WebSphere. software

IBM

IBM WebSphere DataPower SOA Appliances

Part I: Overview and Getting Started

- Understand and effectively deploy DataPower SOA appliances
- Parse and transform binary, flat text, and XML messages
- Learn how to extend your SOA infrastructure

Juan R. Rodriguez
Somesh Adiraju
Joel Gauci
Markus Grohmann
Davin Holmes
Tamika Moody
Srinivasan Muralidharan
Christian Ramirez
Adolfo Rodriguez

Redpaper

ibm.com/redbooks

International Technical Support Organization

IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started

April 2008

REDP-4327-00

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

First Edition (April 2008)

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

This document created or updated on March 27, 2008.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CICS®
DataPower device®
DataPower®
DB2®
developerWorks®
IBM®
IMS™
Lotus Notes®
Lotus®
Notes®
Rational®
Redbooks®
Redbooks (logo)  ®
Tivoli®
WebSphere®

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

Java, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Expression, Internet Explorer, Microsoft, SQL Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

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, 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 1 of the series, this part, provides a general overview of DataPower SOA appliances and a primer to using the appliances in common scenarios. 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 (ITSO), Raleigh Center.

Juan R. Rodriguez is a Consulting IT professional and Project Leader at the IBM ITSO Center, Raleigh. He has an Master of Science (MS) 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, Juan worked at the IBM laboratory in Research Triangle Park, North Carolina, as a designer and developer of networking products.

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

Joel Gauci has been working for IBM Global Business Services in France since 2001, as an Advisory IT Specialist. He mainly works in design and implementation on portal and Web services platforms. He has worked on SOA and WebSphere DataPower projects since May 2006, for leading firms in the mobile telephony area. Joel holds a master degree in computer science and engineering from l'Ecole d'Ingénieurs du Pas-de-Calais School (EIPC), France.

Markus Grohmann is an IT Specialist working as an IBM Business Partner in Austria. He has five years of experience with a broad range of IBM products and their implementation in customer environments. Markus graduated from Salzburg University of Applied Sciences and Technologies in 2002.

Davin Holmes is a Staff Software Engineer for IBM Software Group, Tivoli. He has worked in software development for seven years in a variety of technical areas, which include smartcards, enterprise software integration, and Web services, with a particular focus on information security. Davin is the team lead for the DataPower and Tivoli Security integration effort located at the Gold Coast, Australia site of the Australia Development Laboratory (ADL). He has degrees in electrical and computer engineering and optoelectronics from Queensland University of Technology and Macquarie University.

Tamika Moody is a WebSphere Business Integration Message Broker/WebSphere DataPower Consultant and IT Specialist for IBM. She has over seven years of experience in the IT integration area. Tamika has broad experience in leading middleware engagements ranging from electronic data interchange (EDI) and business-to-business (B2B) implementations to design, implementation, and problem determination of DataPower and IBM middleware solutions.

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. Srinivasan has also 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.

Christian Ramirez is an IBM Software Solutions Architect working for GBM Corporation, an IBM Alliance Company, located in San José, Costa Rica. He has ten years of experience with IBM products and five years experience as an Integration Solution Architect. He has worked with Lotus® Notes®, WebSphere MQ, and WebSphere BI Message Broker. In addition, he has been part a WebSphere Pre-Sales team and has implemented several integration projects.

Adolfo Rodriguez is a Software Architect within the IBM WebSphere Technology Institute (WSTI). He leads a team of engineers who focus on emerging technologies in WebSphere products and, more recently, DataPower SOA appliances. His 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. Adolfo is also an Assistant Adjunct Professor of Computer Science at Duke University, where he teaches networking courses. He has written 12 books and numerous research articles. He holds four degrees from Duke University: a Bachelor of Science in Computer Science, a Bachelor of Arts in Mathematics, MS in Computer Science, and a Ph.D. in Computer Science (Systems).

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

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

Andy Grohman
IBM Charlotte, North Carolina, USA

Bill Hines, Senior Certified Consulting I/T Specialist, IBM Software Services for WebSphere, for use of his article “The (XML) threat is out there...”

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® in one of the following ways:

- Use the online **Contact us** review Redbooks 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

Introduction to DataPower SOA appliances

The emergence and proliferation of the Extensible Markup Language (XML) and Web services has seen an explosion in the middleware infrastructure required to support them. An important component in this middleware architecture is in the enterprise service bus (ESB), a collection of runtime components that provides service intermediation such as routing, transformation/bridging, management, security, and other control functions.

While XML and SOAP enable a rich application-aware communication lingua franca, their emergence in this space has been riddled by three key challenges. First, the size and complexity of traditional software-based middleware product installations has increased installation and maintenance costs of service-oriented architecture (SOA) deployments and decreased overall solution consumability. Second, the text-based, application-centric parsing and processing demands placed on middleware infrastructures have negatively affected overall system performance. While most middleware solutions “scale out”, the overall effect is increased license and operational costs as more instances of individual middleware components must be deployed to meet service consumer demand. Third, the new genre of application-awareness has likewise led to a new genre of security attacks and exposures that use the architecture of middleware components.

DataPower SOA appliances address these three challenges with the creation of specialized, purpose-built, consumable SOA appliances that redefine the boundaries of middleware. As the “hardware ESB,” DataPower SOA appliances are an increasingly important part of the IBM ESB family.

In this chapter, we introduce you to the DataPower appliance and present the most common scenarios and use cases. In addition, we discuss configuration and usage of the DataPower appliance as well as SOA governance.

1.1 Overview of the DataPower appliance

With SOAs, composite applications are created that are comprised of reusable, loosely-coupled service components. The technical foundation of an SOA is in the support for the XML and Web services built on top of it. By using SOAP, SOA clients can invoke services (RPC-style) without explicit support for a wide variety of transport protocols and message formats. By having a SOAP facade in front of an existing service, virtualization can occur where clients invoke a virtualized version of the underlying service. It eliminates the need to understand intricate details of the service's implementation.

In this context, the use of XML enables data to be self-describing with explicit language support for common operations that manipulate the data. For example, by using the XPath language, you have a consistent way to select data items from within an XML document. In SOA, service intermediaries can use XML and other application-layer data to route, secure, control, transform, or otherwise process service requests and responses, decoupled from the actual service implementation that fulfills each particular request.

1.1.1 Challenges in service-oriented networking

Although greatly appealing, the promise of loosely-coupled, virtualized services in SOA comes at a price. Because the data-centric complexity of XML and SOA operations has increased, traditional software-based middleware has struggled to keep up. In particular, software-based service intermediaries have emerged as natural extensions to traditional server-side service stack environments. Unfortunately, their success and impact have been inhibited by three fundamental challenges of consumability, security, and performance. DataPower SOA appliances overcome these challenges.

Consumability

Often, middleware-service stacks have an underlying software engine (generally J2EE in origin) upon which a Web service hosting engine is built. In some sense, this group of products has been built by joining together necessary components in an embedded fashion.

For example, a J2EE servlet engine can be extended to receive SOAP over HTTP by providing a new Web service servlet. The Web service itself is deployed on this servlet. The result is a system built from multiple software layers, each with its own configuration and maintenance requirements. Taken individually, each layer's requirements may prove tedious. Taken together, the collective set of installation and maintenance requirements often proves prohibitive. For example, patch upgrades that affect a layer in the stack of embedded software must be coordinated in a single atomic action. Further complicating this problem is the focus that the traditional software industry has. The industry tends to favor the addition of more functions to a software product over increasing the usability of the existing function.

Security

The advent of SOA has created a common communication framework to understand and operate on application data that has never been seen before. With self-describing XML, intermediaries can extract portions of the data stream and affect application-aware policies.

Unfortunately, this has also enabled a new opportunity for malicious attacks. That is, as XML regularly flows from client to enterprise through IP firewalls without much impediment, the obvious place to attack is in the application data stream itself, the XML. While we are just beginning to understand the repercussions of these types of attacks, they are emerging. XML denial-of-service (XDoS) attacks seek to inject malformed or malicious XML into middleware servers with the goal of causing the server to churn away valuable cycles and processing the

malicious XML. Enterprise-ready application servers are susceptible to many of these types of attacks, leaving a security hole open that must be closed.

Performance

Another key challenge that has emerged with the adoption of XML is in the computational cost of XML processing. Computing on XML in traditional software-based middleware is orders of magnitude more costly (from the computational sense) than native data structures. XML must be parsed into the native data structures of the local computer's architecture.

Further, XML transformations exacerbate processing needs because they require multiple passes through the XML structure and are highly sensitive to the transformation processing engine. Securing XML and SOA at the application (XML) level provides barriers that can requires as much as 60 times the processing capability as plain XML, based on typical workloads.

Additionally, it is often prohibitive from a performance point of view to enable key requirements such as monitoring, auditing, and security. Customers end up sacrificing those functions to keep equipment costs from growing unwieldly.

1.1.2 Meeting SOA challenges with DataPower appliances

The IBM WebSphere DataPower SOA Appliances family contains rack-mountable network devices that overcome many of the challenges that face SOA and XML today. At a high-level, the IBM WebSphere DataPower SOA Appliances offer the following features:

- 1U (1.75-inch thick) rack-mountable, purpose-built network appliances
- XML/SOAP firewall, field-level XML security, data validation, XML Web services access control, and service virtualization
- Lightweight and protocol-independent message brokering, integrated message-level security and fine-grained access control, and the ability to bridge important transaction networks to SOAs and ESBs
- High performance, multi-step, wire-speed message processing, including XML, XML Stylesheet Language Transformation (XSLT), XPath, and XML Schema Definition (XSD)
- Centralized Web services policy and service-level management
- Web services (WS) standard support, such as WS-Security, Security Assertion Markup Language (SAML) 1.0/1.1/2.0, portions of the Liberty Alliance protocol, WS-Federation, WS-Trust, XML Key Management Specification (XKMS), Radius, XML Digital Signature, XML-Encryption, Web Services Distributed Management (WSDM), WS-SecureConversation, WS-Policy, WS-SecurityPolicy, WS-ReliableMessaging, SOAP, Web Services Description Language (WSDL), Universal Description, Discovery, and Integration (UDDI)
- Transport layer flexibility, which supports HTTP/HTTPS, MQ, Secure Sockets Layer (SSL), File Transfer Protocol (FTP), and others
- Scalable, wire-speed, any-to-any message transformation, such as arbitrary binary, flat text and XML messages, which include COBOL copybook, CORBA, CICS, ISO 8583, ASN.1, EDI, and others

DataPower appliances can meet the challenges that are present in an SOA network with the following features:

- Consumable simplicity

  An easy-to-install and easy-to-maintain network appliance that can satisfy both application and network operational groups, supporting current and emerging standards, as well as readily available XML Web services standards.

- Enhanced security

  Key support that includes XML/SOAP firewall and threat protection, field-level XML security, data validation, XML Web services access control, service virtualization, and SSL acceleration.

- Acceleration

  A drop-in solution that can streamline XML and Web service deployments, helping to lower the total cost of ownership and accelerate a return on your assets, as you continue to move to SOA. SOA appliances are purpose-built hardware devices that are capable of offloading overtaxed servers by processing XML, Web services, and other message formats at wire speed.

1.1.3 DataPower appliance models

In this section, we explain each of the three WebSphere DataPower appliance models. Figure 1-1 illustrates the role of these models.

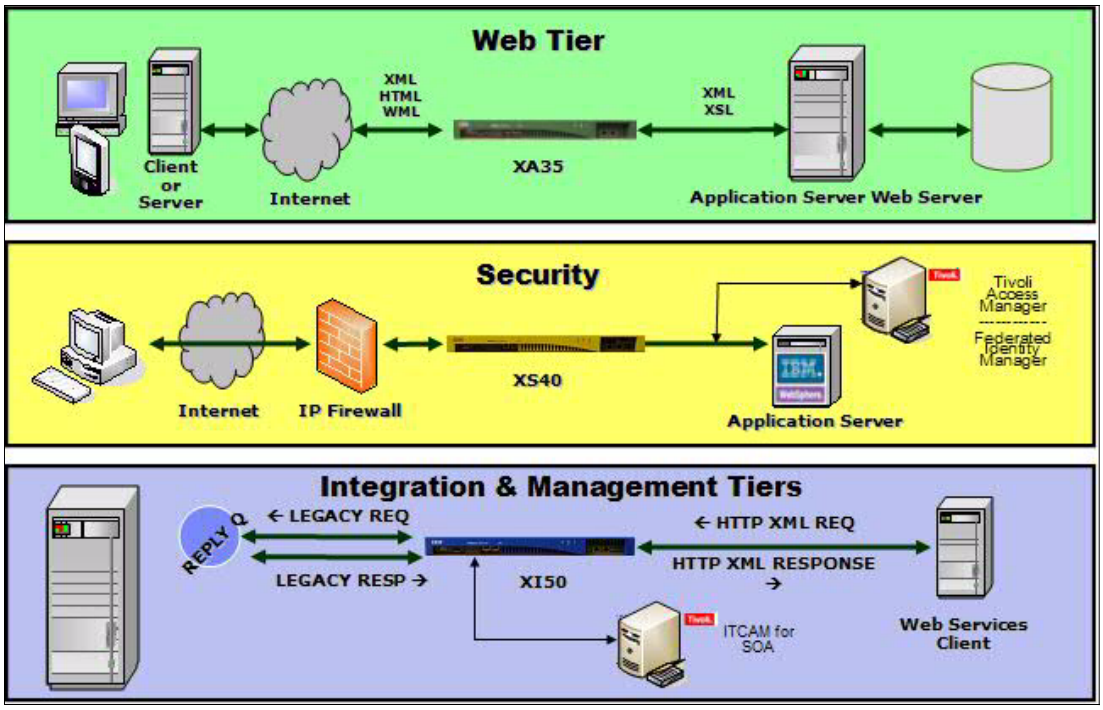


Figure 1-1 DataPower SOA Appliance models

IBM WebSphere DataPower XML Security Gateway XS40

The DataPower XML Security Gateway XS40 appliance provides a security-enforcement point for XML and Web service transactions. It offers encryption, firewall filtering, digital signatures, schema validation, WS-Security, XML access control, XPath, and detailed.

In addition, the XS40 model offers the following features:

- XML/SOAP firewall

  The DataPower XML Security Gateway XS40 filters traffic at wire speed, based on information from layers two through seven of the protocol stack. It filters traffic from field-level message content and SOAP envelopes to IP address, port or host name, payload size, and other metadata. Filters can be predefined with an easy point-and-click XPath filtering GUI and automatically uploaded to change security policies based on the time of day or other triggers.

- XML/SOAP data validation

  With its unique ability to perform XML schema validation as well as message validation, at wire speed, the XS40 model ensures that incoming and outgoing XML documents are legitimate and properly structured. It protects against threats such as XDoS attacks, buffer overflows, or vulnerabilities created by deliberately or inadvertently malformed XML documents.

- Field-level message security

  The XS40 model selectively shares information through encryption or decryption and signing or verification of entire messages or individual XML fields. These granular and conditional security policies can be based on nearly any variable, including content, IP address, host name, or other user-defined filters.

- XML Web services access control

  The XS40 model supports a variety of access control mechanisms, including WS-Security, WS-Trust, X.509, SAML, SSL, Lightweight Directory Access Protocol (LDAP), RADIUS, and simple client/URL maps. The XS40 model can control access rights by rejecting unsigned messages and verifying signatures within SAML assertions.

- Service virtualization

  XML Web services require companies to link partners to resources without leaking information about their location or configuration. With the combined power of URL rewriting, high-performance XSL transforms and XML/SOAP routing, the XS40 model can transparently map a rich set of services to protected back-end resources with high performance.

- Centralized policy management

  With the wire-speed performance of the XS40 model, enterprises can centralize security functions in a single drop-in device that can enhance security and help reduce ongoing maintenance costs. Simple firewall functionality can be configured via a GUI and be running in minutes. By using the power of XSLT, the XS40 model can also create sophisticated security and routing rules. Because the XS40 model works with leading policy managers, it is an ideal policy execution engine for securing next generation applications. Manageable locally or remotely, the XS40 model supports Simple Network Management Protocol (SNMP), script-based configuration, and remote logging to integrate seamlessly with leading management software. Its emerging support for WS-Policy and WS-SecurityPolicy further augments these capabilities.

- Web services management/service level management

  The XS40 model has support for WSDM, UDDI, WSDL, Dynamic Discovery, and broad support for service-level management (SLM) configurations. With this support, it natively offers a robust Web services management framework for the efficient management of distributed Web service endpoints and proxies in heterogeneous SOA environments. SLM alerts and logging, as well as pull and enforce policies, help enable broad integration support for third-party management systems and unified dashboards, in addition to robust support and enforcement for governance frameworks and policies.

IBM WebSphere DataPower Integration Appliance XI50

The DataPower Integration Appliance XI50 model provides transport-independent transformations between binary, flat text files and XML message formats. Visual tools are used to describe data formats, create mappings between different formats, and define message choreography. The XI50 appliance can transform binary, flat text, and other non-XML messages to help offer an innovative solution for security-rich XML enablement, ESBs, and mainframe connectivity.

In addition, the XI50 model offers the following features:

- Any-to-any transformation engine

  The XI50 model can parse and transform arbitrary binary, flat text, and XML messages, including EDI, COBOL copybook, ISO 8583, CSV, ASN.1, and ebXML. Unlike approaches based on custom programming, the patented DataGlue technology of the DataPower appliance uses a fully declarative, metadata-based approach.

- Transport bridging

  With support for a wide array of transport protocols, the XI50 is capable of bridging request and response flows to and from protocols such as HTTP, HTTPS, MQ, SSL, IMS Connect, FTP, and more.

- Integrated message-level security

  The XI50 model includes mature message-level security and access control functionality. Messages can be filtered, validated, encrypted, and signed, helping to provide more secure enablement of high-value applications. Supported technologies include WS-Security, WS-Trust, SAML, and LDAP.

- Lightweight message brokering
  - Sophisticated multi-step message routing, filtering, and processing
  - Multiple synchronous and asynchronous transport protocols
  - Detailed logging and audit trail, including non-repudiation support

IBM WebSphere DataPower XML Accelerator XA35

The DataPower XML Accelerator XA35 model can help speed common types of XML processing by offloading this processing from servers and networks. It can perform XML parsing, XML schema validation, XPath routing, XSLT, XML compression, and other essential XML processing with wire-speed XML performance. The XA35 offers the following benefits:

- Unmatched performance

  The purpose-built message processing engine of the DataPower appliance can deliver wire-speed performance for both XML-to-XML and XML-to-HTML transformations with increased throughput and decreased latency.

- Ease of use

  The self-learning XA35 model provides drop-in acceleration with virtually no changes to the network or application software. No proprietary schemas, coding or APIs are required to install or manage the device. In addition, it supports popular XML integrated development environments (IDEs) to help reduce the number of hours spent in the development and debugging of XML applications.

- Helps reduce infrastructure costs

  Unlike simple content switches that only redirect business documents, the DataPower XML Accelerator XA35 model fully parses, processes, and transforms XML with wire-speed performance and scalability to help reduce the need for stacks of servers. The XA35 model also supports accelerated SSL processing to help further reduce the load on server software.

- Helps cut development costs

  With the XA35 model, multiple applications can use a single, uniformed XML processing layer for all XML processing needs. By standardizing on high-performance hardware appliances, enterprises can deploy sophisticated applications while helping to eliminate unnecessary hours of application debugging and tuning for marginal performance gains.

- Intelligent XML processing

  In addition to wire-speed processing, DataPower appliances support XML routing, XML pipeline processing, XML compression, XML/XSL caching, as well as other intelligent processing capabilities to help manage XML traffic.

- Advanced management

  The DataPower XML Accelerator XA35 model provides real-time visibility into critical XML statistics such as throughput, transaction counts, errors, and other processing statistics. Data network-level analysis is provided and includes server health information, traffic statistics, and management and configuration data.

For full product information about IBM WebSphere DataPower SOA Appliances, refer to the following Web page:

http://www-306.ibm.com/software/integration/datapower/index.html

1.2 DataPower deployment scenarios and use cases

DataPower SOA appliances provide a robust, secure platform for middleware integration that can be deployed in an array of deployment scenarios to perform a variety of middleware use cases. In this section, we highlight the most common scenarios and use cases.

1.2.1 DataPower deployment scenarios

Figure 1-2 on page 8 illustrates common scenarios for deploying DataPower SOA appliances on the intranet, the demilitarized zone (DMZ), and a federated extranet, such as for a business partner. Of particular importance is the capability of the DataPower appliance to pass the most stringent requirements for enterprise DMZ deployment. The DataPower architecture is a secure environment with absolutely no Java™ on the appliance. Network ports are secured by default with no remote access beyond its command line interface (CLI) over the secure SSH protocol, WebGUI over HTTPS, and XML management APIs over HTTPS.

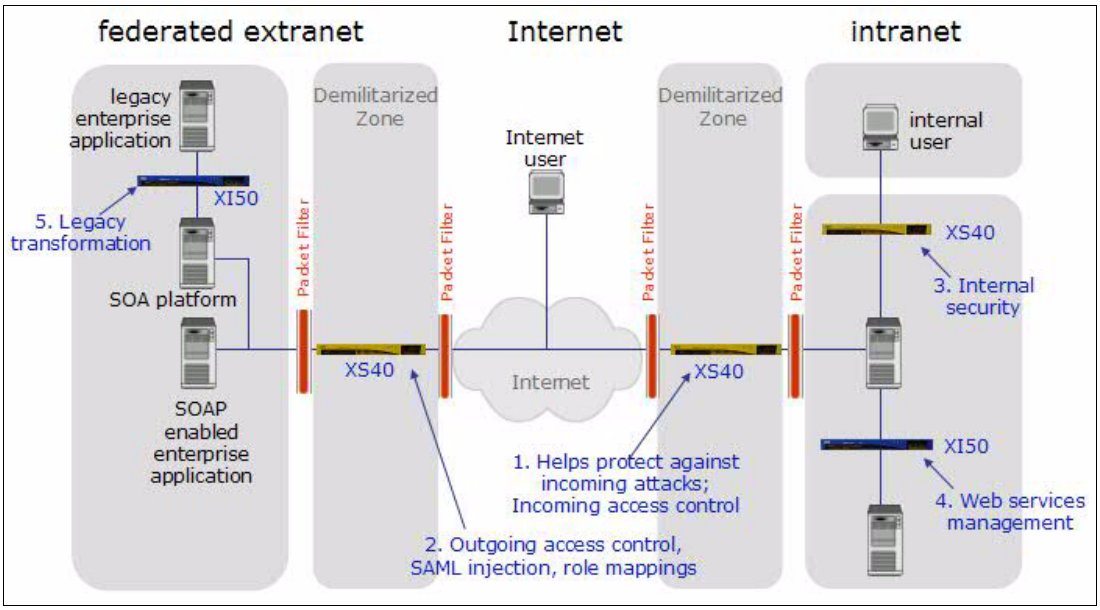


Figure 1-2 DataPower deployment scenarios

1.2.2 DataPower use cases

The DataPower appliance fulfills key roles within an SOA environment. These roles, or use cases, are discussed in this section.

Monitoring and management

The architecture and functional features of the DataPower appliance make it an attractive platform for centralized management and control. With the ability of the DataPower appliance to perform deep-content introspection at wire speed enables, the device can monitor service requests and responses without negatively affecting overall performance of their service invocation. DataPower appliances offer a number of different mechanisms for monitoring traffic through the device from low-level service statistics to more elaborate SLM:

- Statistics

  The DataPower appliance provides real-time visibility into critical statistics such as throughput, transaction counts, errors, message sizes, and other processing statistics. Data network-level analysis is provided and includes server health information, traffic statistics, and management and configuration data.

- Remote management

  The devices are capable of generating SNMP events. Script-based configuration is provided. Remote logging support integrates seamlessly with leading management software.

- Web services management

  The DataPower appliance supports WSDM, UDDI, WSDL, and Dynamic Discovery.

- Service-level management

  The DataPower appliance allows the specification of quality of service (QoS) policies that shape or throttle traffic based on service-level criteria. This enables the prioritization of service requests in support of business goals.

- Integration with various monitoring products such as IBM Tivoli Enterprise Monitoring and Netegrity SiteMinder

  Figure 1-3 illustrates how DataPower appliances can forward service-level information to the IBM Tivoli Composite Application Management for SOA, which in turn presents graphical views of service performance in the SOA.

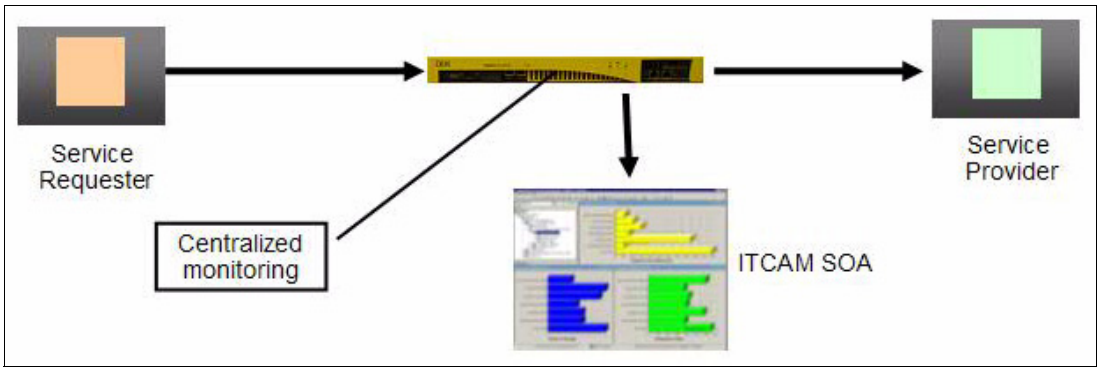


Figure 1-3 Centralized monitoring and management with the DataPower appliance

XML threat protection and security

Traditional firewalls only protect traffic at the IP level. Web services effectively tunnel through the IP firewall layer via standard HTTP or HTTPS and expose the organization's applications to completely new threats. We must ensure that only valid requests for valid services from genuine clients penetrate the enterprise boundary. That is, an *XML firewall* is needed.

