate your DataPower appliance with WebSphere Message Broker. This paper also explains how to provide protection against security attacks by implementing the XML Denial of Service (XDoS) provided by DataPower appliances. The entire IBM WebSphere DataPower SOA Appliances series includes the following papers:

- *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started*, REDP-4327
- *IBM WebSphere DataPower SOA Appliances Part II: Authentication and Authorization*, REDP-4364
- *IBM WebSphere DataPower SOA Appliances Part III: XML Security Guide*, REDP-4365
- *IBM WebSphere DataPower SOA Appliances Part IV: Management and Governance*, REDP-4366

REDP-4365-00