The seriousness of these new threats should not be understated. The article "The (XML) threat is out there..." by Bill Hines clearly defines the breadth and seriousness of different attacks that are possible to any service exposed using XML. You can find this article on IBM developerWorks® at the following address:

`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 by using Web services, several important security steps (recommended by Gartner and others) are required:
  a. Inspect messages for well-formedness.
  b. Validate schema.
  c. Verify digital signatures.
  d. Sign messages.
  e. Implement service virtualization to mask internal resources via XML transformation and routing.
  f. Encrypt data at the field level.
- Systems hosting Web services, particularly public Internet-facing services, should consider the case for hardened gateway devices to act as XML firewalls to protect systems from XML threats.

DataPower appliances address these issues and more by delivering a robust XML firewall for the enterprise. DataPower appliances introduce sophisticated checks, including the following checks, on the incoming XML as illustrated in Figure 1-4:

- XML/SOAP firewall, filtering based on message content, headers, or other network variables
- Incoming/outgoing data validation
- Data schema validation (XML and binary)
- XML threat protection
- Single message XDoS protection
- Multiple message XDoS protection
- Message tampering protection
- Protocol threat protection
- XML virus protection
- Dictionary attack protection
- SQL injection protection

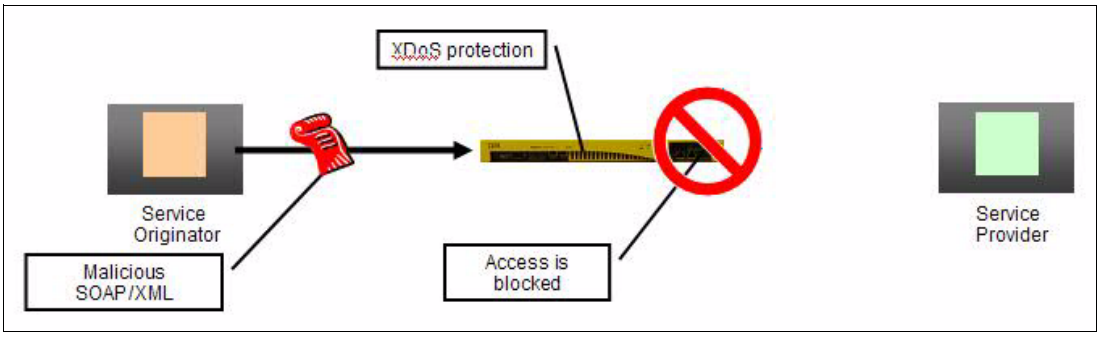


Figure 1-4 Application-layer thread protection with the DataPower appliance

Protection from XML threats is only one way in which enterprise systems must be protected. In addition, DataPower appliances provide a sophisticated set of security capabilities, which includes the following selection:

- XML Web services access control

  DataPower appliances support a variety of access control mechanisms, including WS-Security, WS-Trust, X.509, SAML, SSL, LDAP, RADIUS, and simple client/URL maps. They can control access rights by rejecting unsigned messages and verifying signatures within SAML assertions.

- Authentication and authorization

  Appliances directly support IBM Tivoli Federated Identity Manager for capabilities such as mapping identities for downstream access and retrieve authorization in IBM Tivoli Access Manager.

- Field level message security

  DataPower appliances can selectively share information through encryption and decryption as well as the signing and verification of entire messages or of individual XML fields. These granular and conditional security policies can be based on nearly any variable, including content, IP address, host name, or other user-defined filters.

Functional acceleration

XML has proven to be a great force in the software industry, and all the more so with the current focus on SOA and the related increased adoption of Web services. Clients need XML for implementing SOA, but cannot afford to spend precious CPU cycles processing it. The DataPower appliance pushes the XML processing into the firmware and enables enterprises to focus applications on doing actual business logic.

The flexible, self-describing, language-independent format of XML makes decoupling partner systems much easier. However, the heavy reliance on XML for data transfer between services presents problems. For example, XML can result in lengthy message payloads and large amounts of overhead for schema validation and parsing. The processing overhead of dealing with XML can tax application servers and middleware infrastructure, drastically decreasing performance.

The evolution of network infrastructure has seen an increasing trend toward replacing general purpose software systems with dedicated hardware for increased performance. In this same way, there is an evolution toward the usage of dedicated hardware for performing repetitive XML tasks such as parsing, schema validation, and XSL translation.

Service protocols based on XML also lack any inherent built-in security mechanisms. SOAP over HTTP passes potentially sensitive data in plain text over the network. While there have been emerging standards, such as WS-Security, to help deal with security concerns, implementing these standards further drains computing resources on critical servers.

The IBM line of DataPower SOA appliances helps address the performance and security needs of enterprise-level SOA architectures by off-loading the XML processing onto dedicated hardware. In doing so, CPU resources are freed of application servers and middleware platforms to provide higher service throughput. The performance advantage of DataPower appliances are often close to seventy times higher than when using general purpose systems alone. When digital signature checking and message encryption/decryption take place, a great deal of overhead occurs in processing messages. The XML appliance can off-load this processing from application servers onto dedicated hardware that is capable of performing these tasks in a fraction of the time.

Figure 1-5 shows an example of a Web service flow that is sent encrypted by using WS-Security standards from a client through the Internet. The intermediary DataPower appliance decrypts and authenticates the message before forwarding it in the clear over the last mile hop to an eventual service provider. The appliance could have easily secured the last mile encrypted under a transport-level mechanism, such as SSL, while avoiding the expensive WS-Security processing on the service provider.

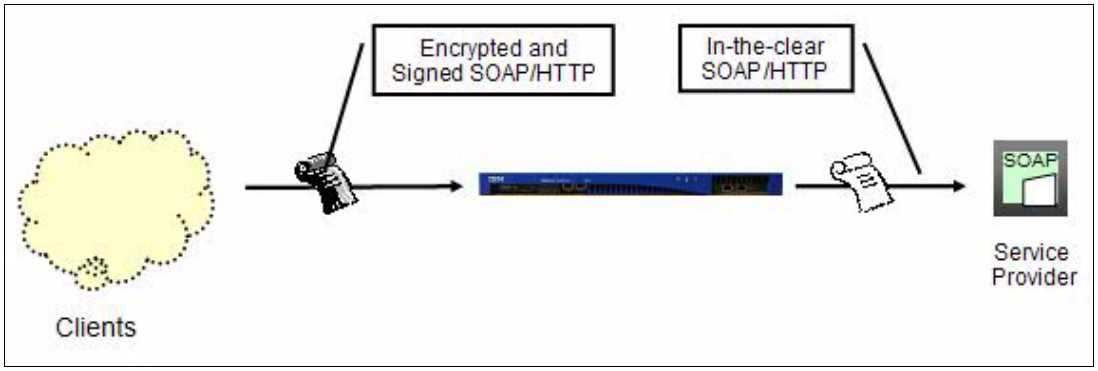


Figure 1-5 Functional acceleration with the DataPower appliance

Application-aware routing and data aggregation

The DataPower appliance enables the classification of data requests based on service- and application-level information. This information can be used in a number of ways:

- Routing

  The hosting of a service is likely to change over time as demands for its availability and resilience increase. Its URL, and possibly even its hosting server, may change. In a mature SOA, there may even be more than one implementation of the service. At runtime, the DataPower appliance is capable of choosing from these instances depending on the dynamic network conditions and service-level information, such as with the XPath language as shown in Figure 1-6.

  Clients of the service should remain unaffected by these changes. With DataPower appliances, a Web services proxy can be configured, decoupling the client completely from the implementation. Adjustments and translations can be made to all relevant metadata of the service from URL re-writing through to WS-Addressing and HTTP header manipulation. Routing information can be supplied in a variety of different ways including direct configuration, setting routing data as part of a transformation, and database or registry lookups.

- Data model and namespace

  Wherever possible, enterprise services should aim to expose a standardized data model. This model may, and arguably should, be different from that of the actual implementation of the service so that changes to the service do not affect the clients of the service. DataPower appliances allow wire-speed translation of data models by using XSLT, completely decoupling the client from the implementation.

- Versioning

  The routing and data model and namespace capabilities can be brought together to assist with service versioning. Clients should be insulated from version changes to a service interface. In practice, it is difficult to notify all users of a service interface change. Rather than maintain several versions of the implementation of the service, the DataPower appliance can translate between old and new URLs, host names, data models, headers, and any other relevant metadata.

- Message enrichment

  Figure 1-6 further shows how the DataPower appliance can retrieve data from a database to enable lookup-based routing. In addition, it can augment service requests as they pass through the appliance. In this way, messages can be enriched with data dynamically.

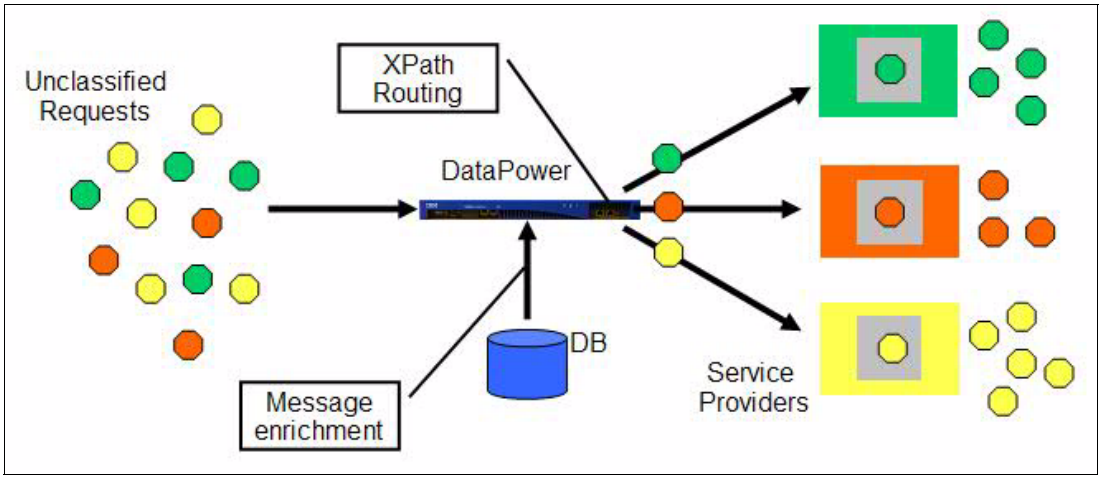


Figure 1-6 Content routing and aggregation with the DataPower appliance

Protocol and format bridging

With a DataPower appliance, services can be exposed by using different formats and protocols than the ones in which they are implemented. Ultimately, this translation capability encourages reuse and lowers total cost of ownership (TCO) by avoiding costly re-implementation of existing services. More interestingly, DataPower appliances can expose services by using several different formats and protocols at once, thus supporting a wide range of clients:

- Protocols

  Services can be exposed and called by using any combination of the typical protocols that are used for passing SOAP and XML messages in an SOA, such as HTTP, HTTPS and Java Message Service (JMS). Direct communication with WebSphere MQ and IMS Connect is also supported.

- Any-to-any transformation engine

  If the enterprise's standard protocols reach beyond the commonly accepted Web service data formats, appliances can parse and transform arbitrary binary, flat text, and XML messages, including EDI, COBOL copybook, ISO 8583, CSV, ASN.1, and ebXML. Unlike approaches based on custom programming, the patented DataGlue technology of the DataPower appliance uses a fully declarative, metadata-based approach that delivers wire-speed performance, thus lowering the heavy cost of integration bridging.

Figure 1-7 shows an example of how an existing enterprise service, such as a CICS application, can leverage a DataPower appliance to provide its Web service facade. In this example, the DataPower appliance is converting the SOAP format over the HTTP transport to the COBOL copybook format over the MQ transport. The support of the DataPower appliance for transports and protocols is *orthogonal*, meaning it can support any format over any transport to any other combination. For example, the illustration in Figure 1-7 could have easily been a COBOL/MQ facade to an existing SOAP/HTTP Web service.

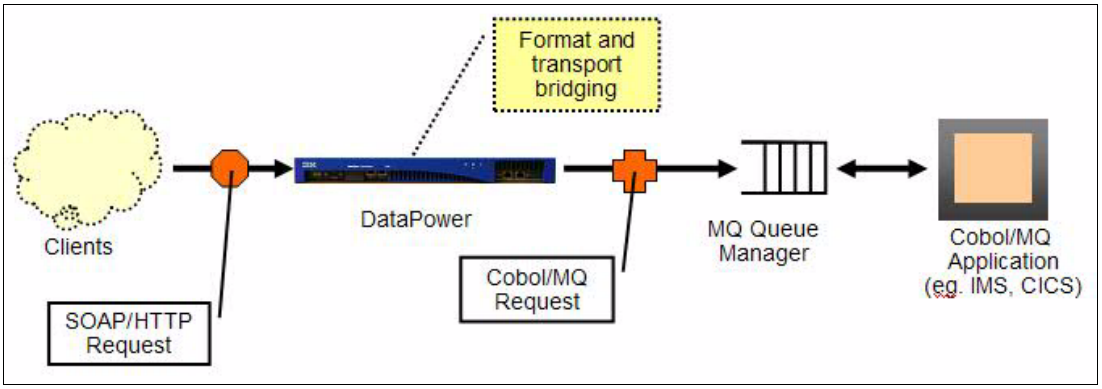


Figure 1-7 Bridging with the DataPower appliance

1.3 Configuration and usage of DataPower appliances

DataPower appliances provide a powerful intermediation architecture with a heavy emphasis on ease-of-use and consumability. Service intermediation is declaratively defined by using a notion of a flow of basic mediation actions. An action can correspond to basic functions of intermediary processing, such as routing, decryption, or logging. By using these building blocks, the administrator builds these collective flows by augmenting actions as processing steps. To this end, the DataPower appliance provides a powerful WebGUI as shown in

Figure 1-8. It shows the palette of common mediations (actions) that can be dropped in the message processing policy.

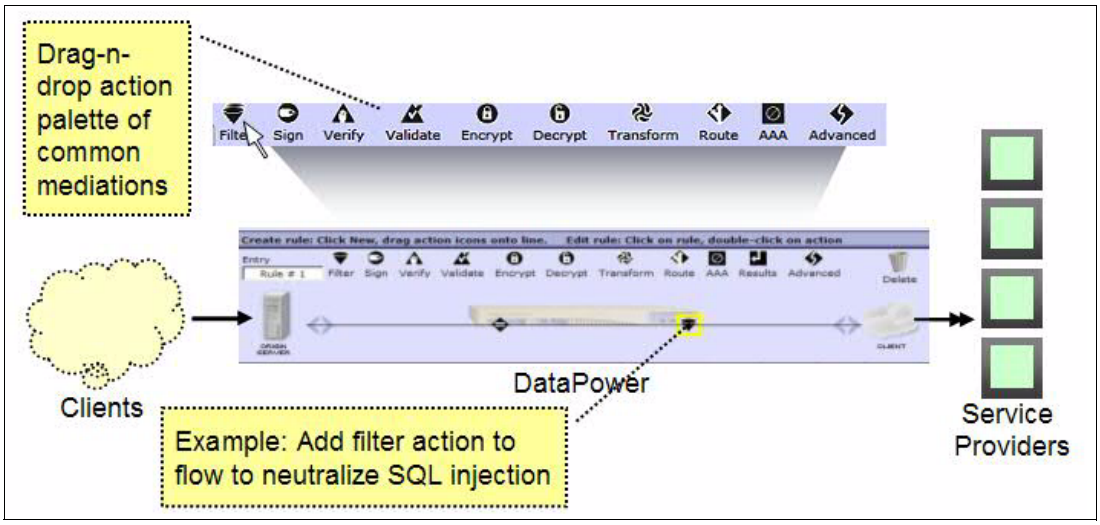


Figure 1-8 Award-winning WebGUI of the DataPower appliance

In addition to the WebGUI, the DataPower appliance provides a CLI that is accessible via SSH and Telnet. Programmatic support is enabled through XML management interfaces, such as the Service-Oriented Management API and the Appliance Management Protocol (AMP). An Eclipse plug-in enables tooling support for configuration. Multiple appliances can be managed together as part of a set through the use of IBM Tivoli Composite Appliance Management System Edition for WebSphere DataPower.

1.4 SOA governance

The clear fit of the DataPower appliance for gateway-style use in the ESB pattern means that it becomes the single entry point for service requests. This is the primary place to perform service virtualization and impose common policies on the services of the enterprise. Those common policies and information concerning where services are to be found can be configured directly into the gateway. However, they are really part of the wider cataloging of services for the enterprise. It is preferable if the gateway simply acts upon the policies set at the enterprise level. This is one of the reasons why a service registry is needed.

With the introduction of the WebSphere Service Registry and Repository product, all of the related products in the SOA Foundation suite are being upgraded to include mechanisms to contact the registry at run time or interact with the registry during configuration time. This includes integration with other key ESB products such as WebSphere Enterprise Service Bus and WebSphere Message Broker.

The following examples of registry usage may be relevant to DataPower appliances:

- Validation

  Access is given to schemas for validation of XML messages at configuration time. This is principally relevant to the XML firewall role for DataPower appliances, but potentially also to the ESB Gateway.

- Service virtualization

  Access is made to the registry at run time in order to acquire the actual endpoint based on a logical profile for the service. This is particularly applicable when DataPower appliances are used in the role of an ESB Gateway.

- Policy

  Access is given at run time, configuration time, or both to such policies as a WS-Policy from the registry. This can be for either the XML firewall or for the ESB Gateway roles, with different types of policy information being relevant to each. This is also particularly relevant to the gateway to multiple ESBs role where layers of ESB Gateways each have a policy.

- Availability and performance

  Runtime access is given to data-enable choices between implementations of services based on their availability and performance characteristics. This is particularly relevant to the ESB Gateway role.

- Namespace translation

  Access is given to standard transformations between known namespaces, most probably provided as XSLT files. This is particularly relevant to the ESB Gateway pattern where routing between different versions of a service might be required to seamlessly support old clients against new implementations of a service. This can also be relevant where services are promoted up the gateway hierarchy and need to conform to different object or namespace structures.

The reach of the registry is much wider than this alone. You can find more detail about WebSphere Service Registry and Repository in the following resources:

- “IBM WebSphere Service Registry and Repository V6 maximizes the business value of service-oriented architecture (software announcement)”

  http://www-306.ibm.com/common/ssi/rep_ca/0/897/ENUS206-230/ENUS206-230.PDF

- *WebSphere Service Registry and Repository Handbook*, SG24-7386

  http://www.redbooks.ibm.com/abstracts/sg247386.html?Open

2

Getting started

In this chapter, we provide a step-by-step procedure to initialize the DataPower appliance for use in minutes. We explain how to install and configure the appliance, log in via the WebGUI Control Panel, and set up two simple services that are provided by the device. We introduce the following two basic services:

- *Web Service Proxy Service*, which presents a Web service facade regardless of the nature of the back-end service
- *XML Firewall Service*, which redirects client requests for a Web page to an HTTP server

You can use the services for verification that the initialization procedure was successful.

2.1 Installing the device

In the following steps, we summarize the installation of an IBM WebSphere DataPower XML Appliance (XA35 XML Accelerator, XS40 XML Security Gateway, or XI50 XML Integration Appliance) in a rack. We include additional part numbers that are needed for use with IBM Enterprise Rack (Power Systems).

2.1.1 IBM Enterprise Rack (Power Systems)

IBM has conducted Rack Fragility Testing for the DataPower appliance in the IBM Enterprise Rack. For clients who want to install and integrate DataPower appliances with Power Servers in an IBM Enterprise Rack, note that the following parts are required for installation:

- PN 74F1823, Nut clip (2x)
- PN 26H7213, M5 screw (2x)

IBM is planning to update the *IBM WebSphere DataPower Common Installation Guide* with these part numbers. You can download this guide from the Web at the following address:

https://www14.software.ibm.com/webapp/iwm/web/reg/download.do?source=swg-datapower&S_PKG=xi50_9003_s2&dlmethod=http

2.1.2 Required tools and cables

Installation of the IBM WebSphere DataPower XML Appliance requires the following customer-provided tools and cables:

- Medium cross-tip (Phillips) screwdriver

  The appliance chassis is assembled with security screws that prevent access beyond customer-accessible areas.

- Cables for connection to the appliance Ethernet ports

2.1.3 Rack mounting an IBM WebSphere DataPower XML Appliance

You use mounting rails and rail extenders to mount the appliance in a rack. To mount the rack:

1. Assemble the rail extenders and the mounting rails.
2. Attach both mounting rails and rail extender assemblies to the rack.
3. Adjust the position of the rails as necessary.
4. Tighten the screws.
5. Slide the appliance on the mounting rails.
6. Use the angle brackets or screws to secure the appliance to the rack.

2.2 Setting up the DataPower appliance

You set up the DataPower appliance by using the command line interface (CLI) with physical connectivity to the device and a serial cable from your PC. After you set up the appliance, you can access it for most purposes by using the WebGUI.

Important: The DataPower appliance is physically secure. If the appliance has been tampered with, or opened, the setup and configuration steps will not work, and the device must be returned to IBM.

For the initial configuration, you must have the following requirements:

- PC running the HyperTerminal program

  Ensure that the software is configured for standard 9600 8N1 (9600 baud, 8-bits per character, no parity, 1 stop-bit) operation. You can use other terminal programs if convenient.
- The DB-9 null-modem cable and both power supply cables shipped with the device
- USB-to-serial converter, if the terminal or PC is not equipped with a serial port
- IP address to assign to the appliance provided by your system administrator

2.2.1 Connecting the DataPower appliance

Connect both power supplies to the ac power by using the IBM cables that shipped with the DataPower appliance. Both power supplies must be connected to ac power, or the firmware will be in a failed state.

You connect to the appliance with the following steps. In our example, we use a PC running HyperTerminal and a USB-to-serial converter to connect the DataPower device® to the PC.

1. Plug the **DB-9 null-modem cable** into the appliance.
2. Connect the USB-to-serial converter into the PC and DB-9-null cable modem into the serial port in the front of the device.
3. Configure the management network interface (mgmt) for network connectivity. Configure any of the three other ports as well.

   Management Port: Use the Management Port (mgmt) to provide Web-based access to the device.

4. From your PC, click **Start** → **Programs** → **Accessories** → **Communications** → **HyperTerminal** to start HyperTerminal.
5. Create a connection. In the window that opens, in the Name field, type `DataPowerX150`. Select any icon and click **OK**.
6. Locate the power switch located at the back of the device and turn it on.
7. In the HyperTerminal connection window, you see a message indicating that booting is in progress. Wait for the login prompt.

2.2.2 Initializing the appliance

To initialize the appliance:

1. After the booting is complete, press Enter to access a login prompt (Figure 2-1) if one is not already visible.

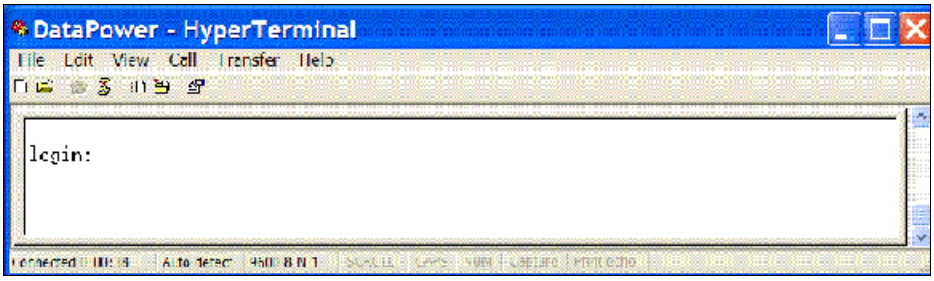


Figure 2-1 HyperTerminal login window

2. Login with user ID of `admin` and a password of `password`. In some cases, the password can be `admin` or `admin1`.
3. Review the licensing information, type `accept` to continue (Figure 2-2).

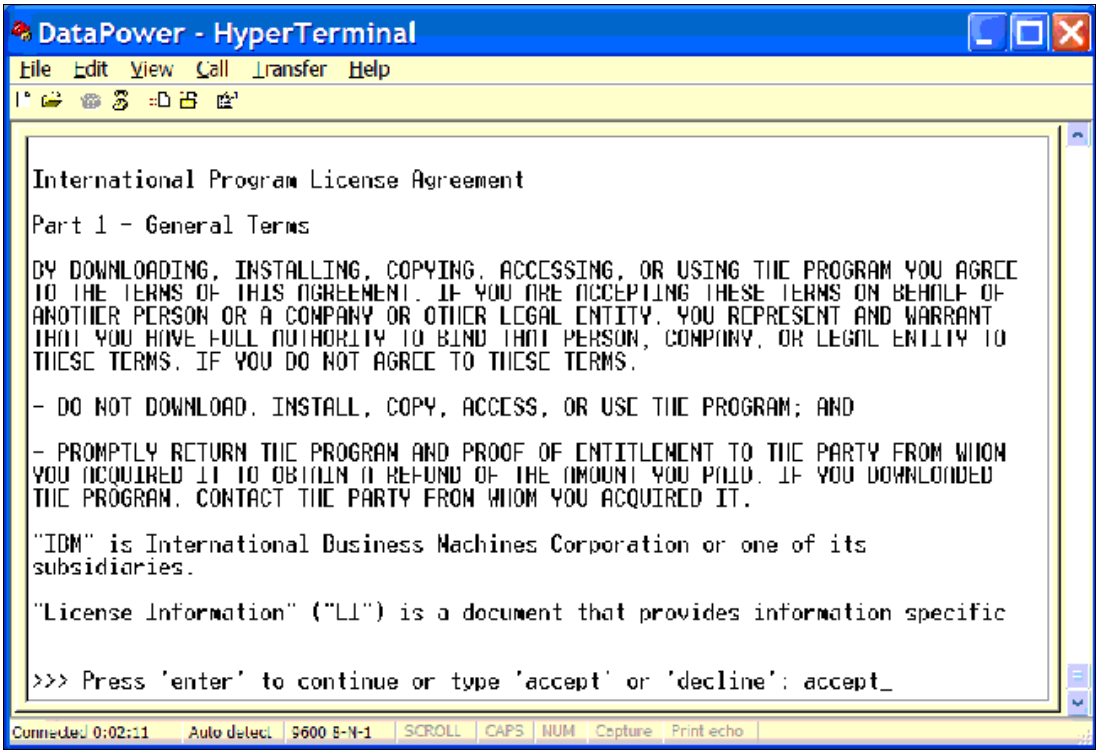


Figure 2-2 License agreement

4. When logging in as `admin` for the first time, change the admin password when prompted.

> **Remember**: Make sure you remember the new password since you cannot reset it.

Setting up the appliance by using CLI commands

For the remainder of the configuration steps, you use CLI commands. For more information about CLI commands, refer to the DataPower reference documentation that is listed in “Other publications” on page 185.

We use a small subset of the CLI commands to configure minimum connectivity and enable the Web Management Interface. After you configure the Web Management Interface, you can use a browser to access the appliance and perform additional interface and service configuration as defined in the WebGUI documentation. Figure 2-3 on page 22 shows a summary of the CLI commands that you enter in the HyperTerminal window.

1. Enter the Global configuration mode:

   ```
   configure terminal
   ```

   Global configuration mode: Most of the CLI commands that are used for configuring the device work only in the Global configuration mode.

2. Access the Management Interface Configuration mode:

   ```
   int mgmt0
   ```

3. Assign an IP address:

   ```
   ip address IPADDRESS/23
   ```

   The value 23: The value 23 is the short hand for assigning the subnet. Consult your system administrator for the appropriate subnet assignment.

4. Set the default gateway:

   ```
   ip default gateway [Gateway IP]
   ```

5. Exit the Management Interface Configuration mode:

   ```
   exit
   ```

6. Enter the Web Management Interface mode:

   ```
   web-mgmt
   ```

7. View the current settings:

   ```
   show
   ```

   Port 9090: The Web Management Interface runs on port 9090 by default.

8. Enable the service for use:

   ```
   admin-state enabled
   ```

   DataPower objects: All objects in DataPower are shipped as disabled. Therefore, you must be enable them before use.

9. Display the configuration for the Web Management Service:

   ```
   show
   ```

10. Exit the Web Management configuration:

    ```
    exit
    ```

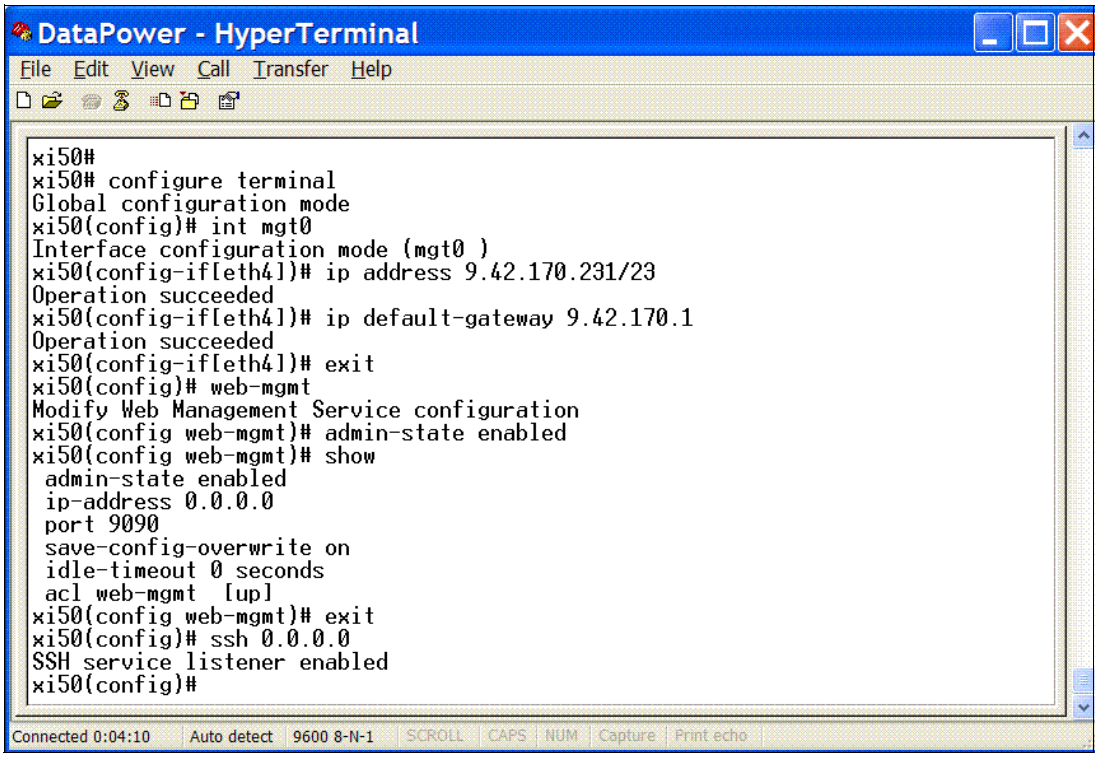


Figure 2-3 Web Management Service configuration

11. Save your configuration settings:

```
write mem
```

12. To verify that the WebGUI is available, launch your browser to connect to the appliance by using the HTTPS protocol:

```
https://9.42.170.231:9090
```

As an alternative to this step, you can access the appliance by using SSH. This method is often convenient to use when you must use CLI remotely. To enable SSH, type the **ssh** command as shown in Figure 2-4.

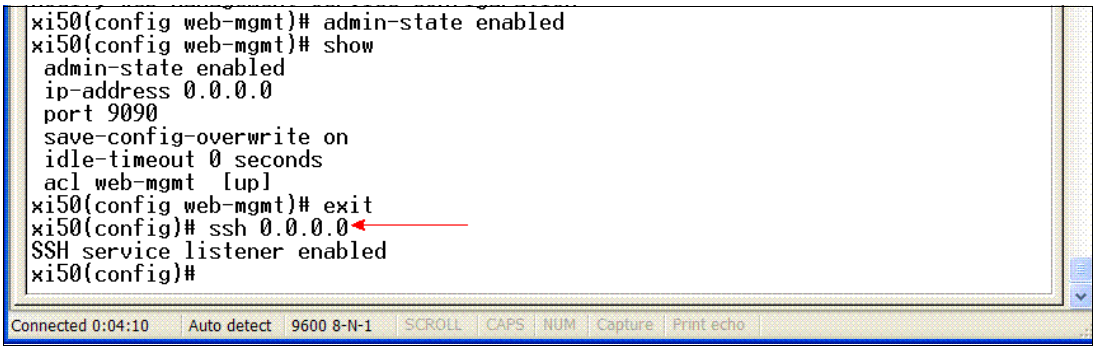


Figure 2-4 Enabling SSH

The DataPower Web Login Page opens as shown in Figure 2-5. If this page does not open, refer to the *IBM WebSphere DataPower Common Installation Guide* for guidance. You can download this guide from the Web at the following address:

https://www14.software.ibm.com/webapp/iwm/web/reg/download.do?source=swg-datapower&S_PKG=xi50_9003_s2&dlmethod=http

Figure 2-5 DataPower WebGUI

2.3 Launching the WebGUI

Log in to the WebGUI to access the DataPower console as explained in the following steps:

1. On the Welcome page (Figure 2-6 on page 24), in the User field, type your user name. In our scenario, we enter the default ID of `admin`.
2. Enter the Password for this account. The *admin* password was reset when the device was initialized.
3. Select the default domain. The default domain is the only domain that is available when the device is initialized.
4. Click the **Login** button.

Figure 2-6 DataPower WebGUI

When you log in, a Web page is displayed that has two sections: the Control Panel and the navigation bar on the left side of the page. The Control Panel is a graphical, Web-based tool that is used to configure and manage the DataPower appliance. Most of the management actions that are accessed via the CLI commands can also be performed by using the WebGUI. (For more information about the CLI, refer to the DataPower reference documentation that is listed in “Other publications” on page 185.) The WebGUI supports the full range of administrative activities, including the creation and management of services. In addition, every page of the WebGUI contains links with help messages.

As shown in Figure 2-7, the Control Panel is grouped into three main sections: Services, Monitoring and Troubleshooting, and Files and Administration. They contain shortcuts to some menus on the navigation bar that are accessed often.

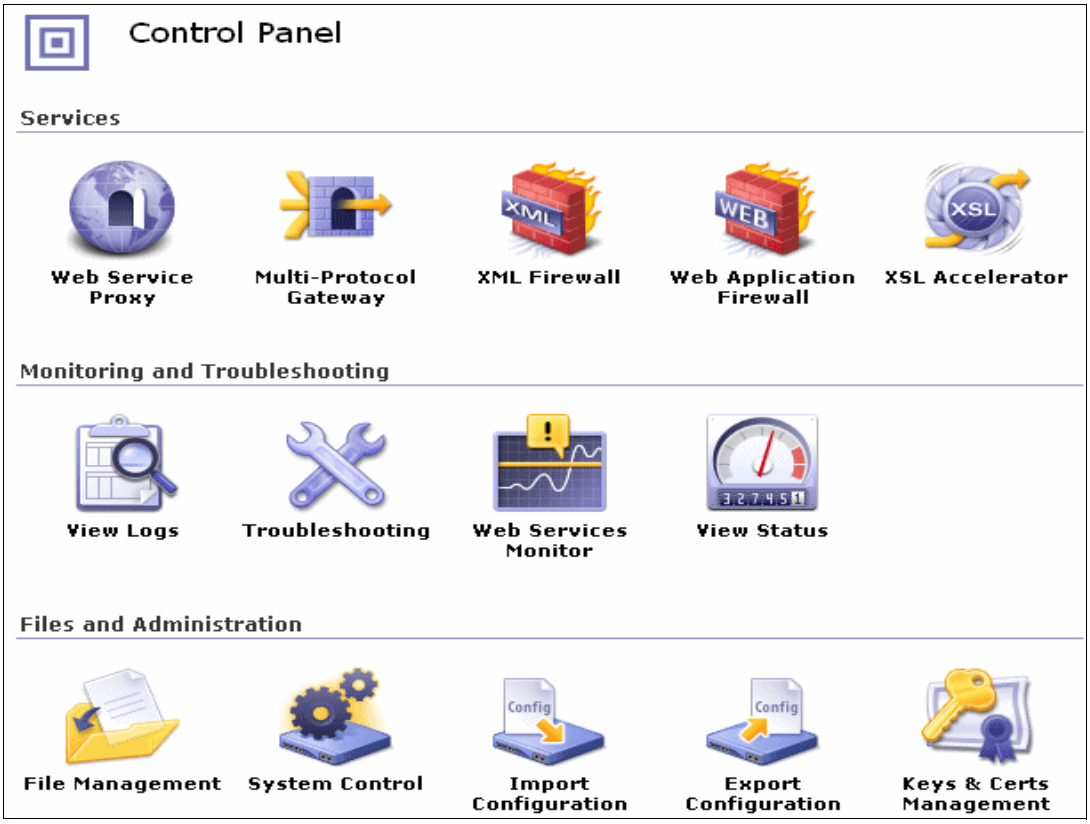


Figure 2-7 Control Panel

The navigation bar (Figure 2-8) consists of five menus that provide the ability to perform configuration or management tasks. We use each of the main tabs on the navigation bar in this example. The configuration items that are created are accessible from the navigation bar and are internally represented as objects for reuse.

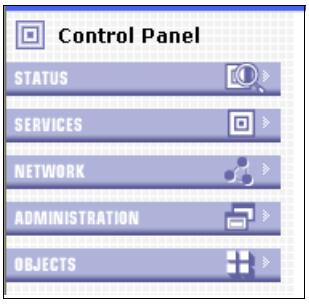


Figure 2-8 Navigation bar

All objects are accessible from either the Control Panel or navigation bar. For example, you can create an XML Firewall Service by either selecting XML Firewall on the shortcut menu of the Control Panel or navigating to **SERVICES** → **Edit XML Firewall** on the navigation bar.

2.4 Example: XML Firewall Service

The scenario in this section entails our working with the DataPower appliance to create a basic XML Firewall Service that supports an incoming HTTP request that is made by using a Web browser. The XML Firewall Service redirects the request to a Web site and returns the response page, if available, to the browser. In later chapters, we illustrate advanced functionality, such as creating a multiprotocol gateway to do XML transformation and protocol bridging.

In this scenario, we demonstrate how quickly, easily, and correctly you can configure an XML Firewall Service. Our scenario also serves as an example to ensure that the appliance was configured correctly. This simple XML Firewall Service provides the basis for implementing powerful security.

Basis for powerful security: As an aside, XML firewalls are implemented to provide “site-specific” XML security practices. Although simple, this XML Firewall Service can be used as the basis for implementing powerful security practices.

You can configure the XML Firewall Service to connect client requests that are transported over HTTP protocols to a back-end service by using the same protocol. You can create new services by using one of the following methods:

- Shortcuts
- SERVICES menu on the navigation bar
- CLI commands

In this scenario, we demonstrate how to use the shortcuts on the Control Panel. Before we begin to create our service, we create an application domain called *HelloDP*.

Application domain: An application domain is much like a private workspace, isolating objects in one domain from other domains. The default domain is the only domain available when the device is initialized, and therefore, you should not do any development in the default domain. Instead, create more than one domain in which to build and operate services.

In addition to ease migration, creating multiple domains makes it possible to restrict access to key system resources and makes it easier to restore the system to its default settings. Domains can contain any services that are provided by the device. Objects that are created in one domain are not visible to other domains.

2.4.1 Creating an application domain

To create an application domain:

1. On the DataPower console, from the navigation bar, select **ADMINISTRATION**. Under Configuration, select **Application Domains** (Figure 2-9). In the right pane, click the **Add** button.

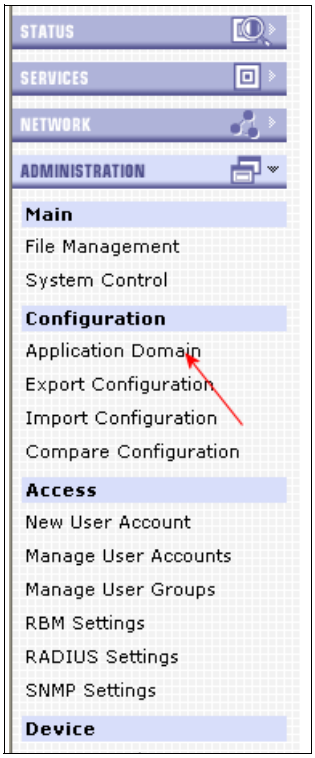


Figure 2-9 Create an Application Domain

2. On the Main tab of the Configure Application Domain page (Figure 2-10), for Name, type the domain, which is `HelloDP`. For Comments, type `Example ITSO Domain`.

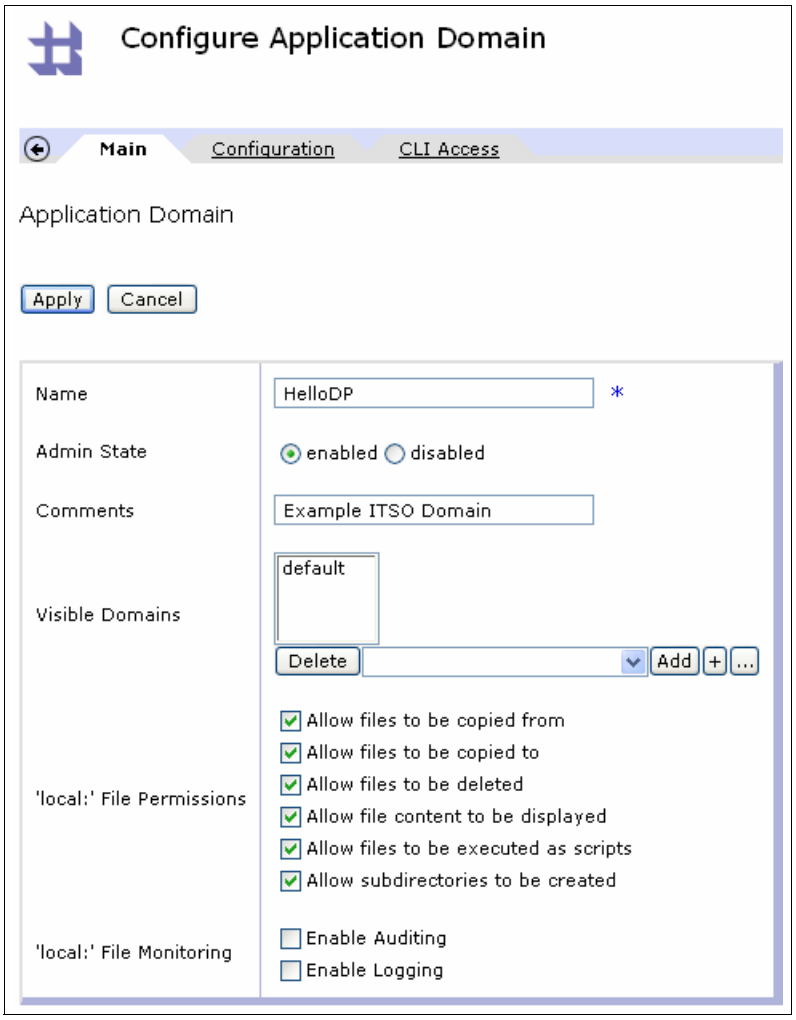


Figure 2-10 Main tab of the Configure Application Domain page

3. Click the **Configuration** tab (Figure 2-11). For Configuration Mode, leave it set to the default setting of **local**. If we want to import an existing domain, we select **import** from the list.

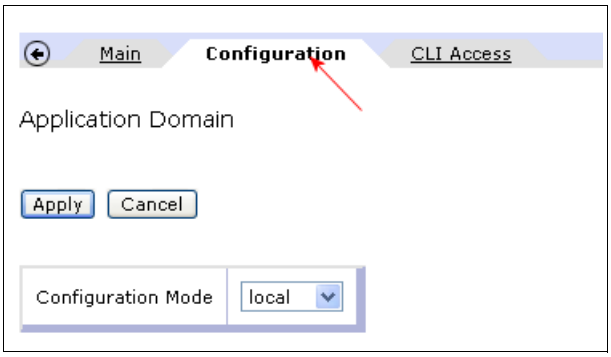


Figure 2-11 Configuration tab of the Configure Application Domain page

4. Click the **CLI Access** tab (Figure 2-12). No other users are created on this appliance. If we want to give other users access to this domain, we add them here. By default, the admin user has access to all domains. Click **Apply** to apply your changes to the appliance.

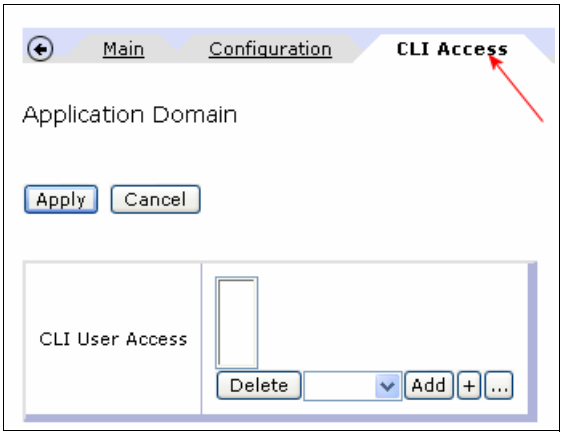


Figure 2-12 CLI Access tab

5. Verify that the new domain HelloDP was created with a status of *new* (Figure 2-13).

Figure 2-13 New HelloDP domain

6. Click **Save Config** to save your changes to your new configuration.
7. Click **Refresh** to verify that the status of your domain has changed to the *saved* status (Figure 2-14).

Figure 2-14 New HelloDP domain with a saved status

8. In order to create the service in the HelloDP domain, switch domains. At the top of the page on the right side, for Domain, select **HelloDP** (Figure 2-15).

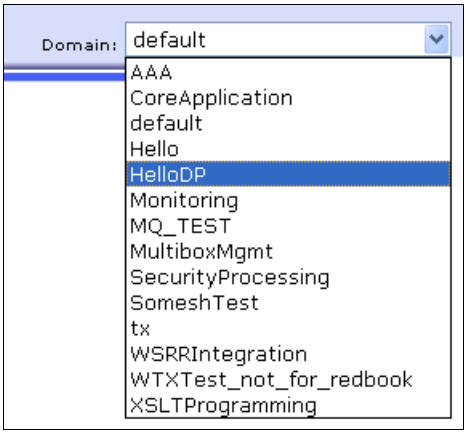


Figure 2-15 Switching from the default to HelloDP domain

9. On the Main DataPower Console page (Figure 2-16), verify that you are in the HelloDP domain. For the Domain field, you should now see `HelloDP`.

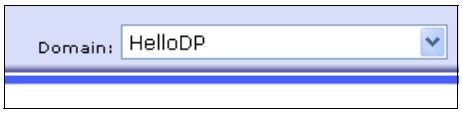


Figure 2-16 Welcome to the HelloDP

2.4.2 Creating the XML Firewall Service in the domain

To create a basic XML Firewall Service in the HelloDP domain:

1. From the Control Panel, click the **XML Firewall** icon (Figure 2-17).

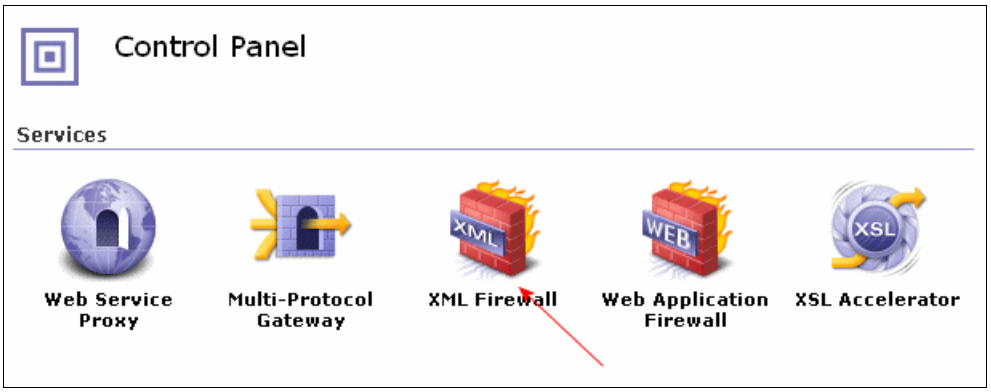


Figure 2-17 XML Firewall shortcut

2. On the Configure XML Firewall page (Figure 2-18), click the **Add Wizard** button.

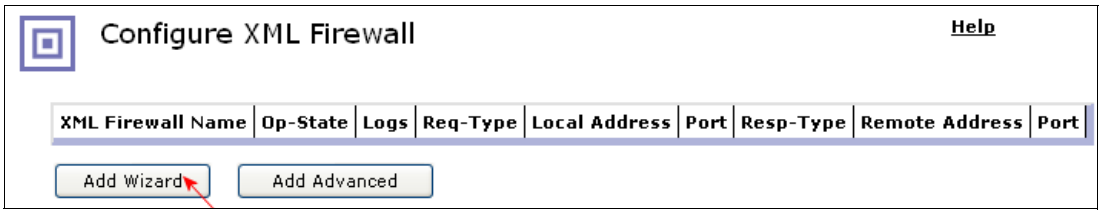


Figure 2-18 XML Firewall Configuration Wizard

3. On the Firewall Wizard page (Figure 2-19), accept the default of **Pass-Thru**. By using this option, the wizard creates an XML firewall with an empty processing policy, so that the appliance forwards all inbound requests for the assigned port without doing any processing. In later chapters, we explain how to create policies for advanced processing of requests and responses. Click **Next**.

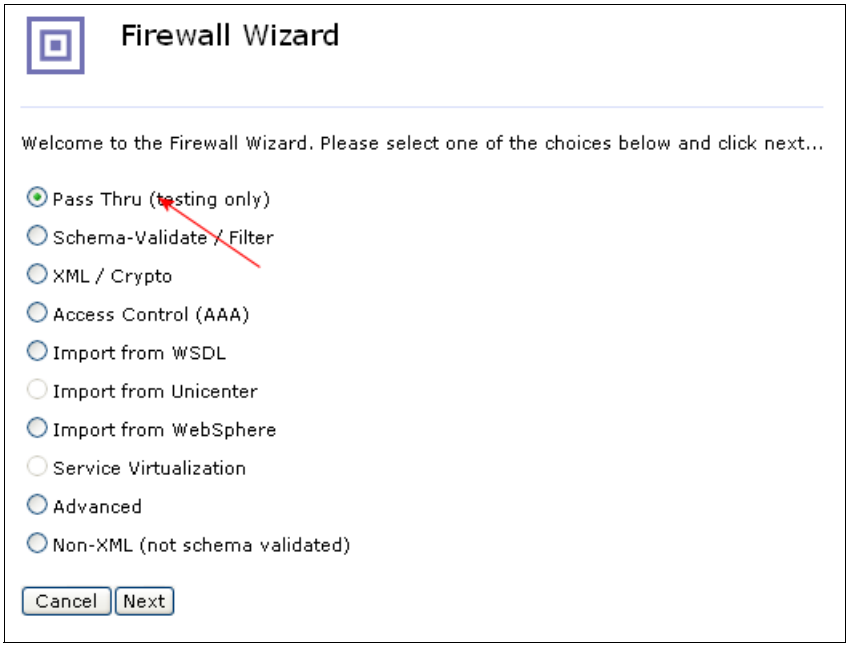


Figure 2-19 Creating a Pass Thru service

4. On the Create a Pass Thru XML Firewall Service page (Figure 2-20), for Firewall Name, type `Hello_XMLFW` and click **Next**.

Figure 2-20 XML Firewall service name

5. On the Select Service Type page (Figure 2-21), select the firewall type.

   You have two choices for the firewall type, loopback and static. In the *loopback mode*, the device sends its outbound messages back to the client that sent the request. With the *static mode*, the device can send outbound messages to a specified back-end server.

   For Firewall Type, leave the default selection **static-backend**. Click **Next**.

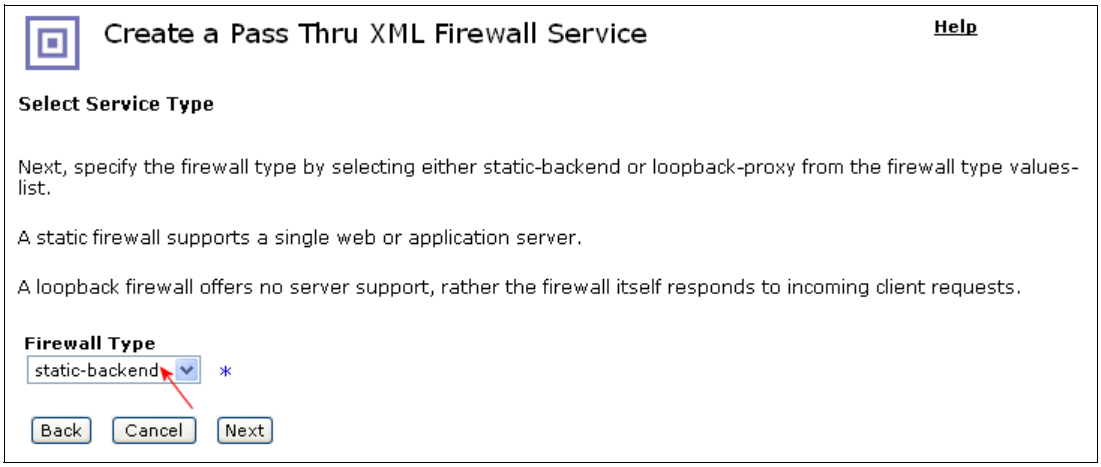


Figure 2-21 Selecting the firewall type

6. On the Back End (Server) Information page (Figure 2-22), for Server Address, type `www.ibm.com` and for Server Port, type `80`. Click **Next**.

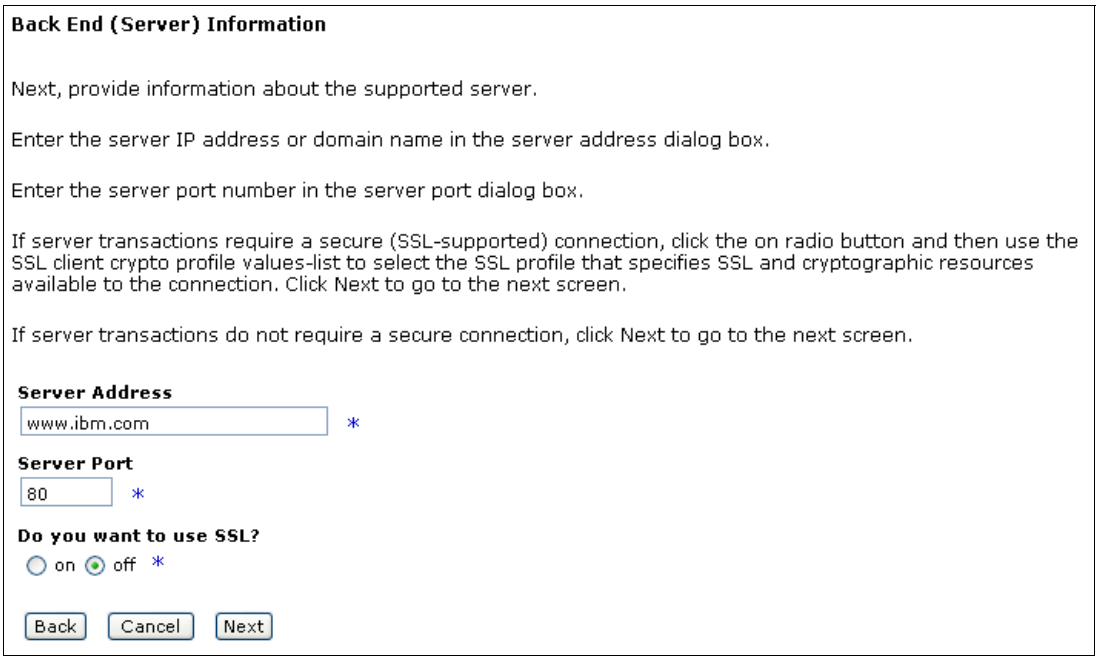


Figure 2-22 Back End (Server) Information page

7. On the Front End (Client) Information page (Figure 2-23), in the Device Port field, type 3055.

   This number becomes the TCP port for all HTTP connections to this gateway. The port number must be unique. If more than one user is defining a service on the device, there will be a conflict. The device does not count the port as used until after you complete all of the steps.

   Device Address: An IP address of 0.0.0.0 means the service will listen on all IP addresses that are assigned to the device. In production, you might choose a specific IP addressed that is assigned to one of the four network interfaces. In this example, we accept the default of 0.0.0.0.

   Click **Next**.

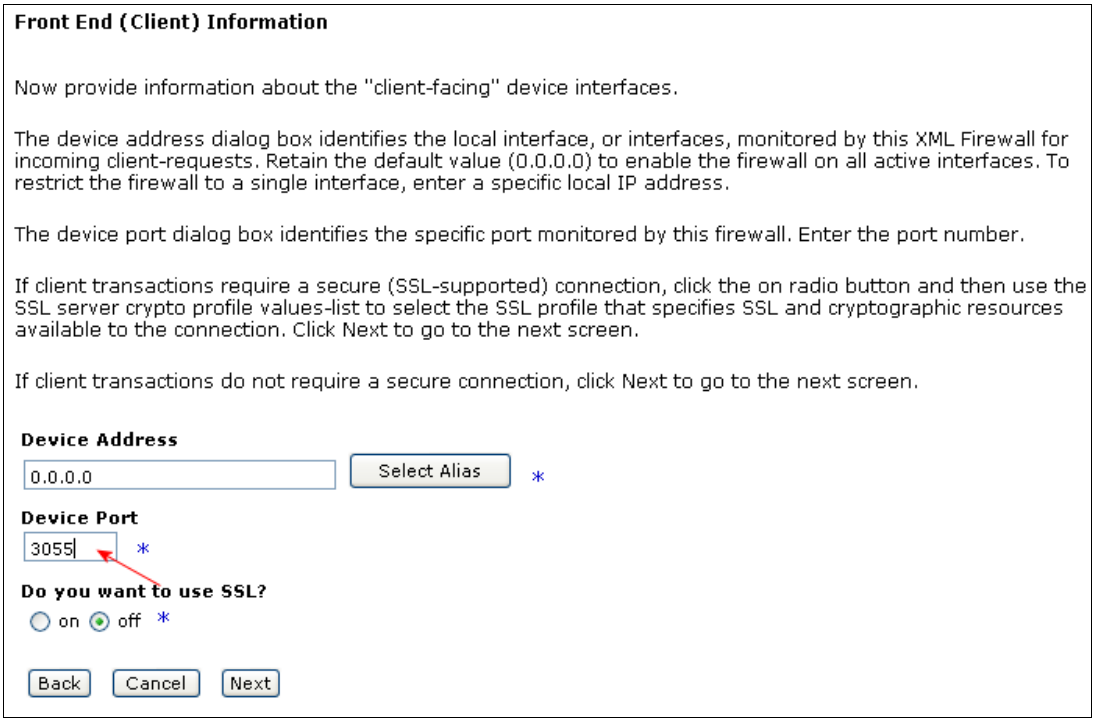


Figure 2-23 Front End (Client) Information page

8. On the final page of the wizard (Figure 2-24), review your configuration and select **Commit**.

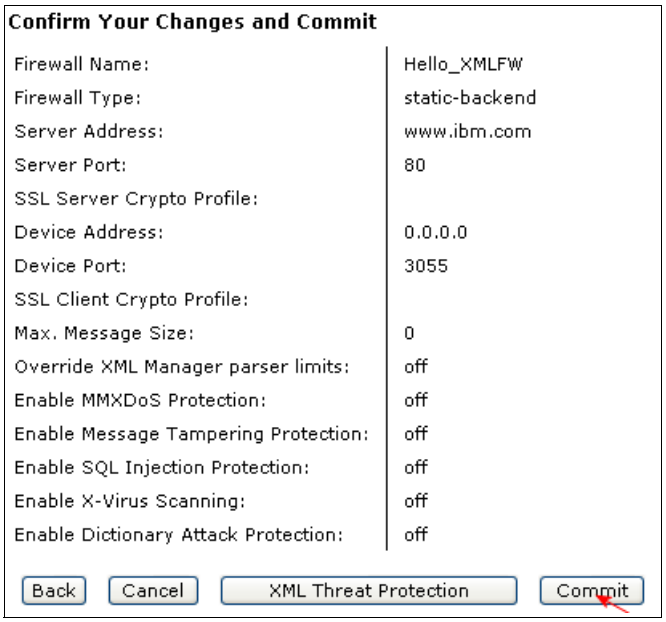


Figure 2-24 Confirming and committing the changes

9. In the message window that opens, you see the message "`You have successfully created your XML Firewall Service`." Click **Done**.
10. Click **Save Config** to save the changes that you made to the HelloDP domain.
11. In the navigation bar, select **Objects** and under Services, select **XML Firewall Service** (Figure 2-25).

Figure 2-25 Services object menu

12. You see the newly created XML Firewall Service with the operation state set to *up* (Figure 2-26). Under Logs Probe, click the **magnifying glass** icon to view the system log for this service. Review the system log and then close the window (inset in Figure 2-26).

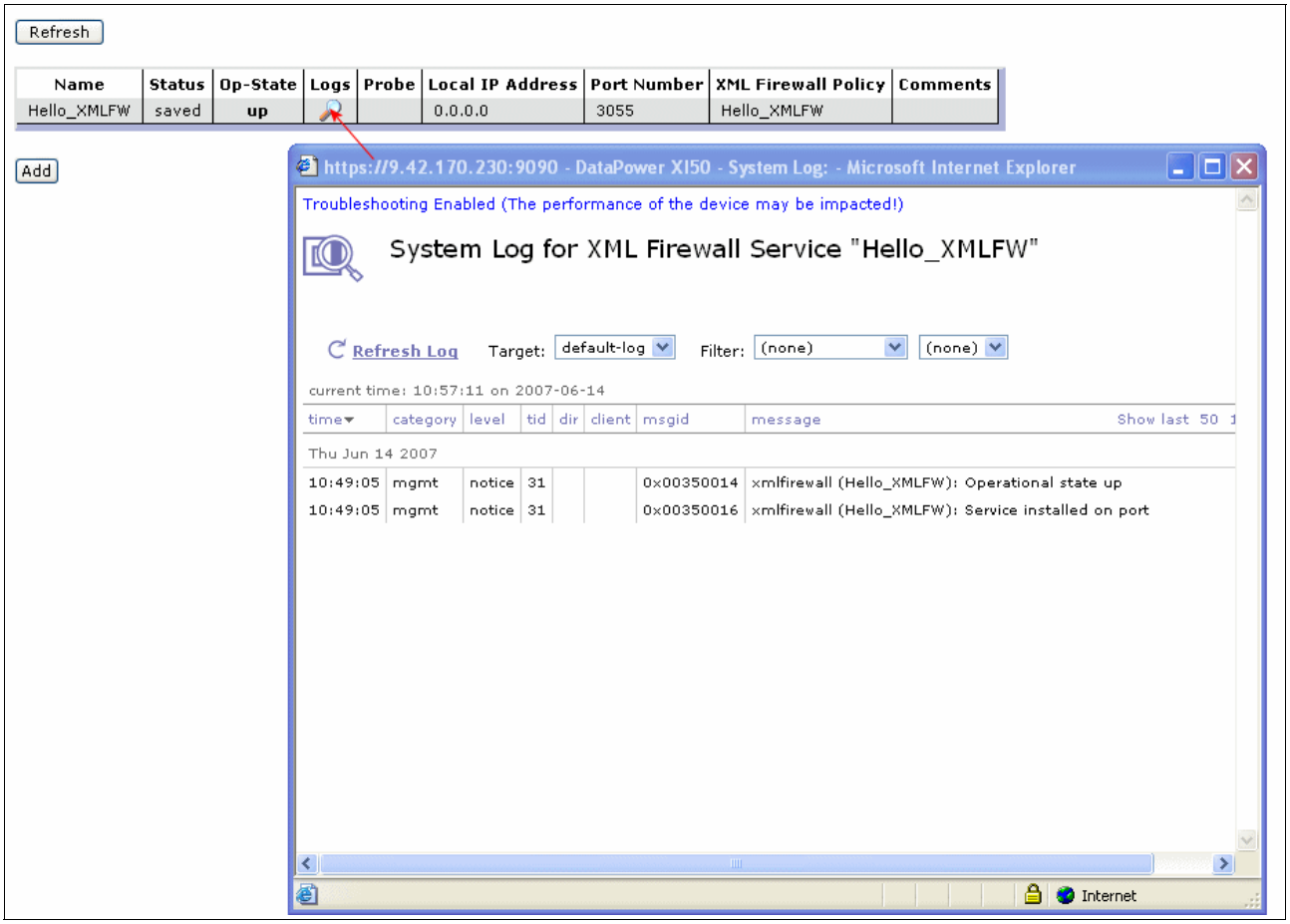


Figure 2-26 System Log for XML Firewall Service "Hello_XMLFW"

13. Make two small configuration changes to the service. First you must change Request and Response Types to Pass-thru, and second you must enable HTTP GET.

    a. Select **Hello_XMLFW** to open the XML Firewall Service for editing (Figure 2-27).

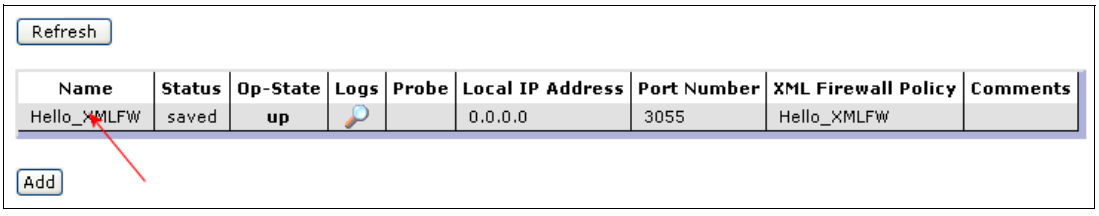


Figure 2-27 Editing the XML Firewall Service

b. Scroll down to the Characterize client traffic type and Character Service traffic type fields. Select **Pass-thru** for both fields (Figure 2-28).

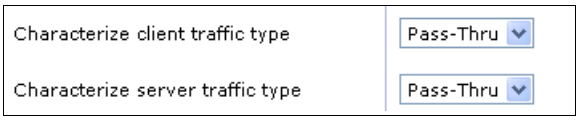


Figure 2-28 Client and server traffic

c. Enable the service to accept HTTP GET. Click the **HTTP Options** tab (Figure 2-29). For Disallow GET (and HEAD), select **off**. Click **Apply** to apply your changes to your service.

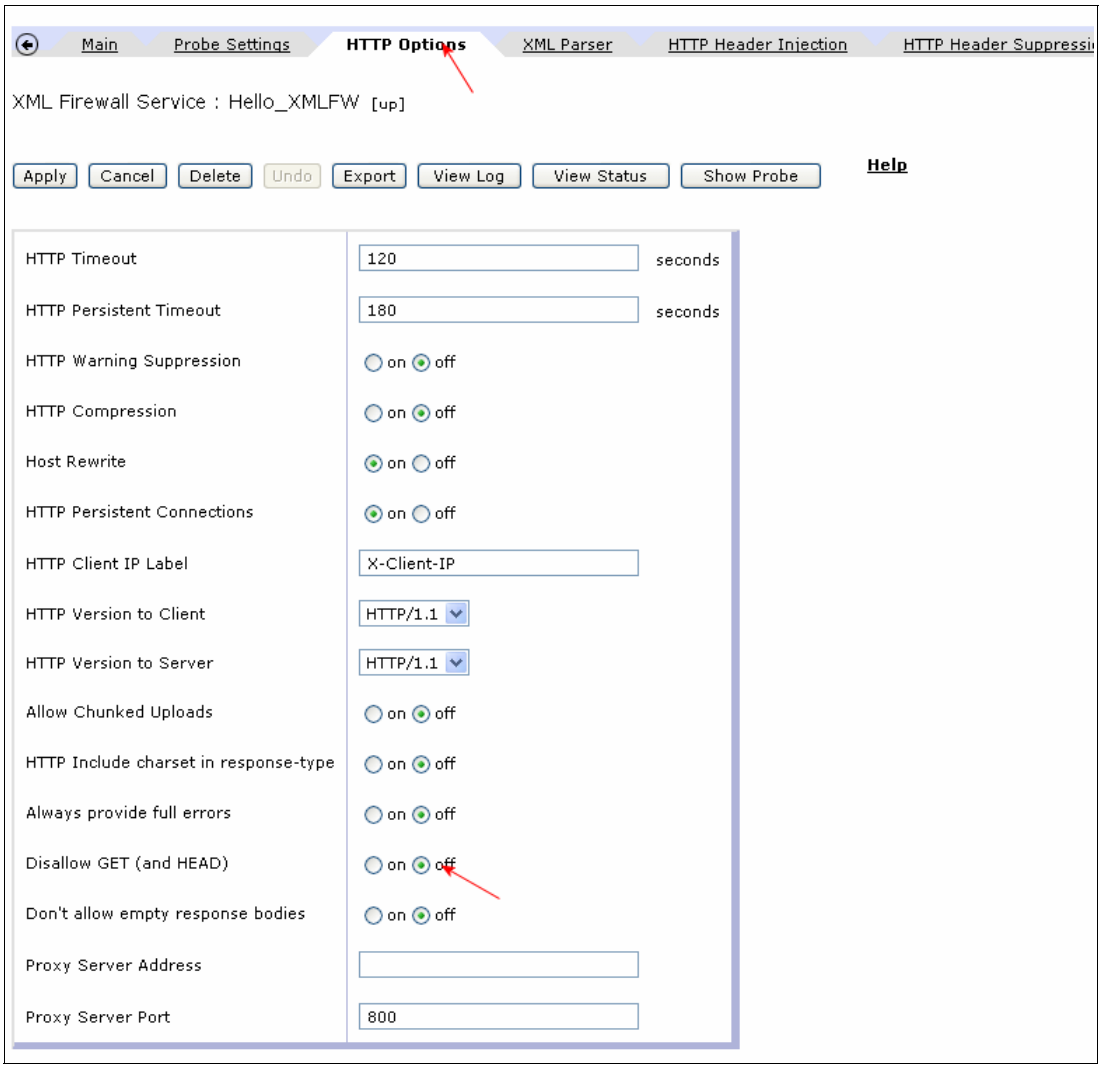


Figure 2-29 HTTP Options Tab

14. Click **Save Config** to save the configuration changes to your HelloDP domain.

You are now ready to test your configuration.

2.4.3 Testing the Hello_XMLFW firewall

You use a Web browser to test the basic XML firewall. In Internet Explorer®, in the Address field, type the following URL for DataPower:

`http://`*`DataPowerIP`*`:3055`

Here *DataPowerIP* is the IP address of the device, and *3055* is the port that you defined in the Hello_XMLFW service. Press **Enter**.

The IBM home page (Figure 2-30) is displayed. The HTTP request from the browser was intercepted by the XML Firewall Service, which redirected it to the page that was defined in the Backend Server Address. The response from the server is returned to the client.

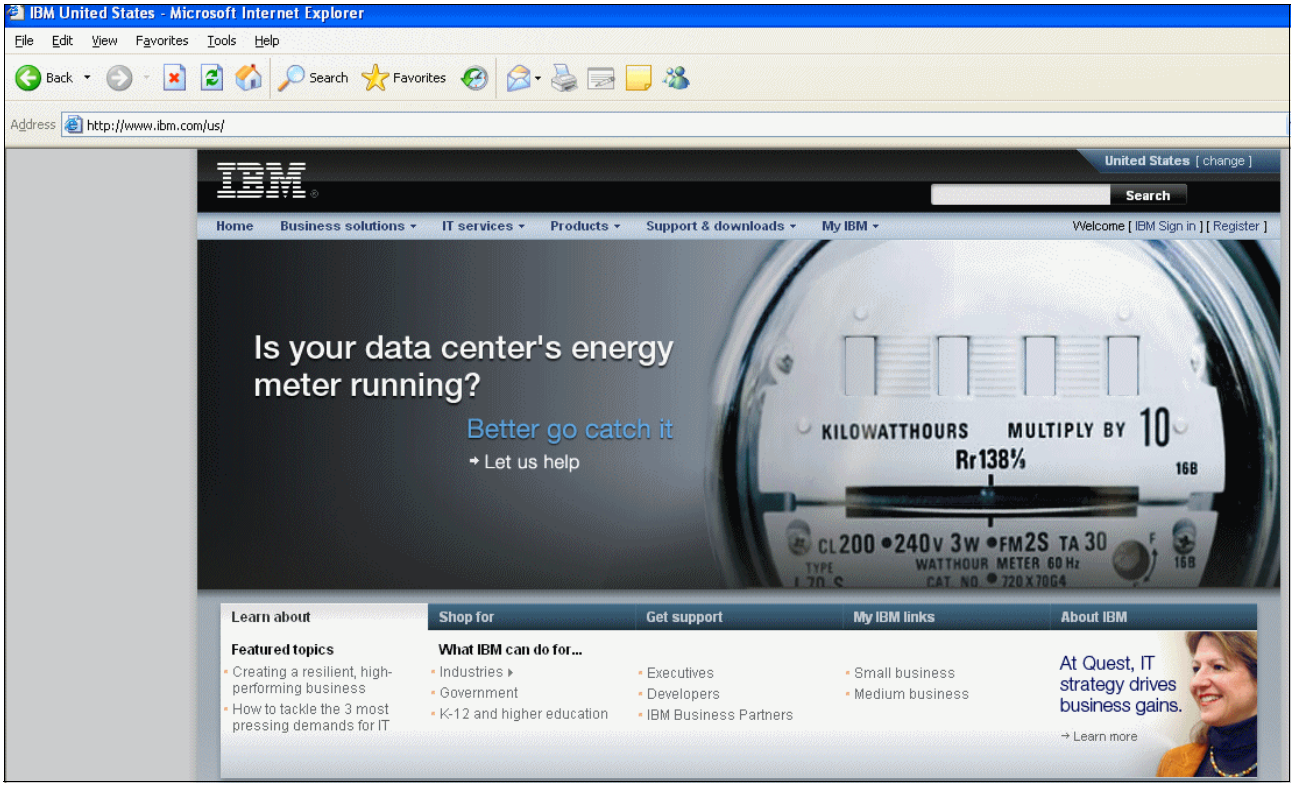


Figure 2-30 IBM home page redirected through the DataPower appliance

2.5 Example: Web Service Proxy Service

In this scenario, we create a simple Web Service Proxy Service that intercepts Web service calls to a back-end service. The Web service proxy offers easy configuration by using Web Services Description Language (WSDL), multi-step policies, and service-level monitoring to provide a Web service facade for arbitrary back-end services.

The Web service proxy provides quick service virtualization by simply uploading WSDL into the device. The Web Service Proxy provides a “living” virtual service that passes messages between the client and the real service, so that the client connects to the proxy and not to the back-end service. In doing this, the service consumer and the service provider do not need to be tightly coupled and bound to each other. The DataPower appliance can hide the details of accessing the service from the consumer.

2.5.1 Creating a Web Service Proxy Service

As we demonstrated in 2.4.1, “Creating an application domain” on page 27, you can create a service by clicking the shortcut in the Control Panel. In this section, we begin by demonstrating the second option for creating a service, which is to use the menus on the navigation bar. To create a new Web service proxy:

1. From the navigation bar of the WebGUI (Figure 2-31), select **SERVICES**. Under Web Service Proxy, select **New Web Service Proxy**.

Figure 2-31 Creating a new Web Service Proxy

2. On the Configure Web Service Proxy page (Figure 2-32), in the Web Service Proxy Name field, type `HelloWSProxy`. Click **Upload** to upload the Primes.wsdl file that we include in the additional materials of this paper. See Appendix B, “Additional material” on page 183, for information about downloading the materials.

> **Primes Web service:** This sample scenario requires that the Primes Web service is installed in WebSphere Application Server.

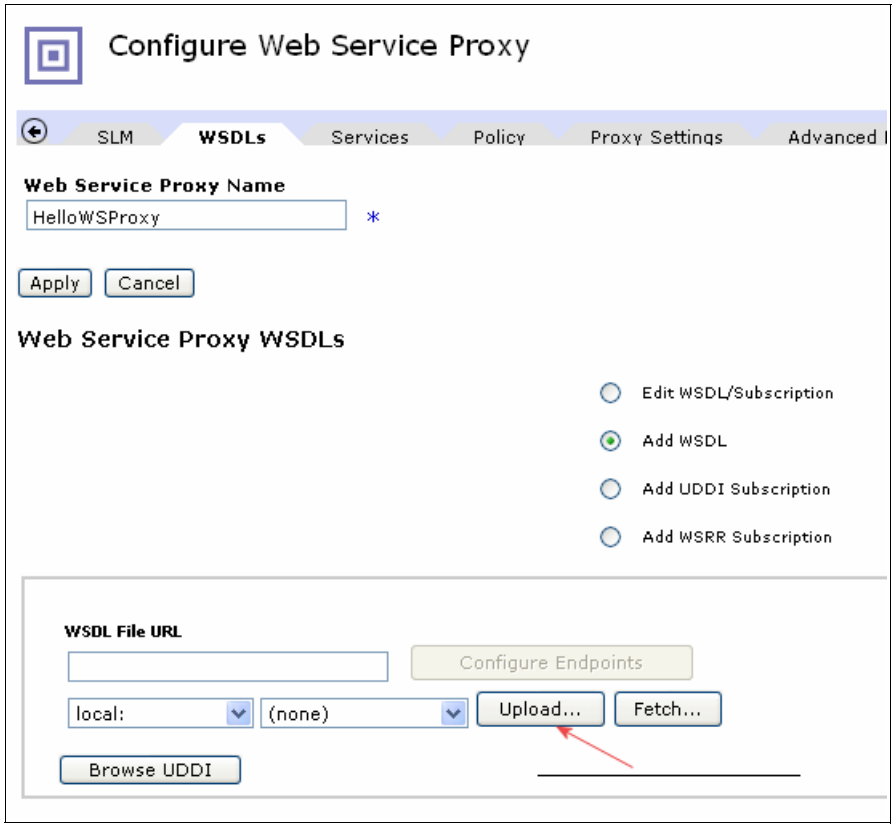


Figure 2-32 Configure Web Service Proxy page

3. Upload the **Primes.wsdl** file to a local directory as illustrated in Figure 2-33. Then click **Upload**.

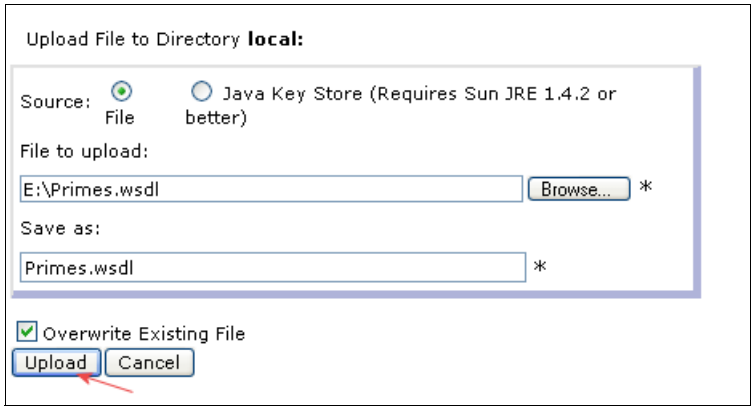


Figure 2-33 Uploading the WSDL file to the device

4. Under PrimesService - Primes (Figure 2-34), complete these steps:
   a. In the URI field, type `/hello`.

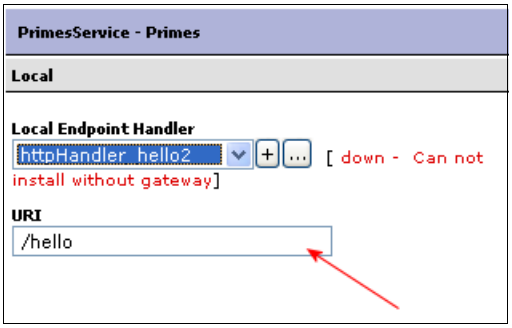


Figure 2-34 Adding a URI for the Web Service Proxy Service

   b. For Local Endpoint Handler, click the **+** button.
   c. From the list, select **HTTP Front Side Handler** (Figure 2-35).

Figure 2-35 Creating an HTTP Front Side Handler for our service

5. On the HTTP Front Side Handler page (Figure 2-36), specify the properties of the Local Endpoint Handler:
   a. In the name field, type `httpHandler_getPrime`.
   b. In the Port Number field, enter the port number, for example `3067`.
   c. For an HTTP protocol handler, it is only necessary to enter a unique port number. The new Local Endpoint Handler is selected in the Web Service Proxy. Leave the endpoints and properties unchanged. You can adjust the remote endpoint configuration if desired.
   d. To apply this WSDL to the Web Service Proxy, click **Apply**.

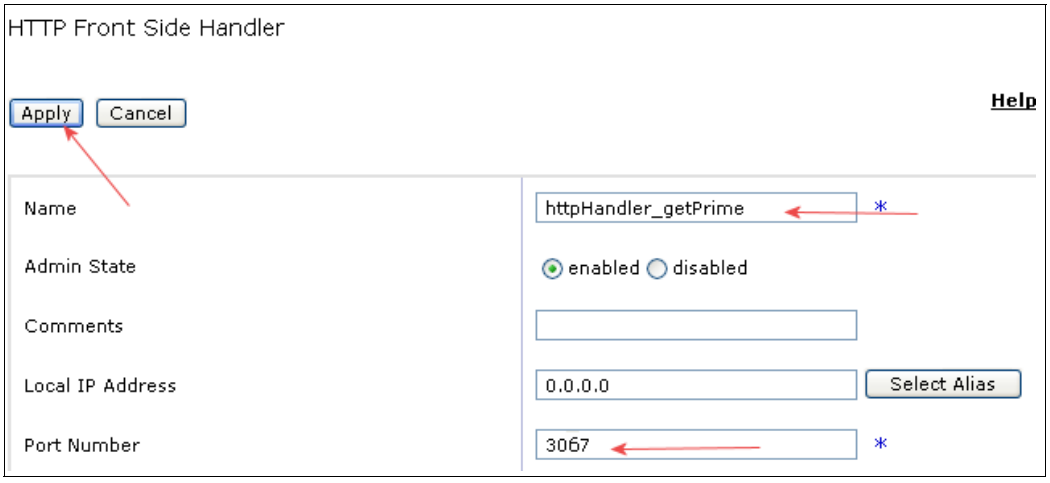


Figure 2-36 Naming the Local Endpoint Handler

6. Click **Save Config** to save configuration changes (Figure 2-37).

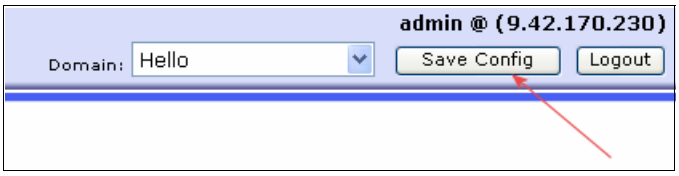


Figure 2-37 Saving your configuration changes

7. Review the configuration as shown in Figure 2-38.

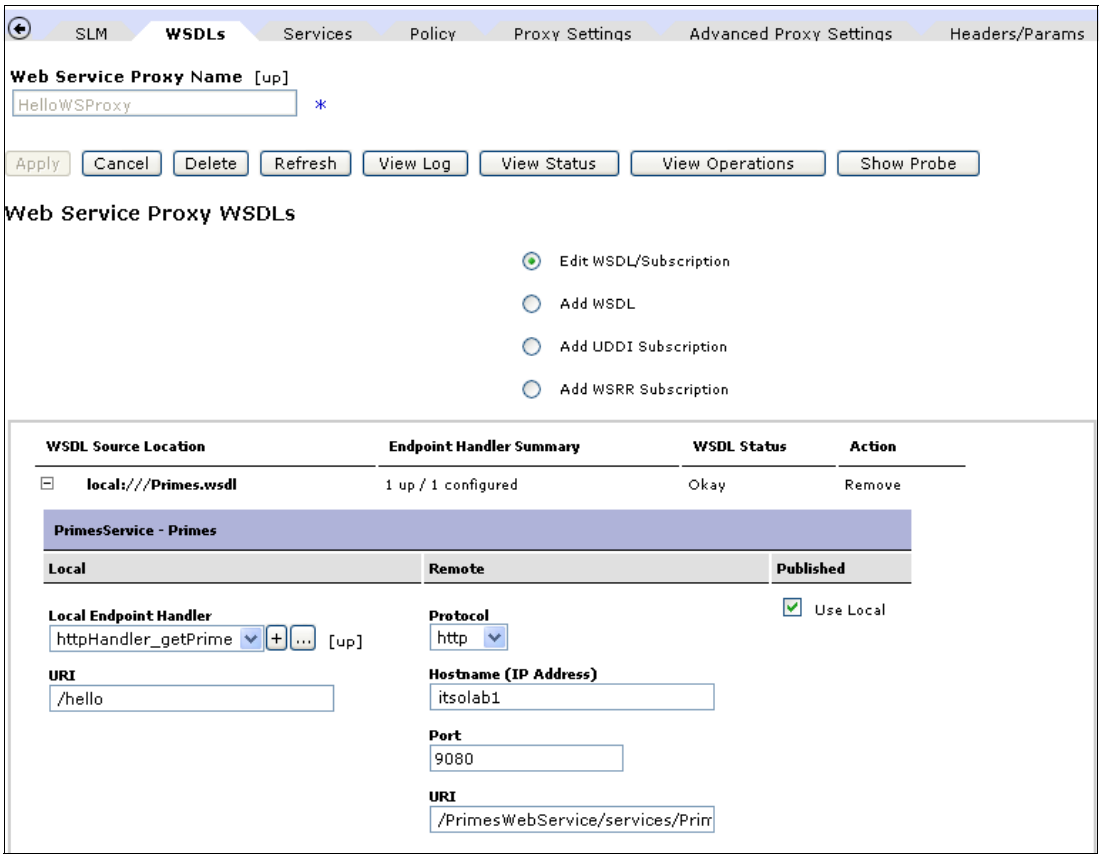


Figure 2-38 Configuration for the Web Service Proxy Service

Note: The Primes.wsdl file that was uploaded in this scenario was stored on the device. From the navigation bar, click **Administration** → **File Management** → **local**. You see the WSDL files and any miscellaneous local files. This directory typically contains files that are used by the services.

2.5.2 Testing the HelloWSProxy service

To test the Web Service Proxy, we use the Rational® Application Developer Web Services Tool to invoke a Web service called *getPrime*. Refer to the additional materials for details about configuring this Web service for use. For more information, see Appendix B, “Additional material” on page 183.

1. Launch Rational Application Developer.
2. Import the Primes.wsdl file into Rational Application Developer.
3. Right-click **Primes.wsdl** and select **Web Services** → **Test with Web Services Explorer**.

4. In the Web Services Explorer window (Figure 2-39), only one service, called *getPrime*, is defined in this WSDL. Complete the following steps:
   a. For Endpoints, type the following URL to add a new endpoint to point to the new Web Service Gateway that we created:

      `http://itso:3067/hello`

   b. In the numDigts int field, type 7 or another number.
   c. Click **Go** to post the getPrime request

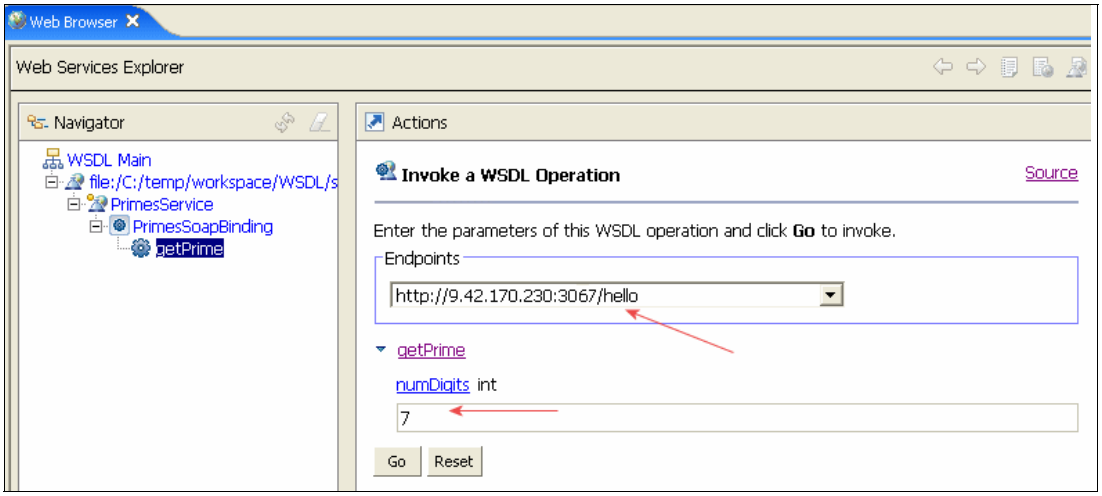


Figure 2-39 Web Services Explorer

The getPrimeResonse.xml message is returned as shown in Figure 2-40.

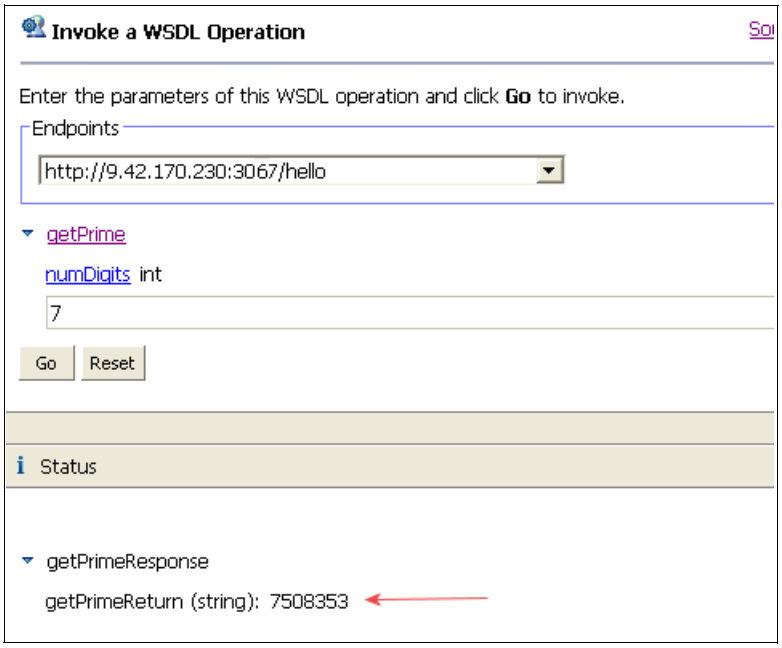


Figure 2-40 Response from the Web service

Using the curl tool

Alternatively, follow these steps if you plan to access the sample Web service:

1. Obtain the primes.xml file from the additional materials that are supplied with this paper. For details, see Appendix B, “Additional material” on page 183.
2. Invoke the **curl** tool to test the sample scenario:

   ```
   curl -X POST -d@primes.xml http://your datapower ipaddress:3067/hello
   ```

   cURL: cURL is a freeware program that is used widely to send HTTP requests from the command line. We use this program extensively in this paper. To download cURL, go to the cURL Web site at the following address:

   http://curl.haxx.se

If the Web Service Proxy Service is installed correctly and the back-end Primes Web service is installed correctly on the application server, the previous command should have the following result:

```
<?xml version="1.0" encoding="UTF-8"?>
          <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"><soapen
v:Header/><soapenv:Body><p234:getPrimeResponse
xmlns:p234="http://com.itso"><getPrimeReturn>2997249767</getPrimeReturn></p234:get
PrimeResponse></soapenv:Body></soapenv:Envelope>
```

2.5.3 Troubleshooting the configuration

To troubleshoot potential problems in your configuration, use the following guidance:

1. Since this is the first time you are invoking this service on the device, enable the Probe facility that shows the details of each request that is going through the service.

   Probe: Probe is an excellent device for debugging problems. However, turn off this device in a production deployment.

2. From the Control Panel, select **Troubleshooting** (Figure 2-41).

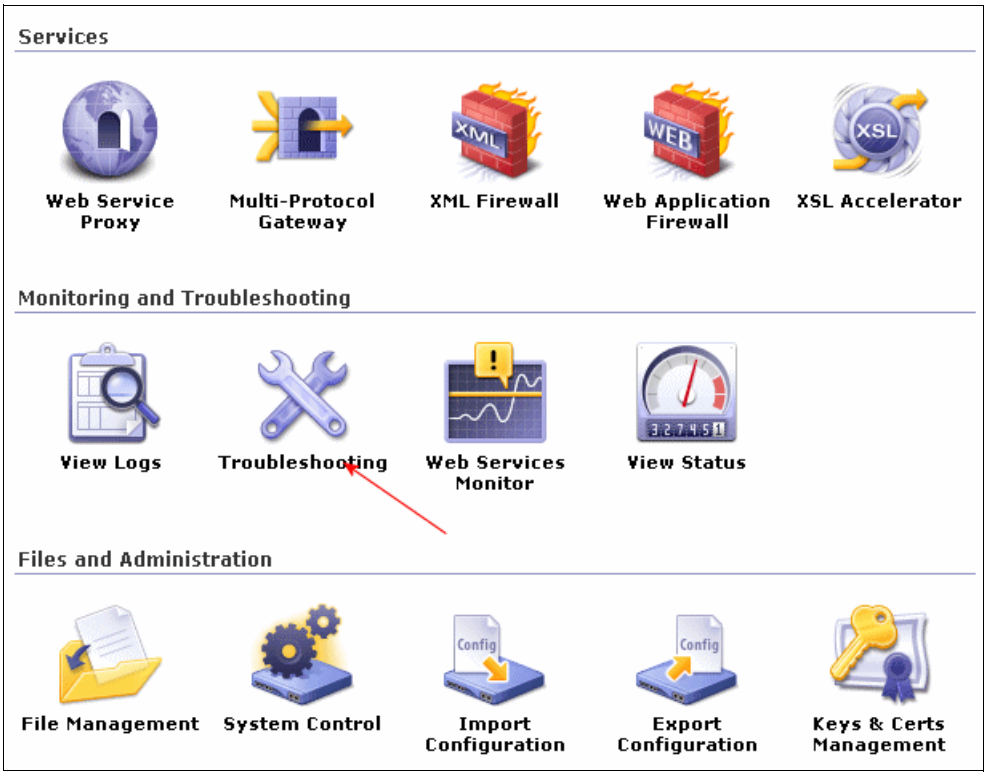


Figure 2-41 Troubleshooting icon

3. On the Troubleshooting Panel page, click the **Probe** tab at the top of the panel (Figure 2-42).

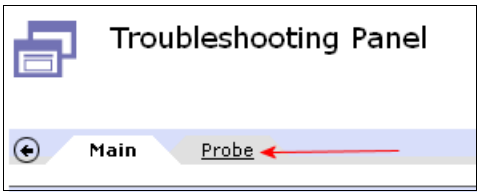


Figure 2-42 Troubleshooting Panel page

4. On the Probe page (Figure 2-43), scroll down until you see Web Service Proxy. From the pull-down list, select **HelloWSProxy**. Click **Add Probe**.

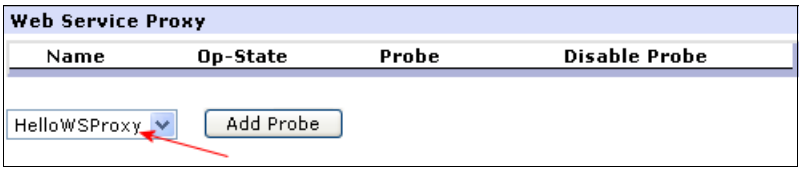


Figure 2-43 Selecting a service

5. You receive a status message like the one shown in Figure 2-44. Click **Close**.

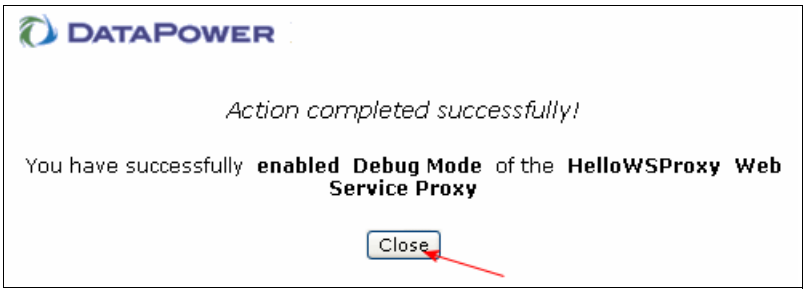


Figure 2-44 Successfully added Probe

6. Return to Web services Testing Tool and invoke the Web service again.
7. After you receive the response, from the Control Panel, navigate to the Troubleshooting panel.
8. For the HelloWSProxy XML Firewall Service, click the **magnifying glass** icon under Probe (Figure 2-45).

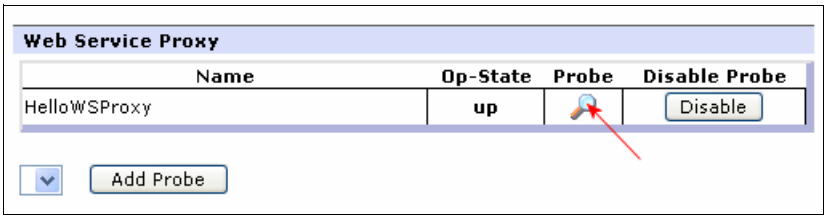


Figure 2-45 Inspecting information collected by the Probe

9. The Transaction list opens for this service. Click the **+** sign next to the magnifying glass icon (Figure 2-46).

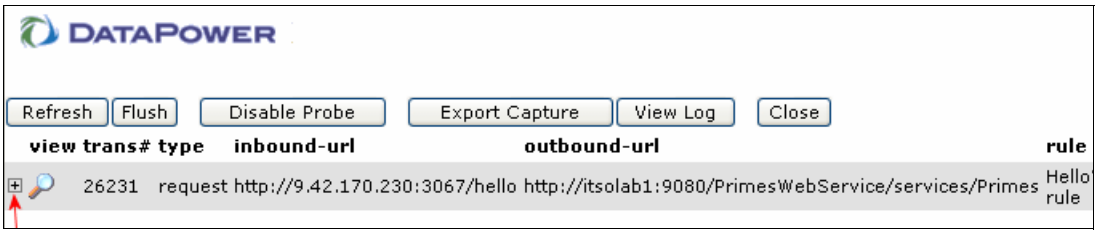


Figure 2-46 Transaction list for the HelloWSProxy service

10. As shown in Figure 2-47, you now see two icons. One is for the information that was collected on the request message sent to the DataPower appliance. The other is for the information that was collected on the response sent back to the client.

    Click the **magnifying glass** icon next to the request.

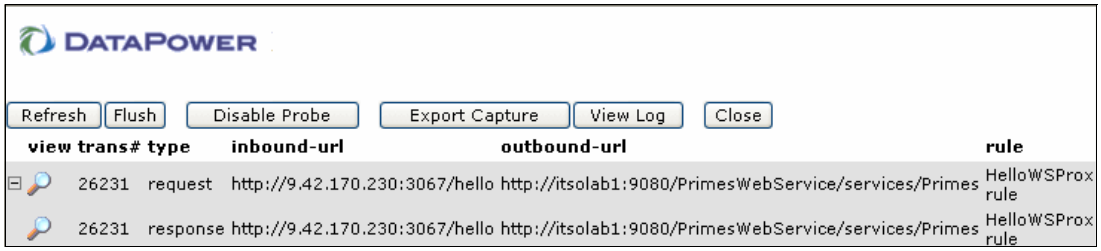


Figure 2-47 Request and Response Information Collected by the Probe

11. In the transaction window (Figure 2-48) that opens, review the entire request message sent to the DataPower appliance. Then close the window.

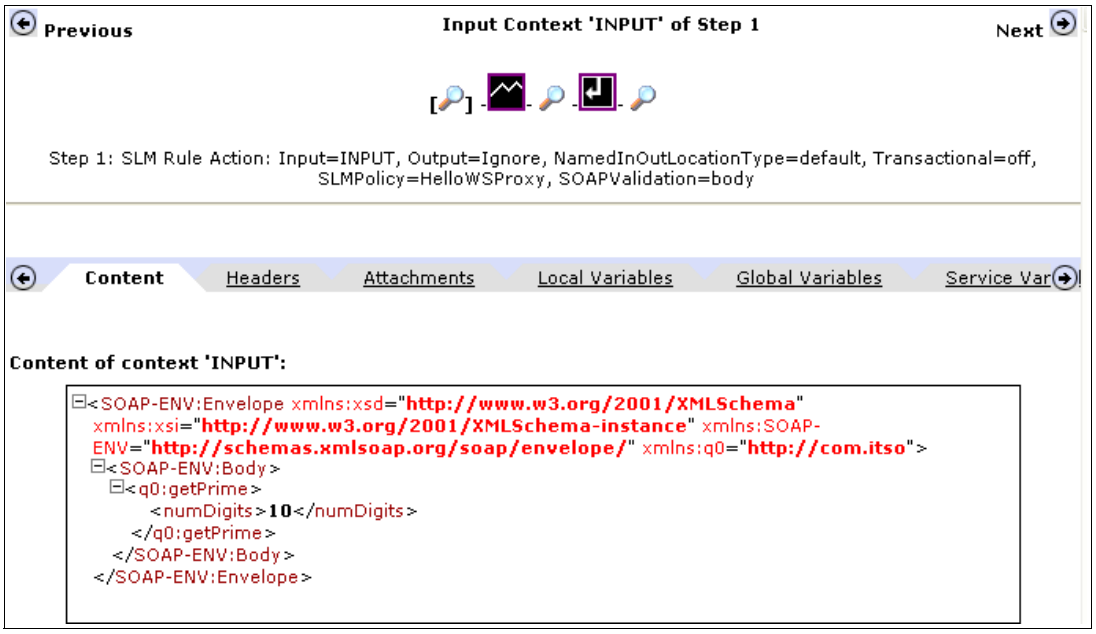


Figure 2-48 Incoming Request Message Sent to the DataPower appliance

12. You return to the Transaction List (Figure 2-47). Click the **magnifying glass** icon next to response.

13. In the transaction window (Figure 2-49) that opens, review the entire response message that was sent to the client. Then close the window.

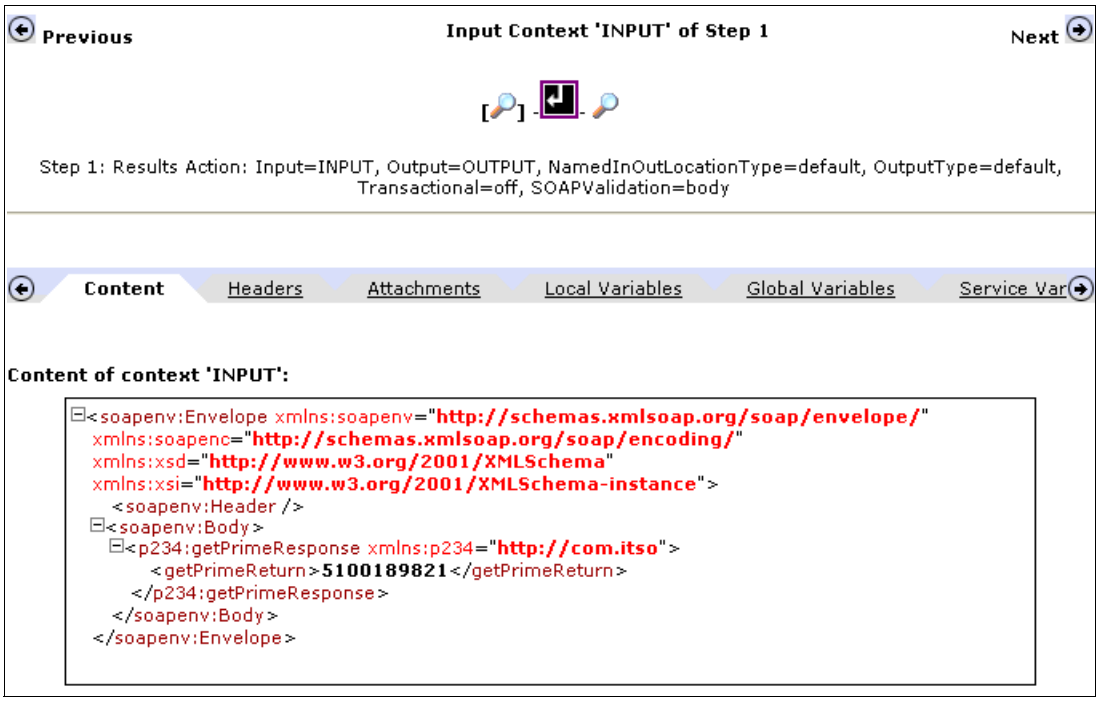


Figure 2-49 Outbound response sent to the client from the DataPower appliance

14. Click **Disable Probe** (Figure 2-50).

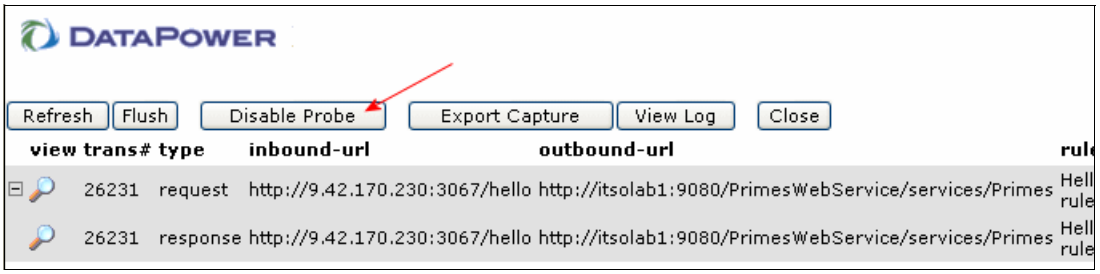


Figure 2-50 Disable Probe

15. You receive a status message indicating success. Click **Close Transaction List**.
16. Close the window.
17. Click the icon in the upper right corner of the WebGUI to return you to the Control Panel (Figure 2-51).

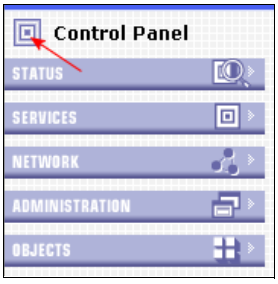


Figure 2-51 Returning to the Control Panel

You have now completed the troubleshooting task.

System logs: As an alternative, you can look for debugging information is the system logs. On the navigation bar, click **STATUS** → **System Logs**. On the system logs, you only see messages for the services that are defined in that domain. However, the default domain can see messages for all domains.

2.6 Summary

In this chapter, we demonstrated how quickly and easily you can configure a DataPower device. We also explained how to create two simple services, an XML Firewall Service and a Web Service Proxy Service. Later in this paper, we explain how to add enhancements (such as security), perform integration with back ends (such as MQ), and do complex processing by using XML Stylesheet Language Transformations.

3

Enabling existing applications

An important aspect of service-oriented architecture (SOA) enablement is the challenge of integrating existing applications into the SOA environment, for example:

- Reducing the impact to back-end applications
- Using Web and Web Services protocols
- Enabling security
- Integrating with other services

These issues and others are leading to the need for an enterprise service bus (ESB) in SOA. In this chapter, we build a core scenario for providing a Web Service facade to an existing COBOL application. We incrementally expand this scenario in the following chapters to build fully secure access to this sample application.

3.1 The enterprise service bus

Successful implementation of an SOA requires applications and an infrastructure that can support the SOA principles. Applications can be enabled for SOA by creating service interfaces to existing or new functions. The service interfaces should be accessed by using an infrastructure that can route and transport service requests to the correct service provider. As organizations expose more and more functions as services, it is vitally important that this infrastructure should support the management of SOA on an enterprise scale.

The ESB is a middleware infrastructure component that supports the implementation of SOA within an enterprise. Figure 3-1 illustrates where the ESB fits in the SOA reference architecture.

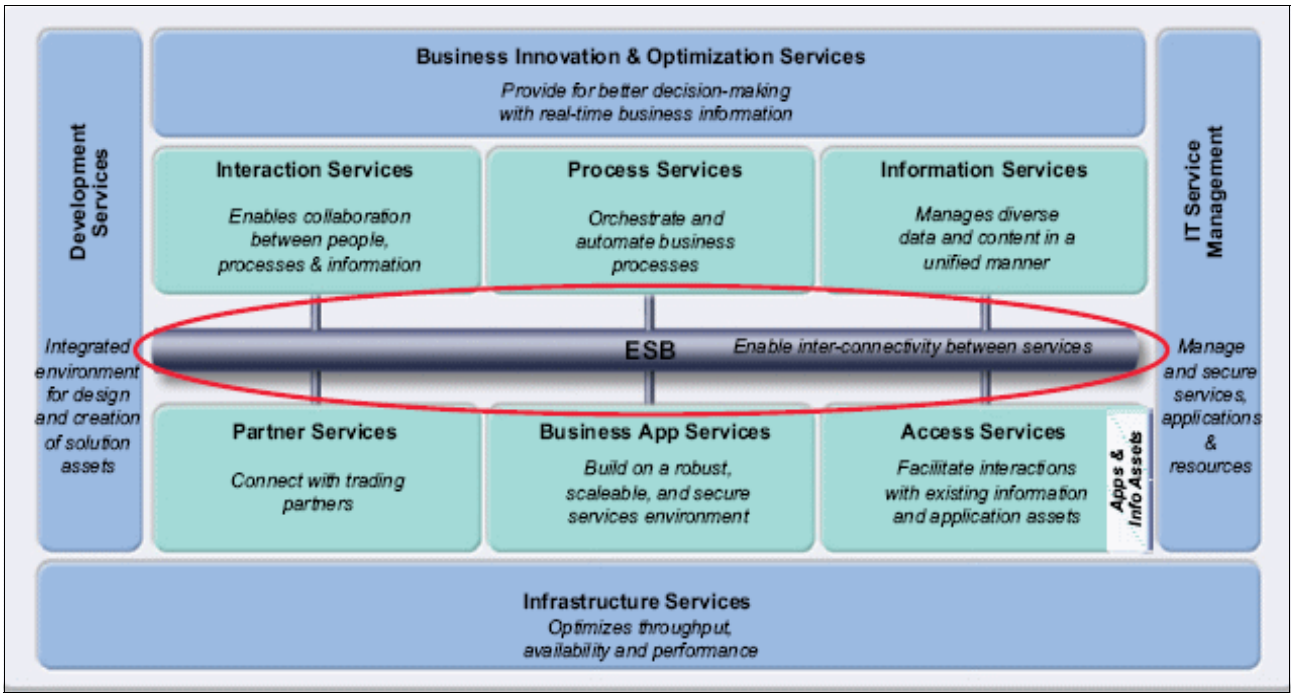


Figure 3-1 IBM reference architecture

You can see the need for an ESB by considering how it supports the concepts of SOA implementation, which is in the following ways:

- Decoupling the consumer's view of a service from the actual implementation of the service
- Decoupling technical aspects of service interactions
- Integrating and managing services in the enterprise

Decoupling the consumer's view of a service from the actual implementation greatly increases the flexibility of the architecture. One service provider can be substituted for another, for example, because another provider offers the same services for lower cost or with higher standards. This can occur without the consumer being aware of the change or without the need to alter the architecture to support the substitution.

This decoupling is better achieved by having the consumers and providers interact via an *intermediary*. Intermediaries publish services to consumers. The consumer binds to the intermediary to access the service, with no direct coupling to the actual provider of the service. The intermediary maps the request to the location of the real service implementation.

In an SOA, services are described as being loosely coupled. However, at implementation time, there is no way to loosely couple a service or any other interaction between systems.

The systems must have a common understanding to conduct an interaction. To achieve the benefits of loose coupling, consideration should be given to how to couple or decouple various aspects of service interactions. Examples include the platform and language in which services are implemented, the communication protocols used to invoke services, or the data formats used to exchange input and output data between service consumers and providers.

Further decoupling can be achieved by handling the technical aspects of transactions outside of applications. This can apply to the following aspects of interactions among others:

- How service interactions are secured
- How the integrity of business transactions and data is maintained, for example, through reliable messaging, the use of transaction monitors, or compensation techniques
- How the invocation of alternative service providers is handled in the event that the default provider is unavailable

These aspects imply a need for middleware to support an SOA implementation. The following functions might be provided by the middleware among others:

- Map service requests from one protocol and address to another
- Transform data formats
- Support a variety of security and transactional models between service consumers and service providers and recognize that consumers and providers might support or require different models
- Aggregate or disaggregate service requests and responses
- Support communication protocols between multiple platforms with appropriate qualities of service
- Provide messaging capabilities, such as message correlation and publish/subscribe, to support different messaging models such as events and asynchronous request/response

This middleware support is the role of an ESB.

3.1.1 Definition of an enterprise service bus

An ESB provides an infrastructure that removes any direct connection between service consumers and providers. Consumers connect to the bus and not the provider that implements the service. This type of connection further decouples the consumer from the provider. A bus implements further value-add capabilities. For example, security and delivery assurance can be implemented centrally within the bus instead of having this buried within applications. Integrating and managing services in the enterprise outside of the actual implementation of the services in this way helps to increase the flexibility and manageability of an SOA.

The primary driver for an ESB, however, is that it increases decoupling between service consumers and providers. Protocols, such as Web services, define a standard way of describing the interface to a service provider that allows some level of decoupling (because the actual implementation details are hidden). However, the protocols imply a direct connection between the consumer and provider.

Although it is relatively straight forward to build a direct link between a consumer and provider, these links can lead to an interaction pattern that consists of building multiple point-to-point links that perform specific interactions. With a large number of interfaces, this quickly leads to

the build up of complex spaghetti links with multiple security and transaction models. Routing control is distributed throughout the infrastructure, and probably no consistent approach to logging, monitoring, or systems management is implemented. This environment is difficult to manage or maintain and inhibits change.

A common approach to reducing this complexity is to introduce a centralized point through which interactions are routed, called a *hub-and-spoke architecture*, as shown in Figure 3-2.

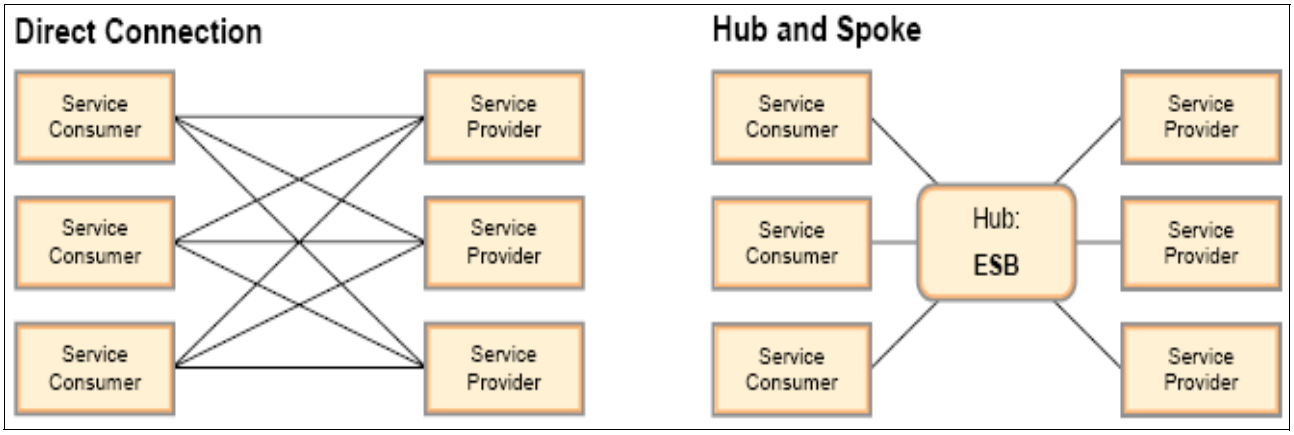


Figure 3-2 Direct connection versus hub-and-spoke connection

A hub-and-spoke architecture is a common approach that is used in application integration architectures. In a hub, the distribution rules are separated from applications. The applications connect to the hub and not directly to any other application. With this type of connection, a single interaction from an application can be distributed to multiple target applications without the consumer being aware that multiple providers are involved in servicing the request. This connection can reduce the proliferation of point-to-point connections.

The benefit of reducing the number of connections truly emerges if the application interfaces and connections are genuinely reusable. For example, consider an application that needs to send data to three other applications. If it is implemented in a hub, the sending application must define a link to the hub, and the hub must have links that are defined to the three receiving applications, for a total of four interfaces that must be defined.

If we implement the same scenario by using multiple point-to-point links, the sending application must define links to each of the three receiving applications, for a total of three links. A hub offers the benefit of reduced links only if another application must also send data to the receiving applications and can use the same links as those that are defined for the first application. In this scenario, the new application must define a connection between itself and the hub, which can then send the data correctly formatted to the receiving applications.

Hubs can be federated together to form what is logically a single entity that provides a single point of control, but is actually a collection of physically distributed components. This is commonly termed a *bus*. A bus provides a consistent management and administration approach to a distributed integration infrastructure.

3.1.2 Enterprise requirements for an enterprise service bus

The use of a bus to implement an SOA has several advantages. In an SOA, by definition, services should be reusable by a number of different consumers to achieve the benefits of reduced connections. In addition, the ESB must offer the following support:

- High volumes of individual interactions
- More established integration styles, such as message-oriented and event-driven integration, to extend the reach of the SOA

  The ESB should allow applications to be SOA-enabled either directly or by using adapters.
- Centralization of enterprise-level qualities of service and manageability requirements into the hub

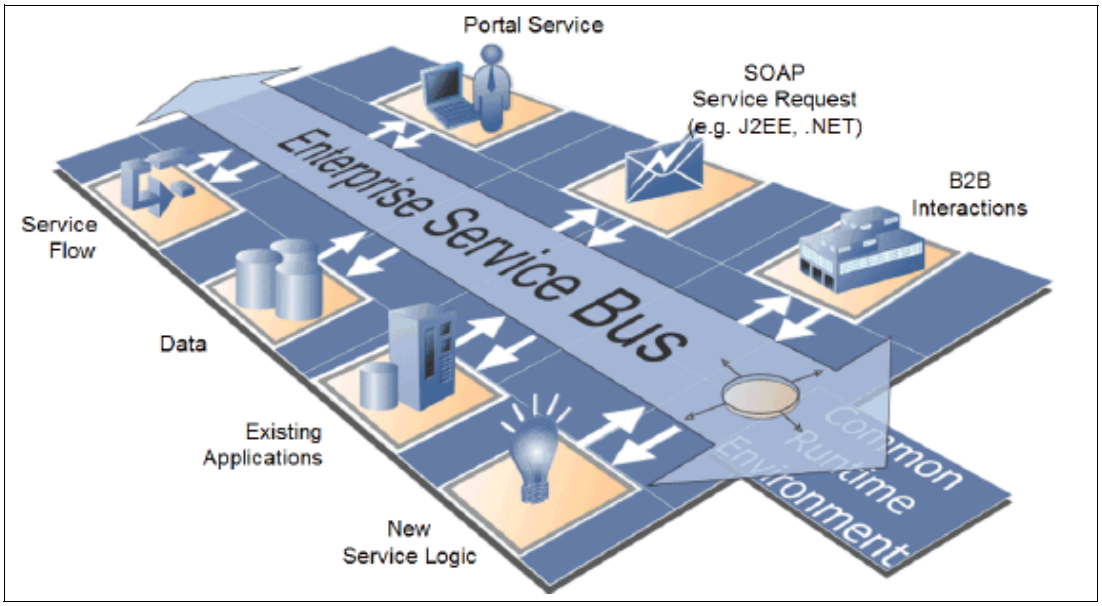


Figure 3-3 High-level view of the ESB

SOA applications are built from services. Typically, a business service relies on many other services in its implementation. The ESB component provides access to the services, so that it enables the building of SOA applications.

Mediation support

The ESB is more than just a transport layer. It must provide mediation support to facilitate service interactions. For example, it must find services that provide capabilities for which a consumer is asking. Alternatively, it must take care of interface mismatches between consumers and providers that are compatible in terms of their capabilities.

An ESB must support a variety of ways to get on and off the bus, such as adapter support for existing applications or business connections that enable external partners in business-to-business interaction scenarios. To support these different ways to get on and off the bus, it must support service interaction with a wide variety of service endpoints. It is likely that each endpoint will have its own integration techniques, protocols, security models, and so on.

This level of complexity should be hidden from service consumers. Consumers need to be offered a simpler model. In order to hide the complexity from the consumers, the ESB is

required to mediate between the multiple interaction models that are understood by service providers and the simplified view that is provided to consumers.

Protocol independence

Services can be offered by a variety of sources. Without an ESB infrastructure, any service consumer that needs to invoke a service must connect directly to a service provider by using the protocol, transport, and interaction pattern that is used by the provider. With an ESB, the infrastructure shields the consumer from the details of how to connect to the provider.

In an ESB, there is no direct connection between the consumer and provider. Consumers access the ESB to invoke services, and the ESB acts as an intermediary by passing the request to the provider by using the appropriate protocol, transport, and interaction pattern for the provider. With this intermediary connection, the ESB can shield the consumer from the infrastructure details of how to connect to the provider. The ESB should support several integration mechanisms, all of which can be described as invoking services through specific addresses and protocols. This should happen even if, in some cases, the address is the name of a CICS transaction and the protocol is a J2EE resource adapter integrating with the CICS Transaction Gateway. By using the ESB, the consumers are unaware of how the service is invoked on the provider.

Because the ESB removes the direct connection between service consumers and providers, an ESB enables the substitution of one service implementation by another with no effect to the consumers of that service. Therefore, by using an ESB, the reach of an SOA can extend to non-SOA-enabled service providers. It can also be used to support the migration of non-SOA providers to using an SOA approach without impacting the consumers of the service.

3.2 A sample scenario and components

The corporation ITSOCorp has a heterogeneous environment as shown in Figure 3-4. The company needs to publish an existing application as a Web service application by using the DataPower component.

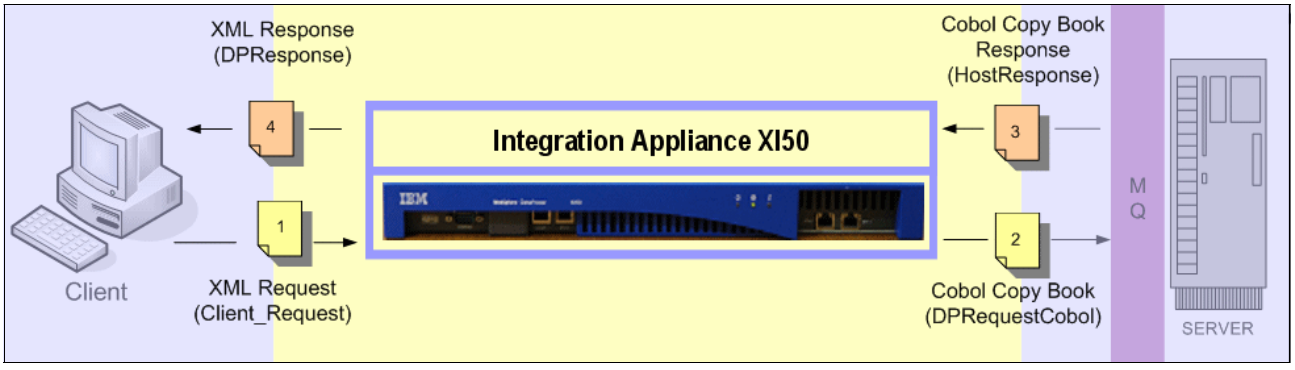


Figure 3-4 Existing application scenario

As shown in Figure 3-4 on page 56, the sample scenario described in this chapter includes the following components:

- Client device

  The client device is responsible for sending a SOAP request message to the DataPower appliance and will receive a XML response message. In this case, we use the cURL tool, which is freely available from the cURL Web site at the following address:

  `http://curl.haxx.se`

- SOAP request message

  This is the message sent by the client to the DataPower appliance.

- DataPower appliance

  This appliance is the main component in the scenario and is responsible for the following tasks:

  - Validating input against the request schema
  - Transforming an incoming SOAP message to a COBOL message
  - Doing a protocol transformation (HTTP to MQ) for connectivity to a back-end application
  - Transforming the response COBOL message back to SOAP
  - Error handling

- COBOL request message

  This request message is sent to the existing application in the COBOL CopyBook format.

- Existing system

  This back-end application processes input COBOL data and returns response COBOL data. In this sample scenario, a C program is used instead of the actual back-end application.

- COBOL response message

  This message is returned by the existing system in the COBOL CopyBook format.

- SOAP response message

  This response message is sent to a client by the DataPower appliance after transforming the COBOL response message to a SOAP message.

Software levels and tools: The following software levels and tools are used in this sample scenario:

- DataPower Firmware v3.6.0.17 or later
- WebSphere MQ v6.0
- The cURL tool
- A C program called redbookmqserver.exe that is supplied with the additional materials of this paper to simulate the back-end COBOL application (For details, see Appendix B, “Additional material” on page 183.)

3.3 Transformations

In this section, we use WebSphere Transformation Extender to generate the files that are required by this scenario to support the request and response transformations in the DataPower appliance.

Mapping artifacts: Completed mapping artifacts are provided in the additional materials that are associated to this paper. For details, see Appendix B, “Additional material” on page 183.

3.3.1 WebSphere Transformation Extender basics

WebSphere Transformation Extender is a powerful, transaction-oriented, data integration solution. It automates the transformation of high-volume, complex transactions without the need for manual coding. It provides enterprises with a quick return on investment (ROI). This product supports electronic data interchange (EDI), XML, SWIFT, HIPAA and other standards-based business-to-business integration. In addition, it supports the real-time integration of data from multiple applications, databases, messaging middleware, and communications technologies across the enterprise.

More information: You can find information about WebSphere Transformation Extender on the Web at the following address:

`http://www-306.ibm.com/software/integration/wdatastagetx/library/index.html`

For our purposes, we use DataPower support in the WebSphere Transformation Extender Studio to achieve the following tasks:

- Create COBOL-XML mappings
- Test the mappings
- Generate the transformation for the DataPower appliance

The following tools come with WebSphere Transformation Extender Studio:

- *Type Designer* is used to create “type tree” representation of message types. In our case, we have two XML message types and two COBOL message types. The XML message types are specified by an XML Schema Definition (XSD), and the COBOL message types are specified by two copy books. The Type Designer can import XSD and copy book files to generate the type trees.
- *Map Designer* uses the type trees to map an input type tree to an output type tree. Mapping is done on individual elements of the input and output tree elements. Each mapping is specified by a rule. Most of the rules in our scenario are generated by dragging from the input tree to the output tree. Occasionally we must modify the rule manually, for example to affect type conversion or check for validity.

In the remainder of this section, we explain how to map a request XML message to a COBOL message type and how to map a response COBOL message type to an XML data type.

3.3.2 Creating a type tree with Type Designer

The Type Designer tool is used to define, modify, and view type trees. A type tree describes the syntax, structure, and semantics of data. The *syntax* of the data refers to its format including tags, delimiters, terminators, and other characters that separate or identify sections

of data. The *structure* of the data refers to its composition including repeating substructures and nested groupings. The *semantics* of the data refer to the meaning of the data including rules for data values, relationships among parts of a large data object, and error detection and recovery.

For more information about Type Designer, refer to the Type Designer document at the following address:

`ftp://ftp.software.ibm.com/software/websphere/integration/wdatastagetx/1003.pdf`

To create a type tree:

1. Create a directory called `FilesWTX` and copy the files shown in Table 3-1 into this directory.

Table 3-1 Files used for creating the type trees

File	Function
CCOUT4K.cpy	Host Response copybook
CCINP.cpy	Host request copybook
DataTypes.xsd, PayloadTypes.xsd, SOAAssureService.xsd	Client request and response XSD

2. Open the Type Designer application. Select **Start** → **Programs** → **IBM WebSphere Transformation Extender 8.1** → **Design Studio** → **Type Designer**.
3. From the toolbar, select **Tree** → **Import**.
4. In the Import from window of the Importer Wizard (Figure 3-5), select **XML Schema** and click **Next**.

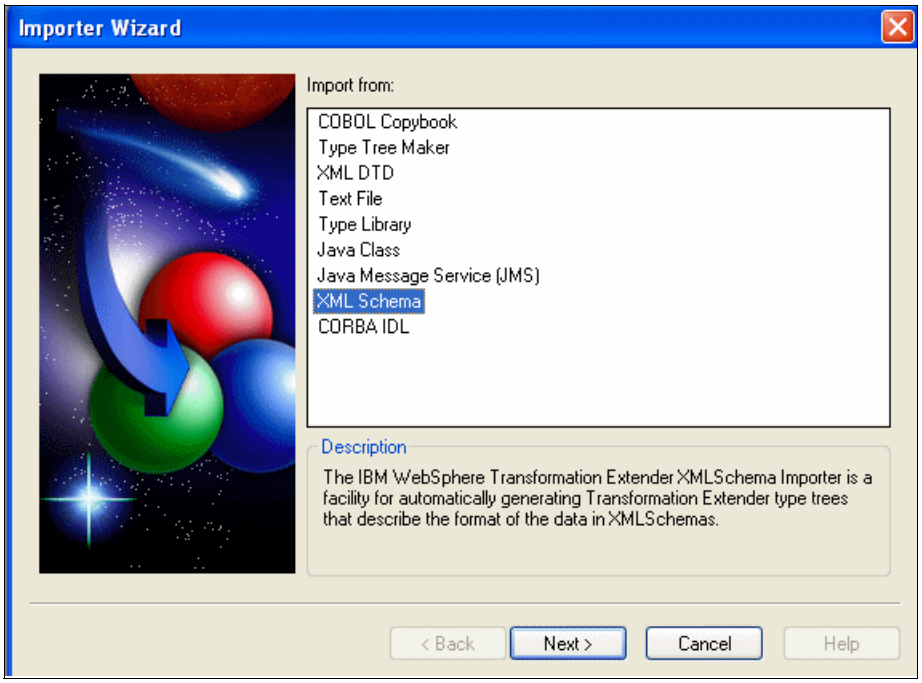


Figure 3-5 Importer Wizard

5. In the next window, click the **Browse** button to navigate to the schema file, SOAAssureService.xsd, in the C:\FilesWTX\ directory. Select **SOAAssureService.xsd**.

   Note: The SOAAssureService.xsd file refers to the DataType.xsd and PayloadType.xsd files. Therefore, all three files should be in the same folder.

   Click **Next**.

6. In the next window, National Language uses the default of **Western**. Click **Next**.
7. In the XML Schema window (Figure 3-6), for File Name, change the type-tree destination file to `C:\WTXWork\SOAAssureService.mtt` and click **Next**.

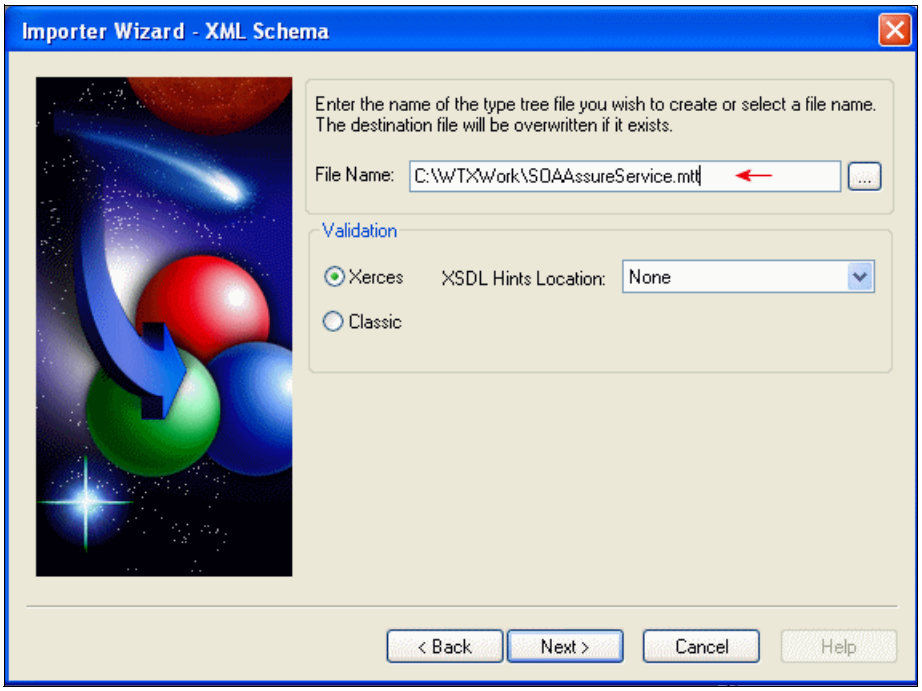


Figure 3-6 Type tree destination file

8. The importer builds the type tree. Click **Finish** (Figure 3-7).

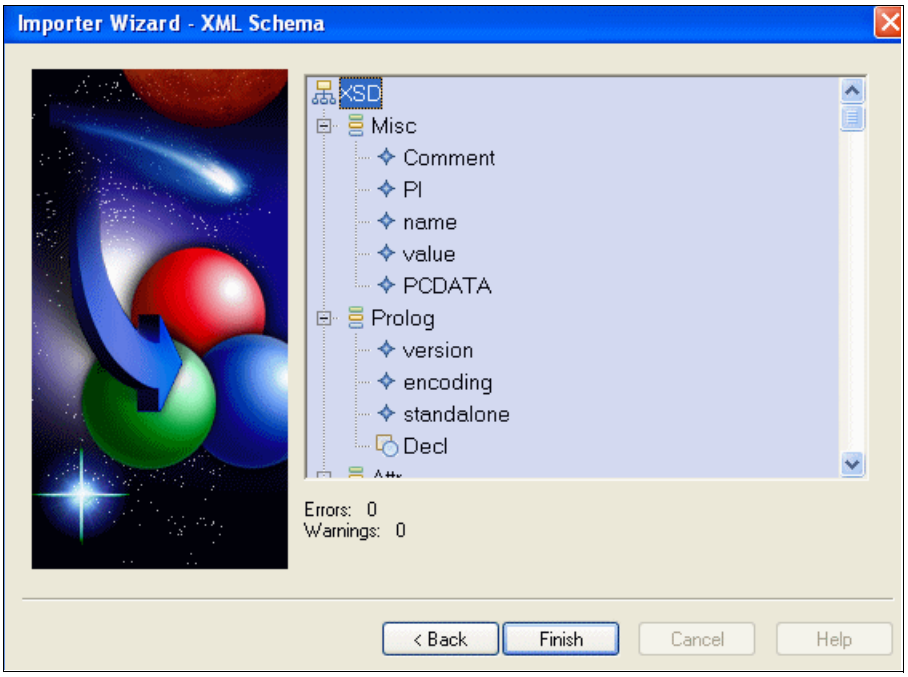


Figure 3-7 Type tree definition

9. When you see the message that prompts you about opening the newly generated type tree, click **Yes**. Figure 3-8 shows the new type tree.

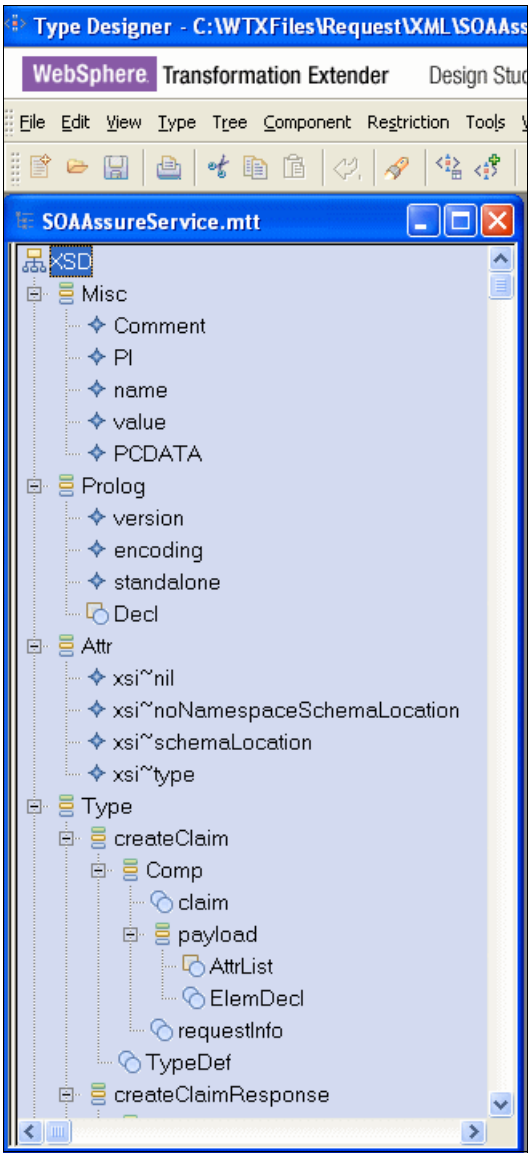


Figure 3-8 SOAAssureService.mtt

10. Create the request COBOL type tree file. From the toolbar, select **Tree** → **Import**.
11. Click **Next**.
12. In the Importer Wizard window, select **COBOL Copy Book** and click **Next**.
13. In the next window, click the **Browse** button, navigate to the C:\FilesWTX\ directory, and select the COBOL file **CCINP.cpy**. Click **Next**.
14. In the Importer Wizard-COBOL Copybook window, you see a check box indicating that you can generate a type tree for a CICS type tree. Since we are not generating a tree for CICS, do not select the check box.

    Select **EBCDIC** from the character-set list and **BIG ENDIAN** from the Byte order list. Click **Next**.

15. In the COBOL Copybook window (Figure 3-9), for File Name, change the type tree destination to `C:\WTXWork\CCINP.mtt`. Click **Next**.

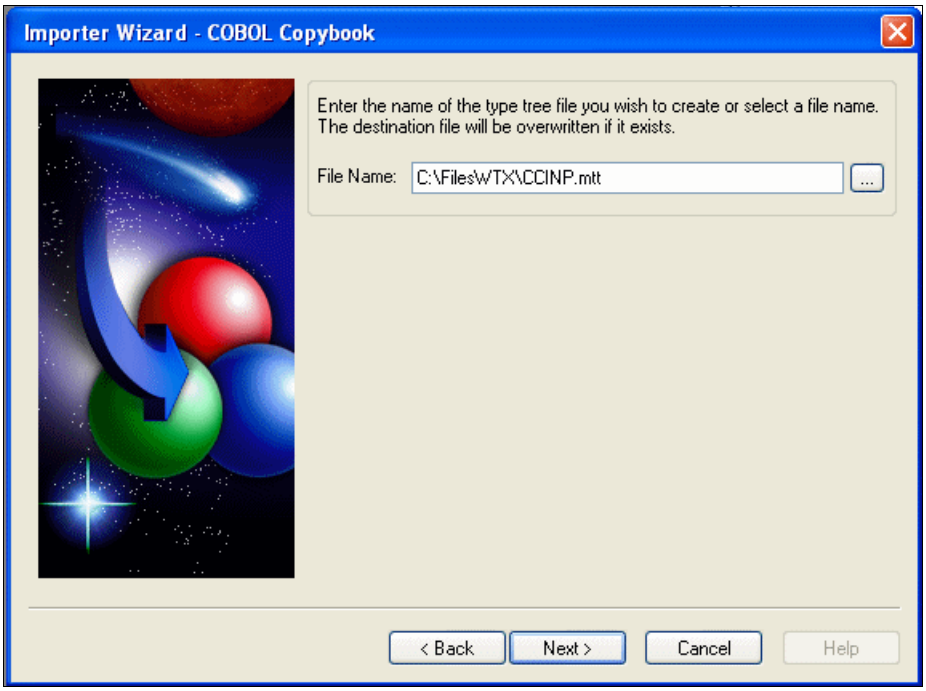


Figure 3-9 COBOL type tree destination file

16. Click **Next**. The importer builds the Type Tree.

17. Click **Finish**. As a result of the steps, the CCINP.mtt and SOAAssureService.mtt files (shown in Figure 3-10) should be in the C:\WTXWork directory. We use these files for the request mapping process in 3.3.3, “Mapping an input type tree to an output type tree by using Map Designer” on page 65.

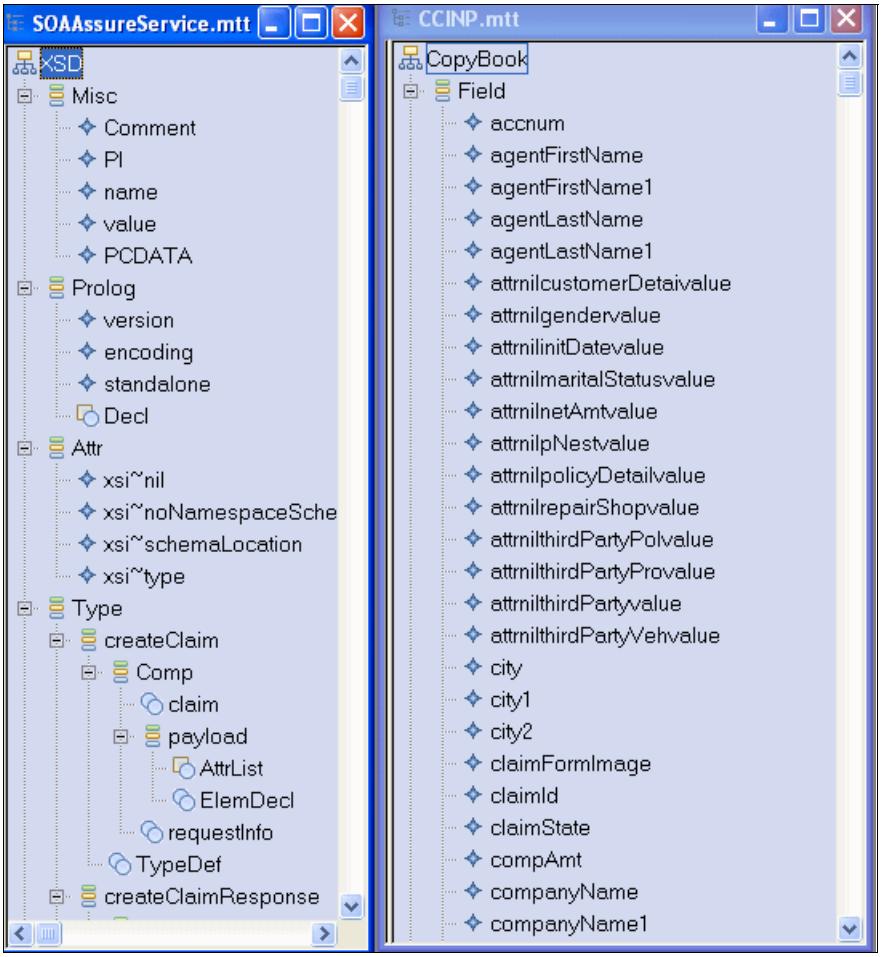


Figure 3-10 XSD and COBOL message definition

18. Create the response COBOL type tree file. From the toolbar, select **Tree** → **Import**.
19. Click **Next**.
20. In the Importer Wizard window that opens, select **COBOL Copy Book** and click **Next**.
21. In the next window, click the **Browse** button, navigate to the COBOL file in the C:\FilesWTX directory, and select **CCOUT4K.cpy**. Then click **Next**.
22. In the Importer Wizard-COBOL Copybook window that opens, do not select the check box for generating a type tree for a CICS type tree, because we do not generate a tree for CICS.

    From the character-set list, select **EBCDIC**, and from the byte order list, select **BIG ENDIAN**. Click **Next**.
23. In the next window. Change the Type tree definition to `C:\WTXWork\CCOUT4K.mtt`. Click **Next**.
24. Click **Finish**.

25. One of properties of the elements of the type tree is to specify the characters that will be used to indicate no data. For example, the default is blanks for the string element type. Because we are not modifying the defaults, we can make DataPower run time more efficient by not using this property altogether. To clear this property:

    a. From the type tree, select **CopyBook** → **Field**.

    b. Right-click and select **Properties**.

    c. Select **Item Subclass** → **None** → **Special value**. In the Value field, erase any characters.

    d. Select **Item Subclass** → **None** → **Required on input**. In the Value field, choose **No**.

    e. Right-click **Item Subclass** → **None** → **Special value**. Then select **Propagate**.

    As a result of these steps, you should have a CCOUT4K.mtt type tree created in the WTXWork directory. We use the CCOUT4K.mtt and SOAAssureService.mtt files for the response mapping process in 3.3.3, “Mapping an input type tree to an output type tree by using Map Designer” on page 65.

26. Exit the Type Tree tool.

3.3.3 Mapping an input type tree to an output type tree by using Map Designer

In this section, we explain how to map an input type tree to an output type tree by using the Map Designer.

Request mapping

Map Designer is a client component of the Design Studio that you use to specify rules for data transformation from input to output type trees. Maps can be built for a specific platform and then run on that platform to perform the transformation of the data.

More information: You can find more information about Map Designer on the Web at the following address:

`ftp://ftp.software.ibm.com/software/websphere/integration/wsdatastagetx/1005.pdf`

Follow these steps:

1. Open the Map Designer application. Click **Start** → **Programs** → **IBM WebSphere Transformation Extender 8.1** → **Design Studio** → **Map Designer**.
2. From the Startup window (Figure 3-11), select **Create a new map source file** and click **OK**.

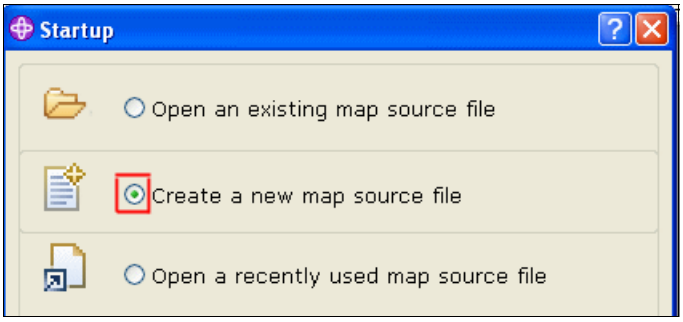


Figure 3-11 Creating a new map source file

3. In the Save As window (Figure 3-12), select the **WTXWork** directory and in the File name field, type `SOABenchCreateClaim` for the new mapping. Click **Save**.

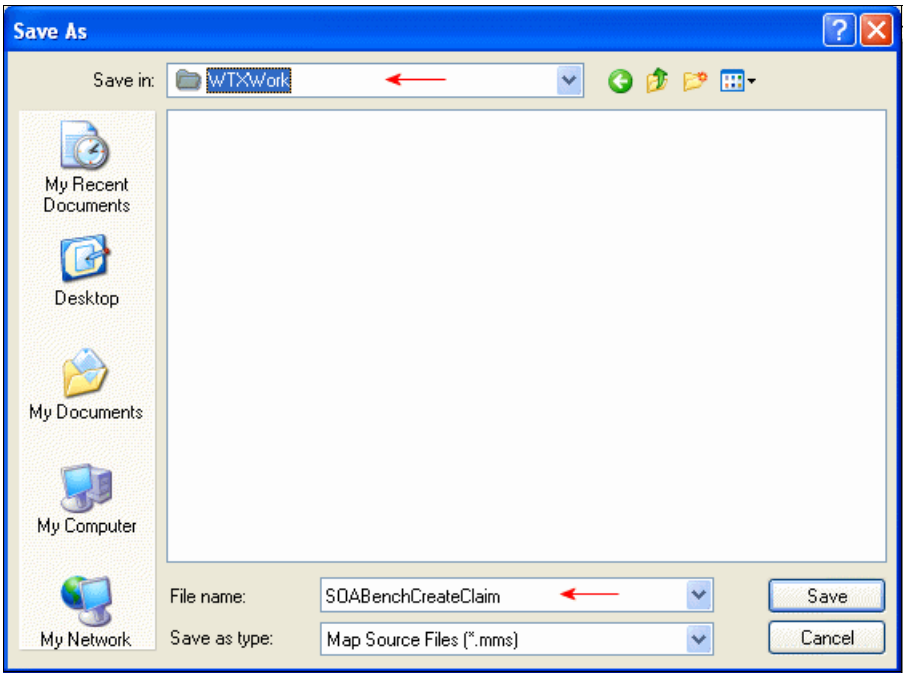


Figure 3-12 Create SOABenchCreateClaim

4. The Design Studio application shows the new map source with the name SOABenchCreateClaim with the From and To windows (Figure 3-13). Right-click **SOABenchCreateClaim** and select **New**.

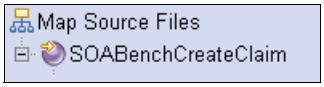


Figure 3-13 New map source file

5. In the Create New Map window (Figure 3-14), for New map name, type `XmlToCobolRequestMap` and click **OK**.

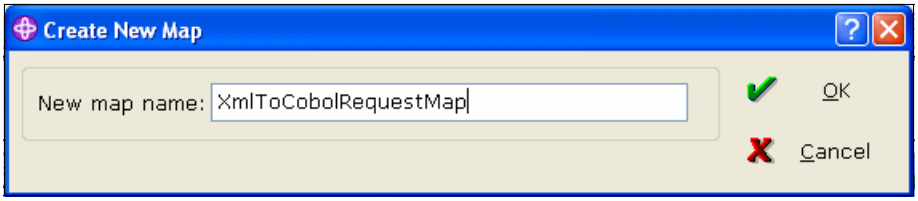


Figure 3-14 New map name

6. Expand **SOABenchCreateClaim** → **XmlToCobolRequestMap**
7. Right-click **Input Cards** and select **New**.

8. In the Add Input Card window, enter the values that are shown in Table 3-2.

Table 3-2 Input card values

Field	Value	Explanation
CardName	XMLInput	Enter any name to identify the input card.
TypeTree	SOAAssureService.mtt	Select the file generated by the Type Designer for the XML type in the WTXWork folder.
Type	Doc XSD	Browse for the DOC type. This is the last option in the XSD attributes list.
Metadata	SOAAssureService.xsd	Select the XSD file from FilesWTX path.
FilePath	Path to WTXTest_RequestSide_Inp.xml	Select the XML file to test the transformation in the WebSphere Transformation Extender Studio. The next section contains details about this file.

Figure 3-5 shows the XMLInput card.

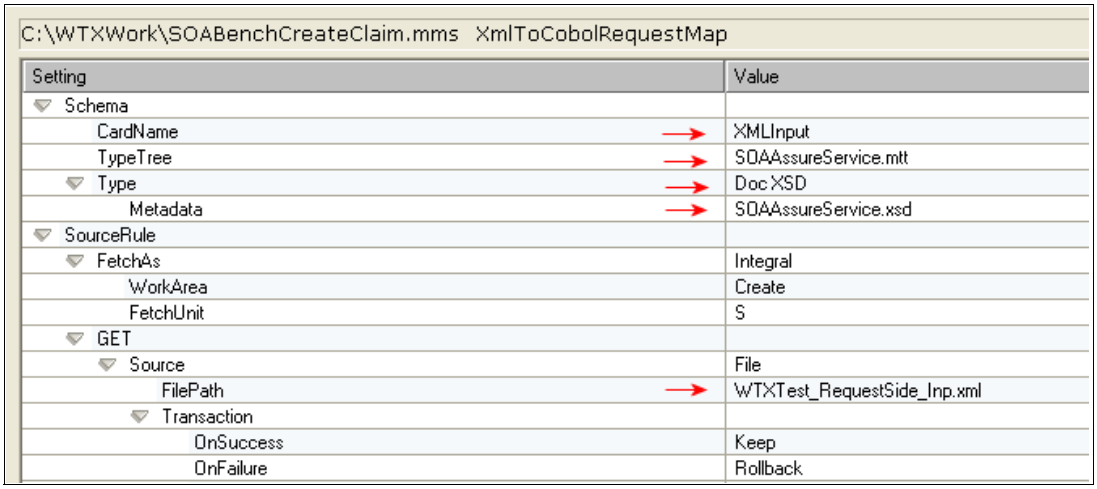


Figure 3-15 XMLInput xard

Click **OK**.

9. Expand **Input Card**. The new input card is displayed. The Input window shows the values for the input message.
10. Right-click **Output Cards** and select **New**.
11. In the next window, enter the values shown in Table 3-3.

Table 3-3 Output card values

Field	Value	Explanation
CardName	COBOLOut	Enter any name to identify the output card.
TypeTree	CCINP.mtt	Select the file generated by the Type Designer for the COBOL type from the WTXWork folder.

Field	Value	Explanation
Type	IN01 Record CopyBook	Browse for the IN01 type. This is the last option in the Copy Book attributes list.
FilePath	Path to WTXTest_RequestSide_OUT.dat	Enter the path and file name to test the result of the transformation. The next section contains details about this file.

Figure 3-16 shows the CobolOUT card.

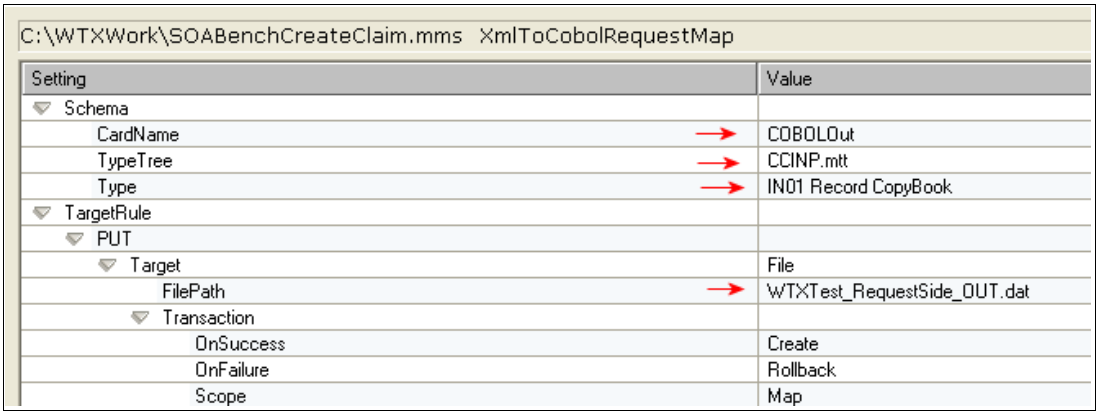


Figure 3-16 CobolOUT card

12. As shown in Figure 3-17, both message definitions are displayed for the source and target to execute the mapping task.

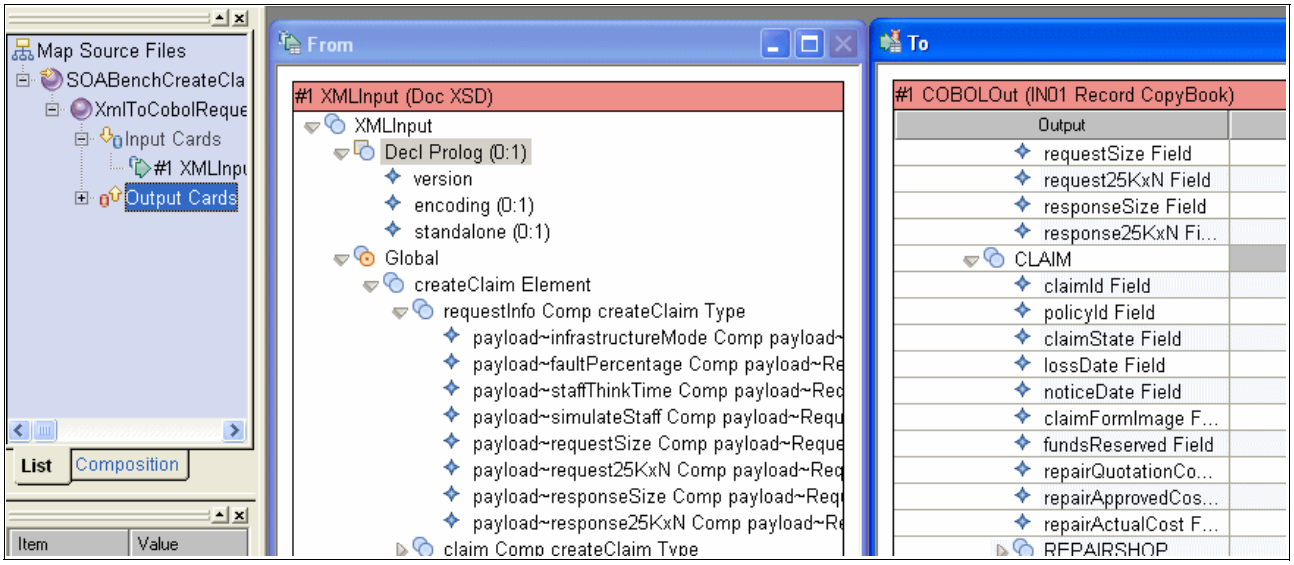


Figure 3-17 From and To message ready for mapping

13. Drag each field for the mapping task. The names on the input and output types should be sufficient to guide the mapping process. You can check the mapping at any point by right-clicking **XMLToCobolRequestMap** and selecting **Build** (Figure 3-18), which generates an error and warning list.

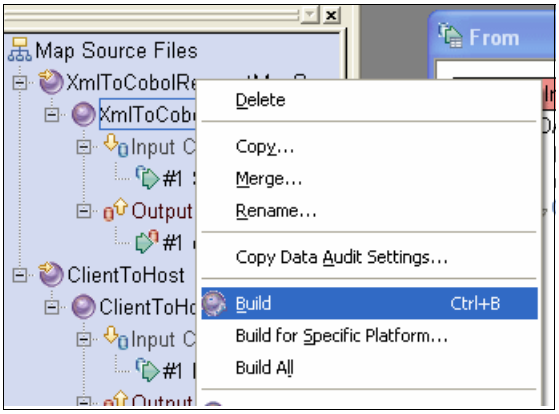


Figure 3-18 Build process

Figure 3-19 shows a sample list of errors and warnings.

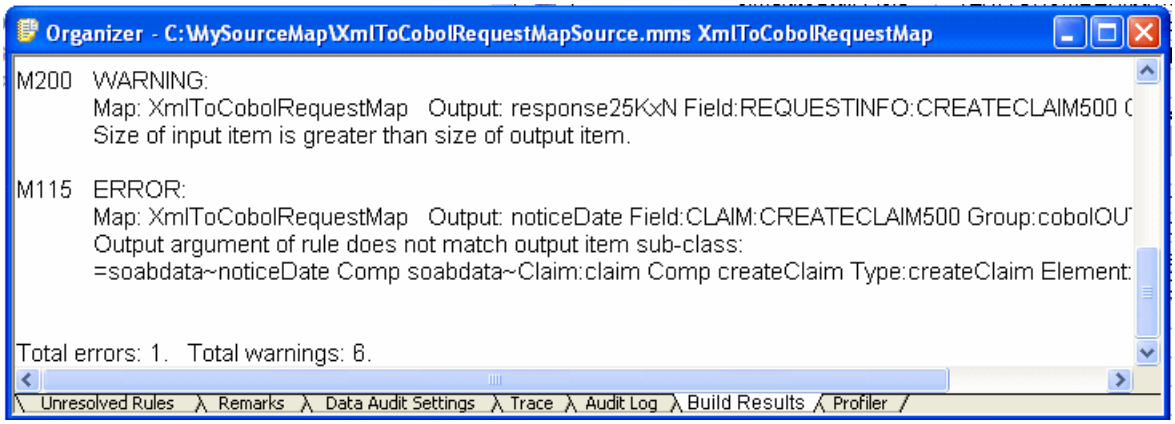


Figure 3-19 Error and warning list

14. Many errors can by corrected by manipulating the rules. For example, for an error to attempt to directly map a Date type to a String, you can force a type conversion by using the DATETOTEXT function. Select the field with the error, right-click and select **Insert Function** as shown in Figure 3-20.

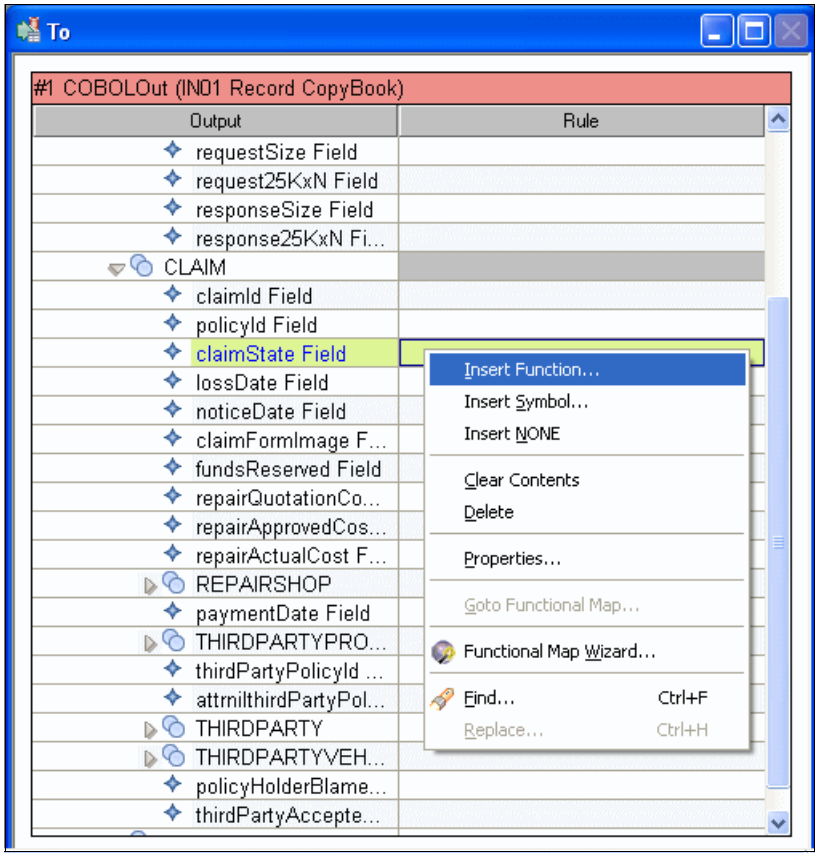


Figure 3-20 Selecting Insert Function

In the category list, select **Conversion**. Then select the correct function and click **Insert**. The final formula must be displayed as follows:

```
=DATETOTEXT(soabdata~noticeDate Comp soabdata~Claim:claim Comp createClaim
Type:createClaim Element:Global:SOAPInput)
```

No mapping rule: Another common error is to not have a rule specified for an output element. To indicate a no mapping rule for an element, right-click the **rule** for the element and select **None**.

15. After completing the mapping, right-click **XmlToCobolRequestMap** and select **Build**. In the final mapping (Figure 3-21), verify that no errors are displayed.

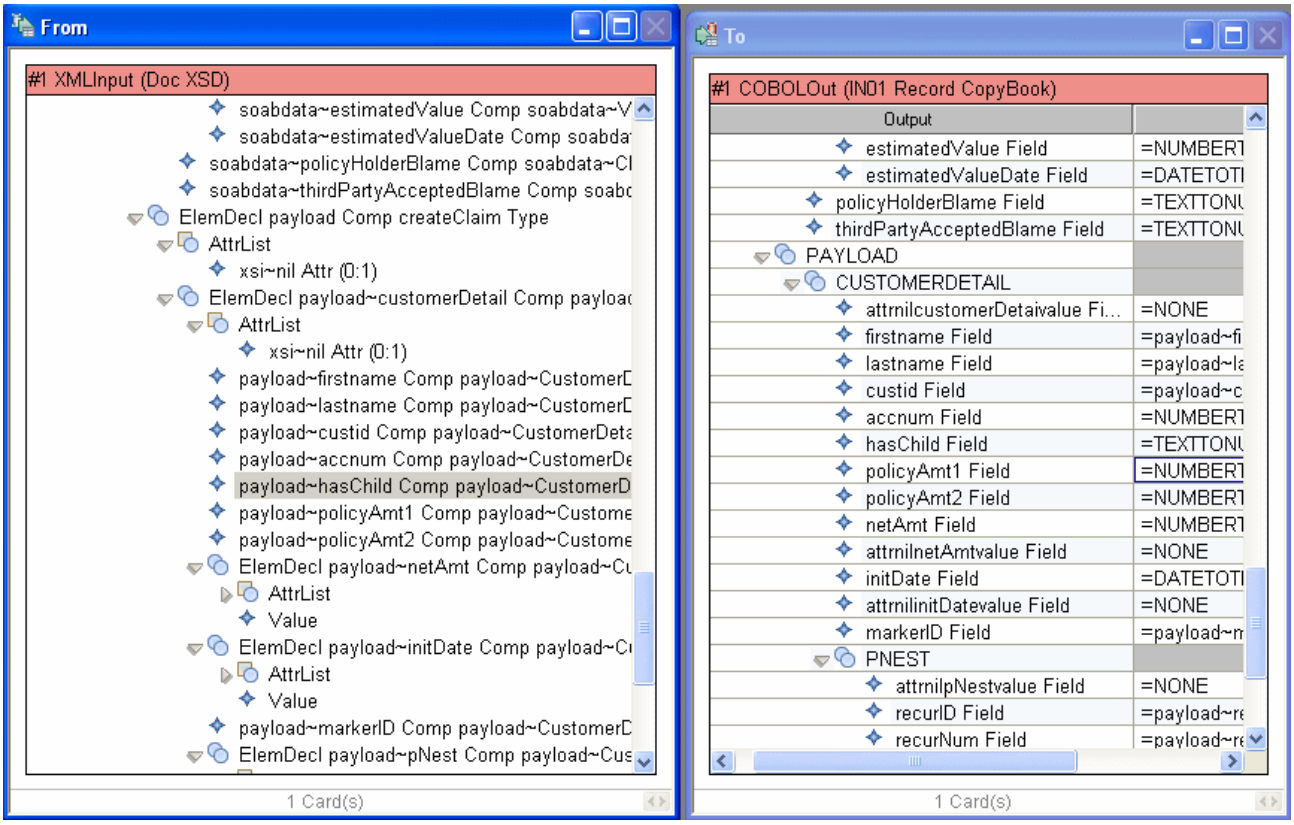


Figure 3-21 Final Mapping

Request map testing

Complete the following steps:

1. Add the “tx” domain to the DataPower appliance.

> **The ‘tx’ domain:** The “tx” domain is a DataPower domain that is supplied by the DataPower appliance for testing maps from the WebSphere Transformation Extender Studio.

2. Create a WTXWork directory. Copy the WTXWork files listed in Table 3-4 into the directory from the additional materials. See Appendix B, “Additional material” on page 183, for more information.

> **Note:** You need this list of all the files for testing both request and response maps.

Table 3-4 WTXWork files

File	Function	Where used
Client_Request.xml	Request data from client	Final deployed scenario
WTXTest_RequestSide_Inp.xml	Request side data for testing WTX and DataPower run times	During development in the WTX Studio

File	Function	Where used
WTXTest_RequestSide_OUT.dat	File to place a transformed request	During development in the WTX Studio
WTXTest_ResponseSide_Inp.dat	Response side data for testing WTX and DataPower run times	During development in the WTX Studio
WTXTest_ResponseSide_OUT.xml	File to place transformed response	During development in the WTX Studio
Host_Response.dat	Response data from host application	Final deployed scenario
converttosoap.xsl	Converting transformed response XML message to SOAP	Final deployed scenario
redbookmqserver.exe	Host server program	Final deployed scenario

Figure 3-22 illustrates the request flow from the WebSphere Transformation Extender Studio to the test firewall in the “tx” domain in the DataPower appliance.

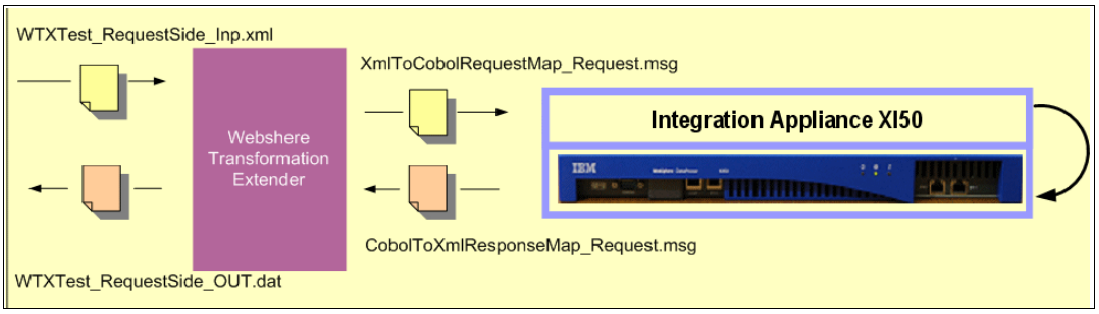


Figure 3-22 Request test with WebSphere Transformation Extender and the DataPower appliance

Figure 3-23 illustrates the response flow from the WebSphere Transformation Extender Studio to the test firewall in the “tx” domain in the DataPower appliance. In the remainder of this section, we take you through setting up and conducting the test.

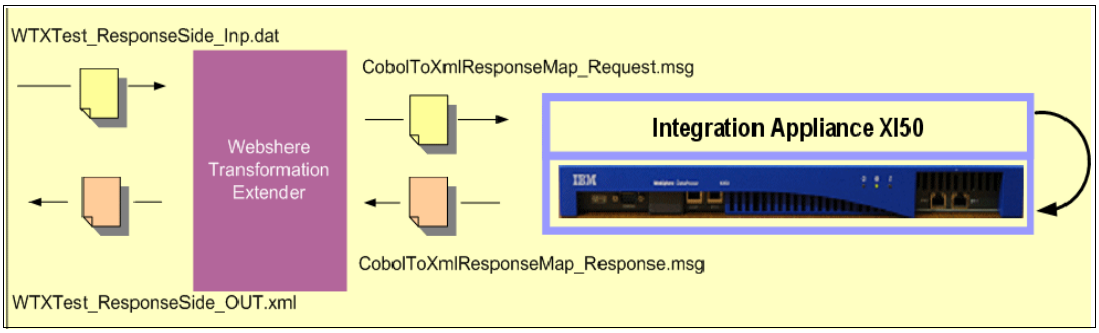


Figure 3-23 Response test with WebSphere Transformation Extender and the DataPower appliance

3. Configure WebSphere Transformation Extender to work with the “tx” domain in the DataPower appliance. Select **Tools** → **Options**.
4. In the Options window (Figure 3-24 on page 73), in the left pane, select **DataPower Maps**. In the right pane, perform the following steps:
   a. In the Host field, type the IP address of the DataPower device.
   b. In the Port field, type 22222, which is the default port value of the “tx-test” firewall in the “tx” domain.

c. In Save Message section, select **Request** and **Response** to save the messages that are sent and received from the DataPower appliance.

d. Click **OK**.

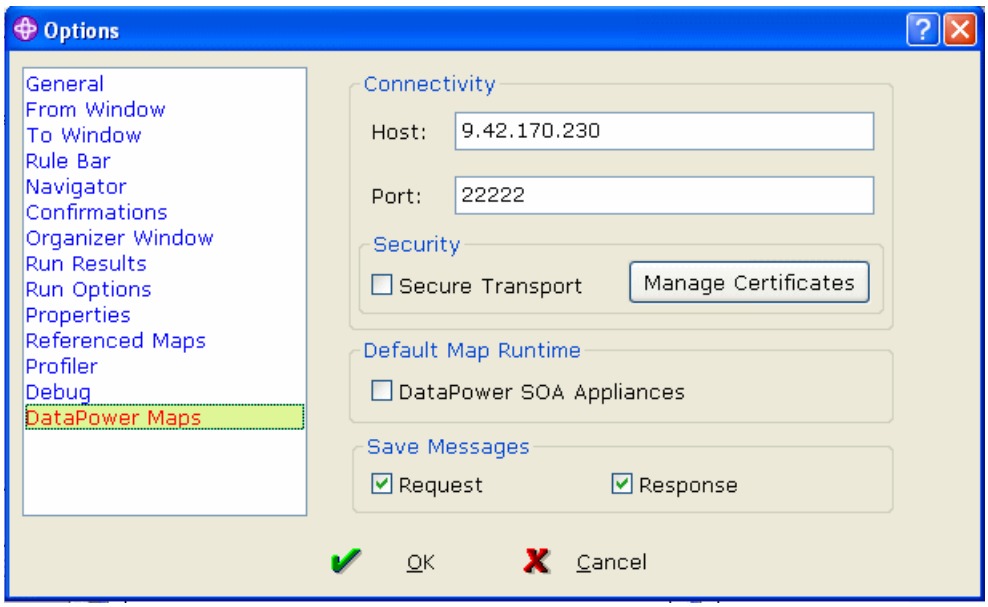


Figure 3-24 DataPower map settings

5. Right-click **XmlToCobolRequestMap** and select **Settings**.

6. In the Map Settings window (Figure 3-25), for MapRuntime, select **WebSphere DataPower** and click **OK**.

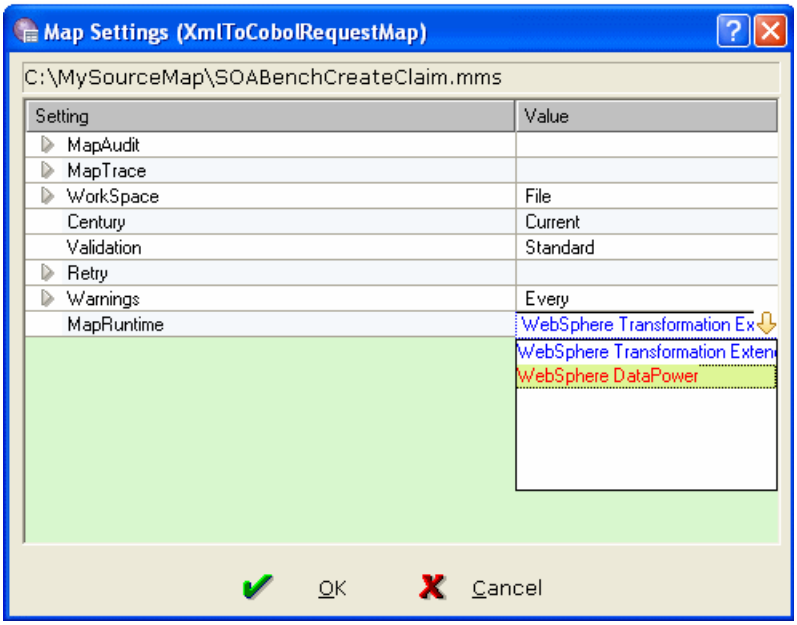


Figure 3-25 Change Map Runtime

7. Right-click **XmlToCobolRequestMap** and select **Build All** to create the files that we need to work with the DataPower device for deployment (CCINP.mts, SOAAssureService.mts, and XmlToCobolRequestMap.xml). When testing from the WebSphere Transformation Extender Studio, these files are sent as SOAP attachments to the “tx-test” firewall in the “tx” domain.
8. Right-click **XmlToCobolRequestMap** and select **Run** (Figure 3-26).

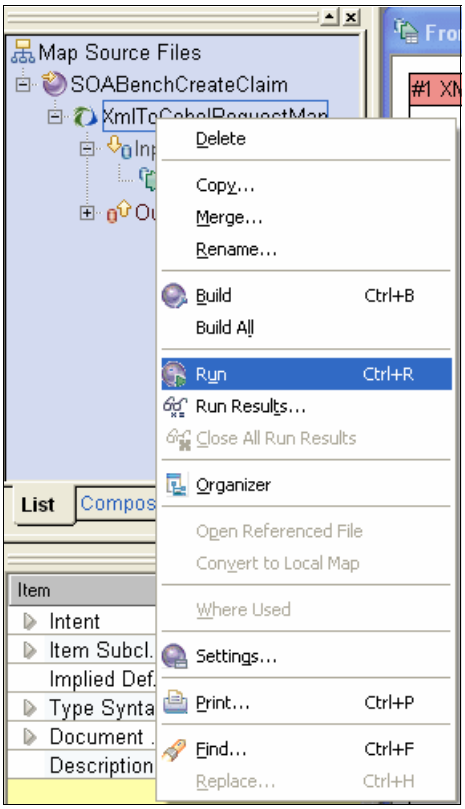


Figure 3-26 Run process

Upon successful completion, you see the DataPower SOA Appliances window (Figure 3-27).

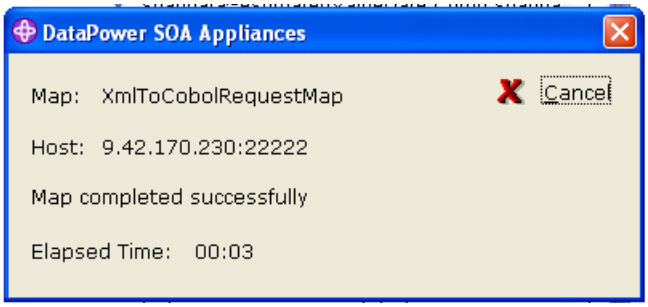


Figure 3-27 Run process results

Response mapping

Follow these steps:

1. Create mapping for a response message. Right-click **SOABenchCreateClaim** and select **New**.
2. In the Create New Map window, for New map name, type `CobolToXmlResponseMap` and click **OK**.
3. Expand **SOABenchCreateClaim** and expand **CobolToXmlResponseMap**.
4. Right-click **Input Cards** and select **New**.
5. In the Add Input Card window, enter the values from Table 3-5.

Table 3-5 Input card values

Field	Value	Explanation
CardName	INCobolResp	Enter any name to identify the input card.
TypeTree	CCOUT4K.mtt	Select the file that is generated by the Type Designer for the COBOL response type in the WTXWork folder.
Type	OUT01 Record CopyBook	Browse for the OUT01 Record CopyBook type, which is the last element in the CopyBook attributes list.
FilePath	WTXTest_ResponseSide_Inp.dat	Select the file to test the transformation the file into the WTXWork folder.

Figure 3-28 illustrates the INCobolResp fields and values.

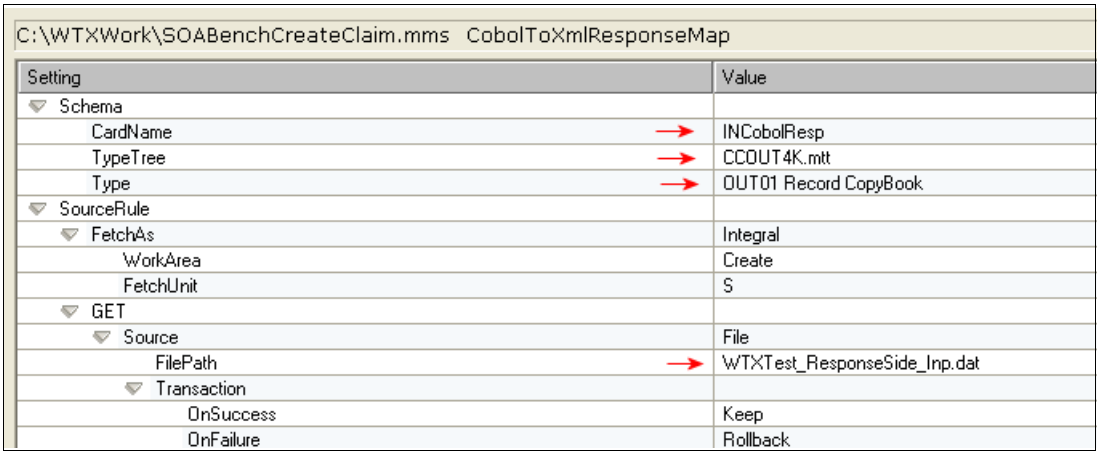


Figure 3-28 INCobolResp

Click **OK**.

6. Expand **Input Card**. The new input card is displayed, and the Input frame shows the values for the input message.
7. Right-click **Output Cards** and select **New**.

8. In the Output card values window, enter the values shown in Table 3-6.

Table 3-6 Output card values

Field	Value	Explanation
CardName	OutXmlResp	Enter any name to identify the output card.
TypeTree	SOAAssureService.mtt	Select the file generated by the Type Designer for the XML type in the WTXWork folder.
Type	Doc XSD	Browse for the Doc XSD type, which is the last option in the XSD attributes list.
Metadata	SOAAssureService.xsd	Select the XSD file.
FilePath	WTXTest_ResponseSide_OUT.xml	Specify the file where the output of the transformation will be placed by the WebSphere Transformation Extender Studio.

Figure 3-9 shows the OutXmlResp fields and values.

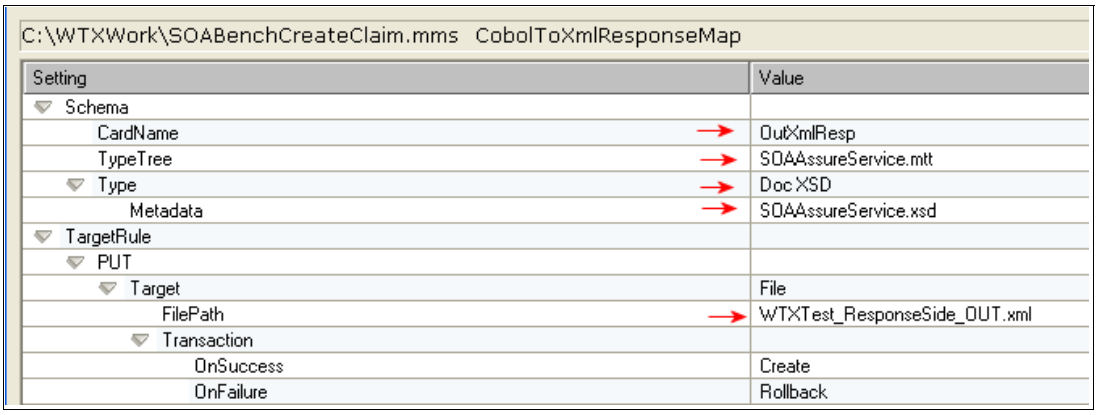


Figure 3-29 OutXmlResp

Click **OK**. Both message definitions are displayed in the From and To windows (Figure 3-30).

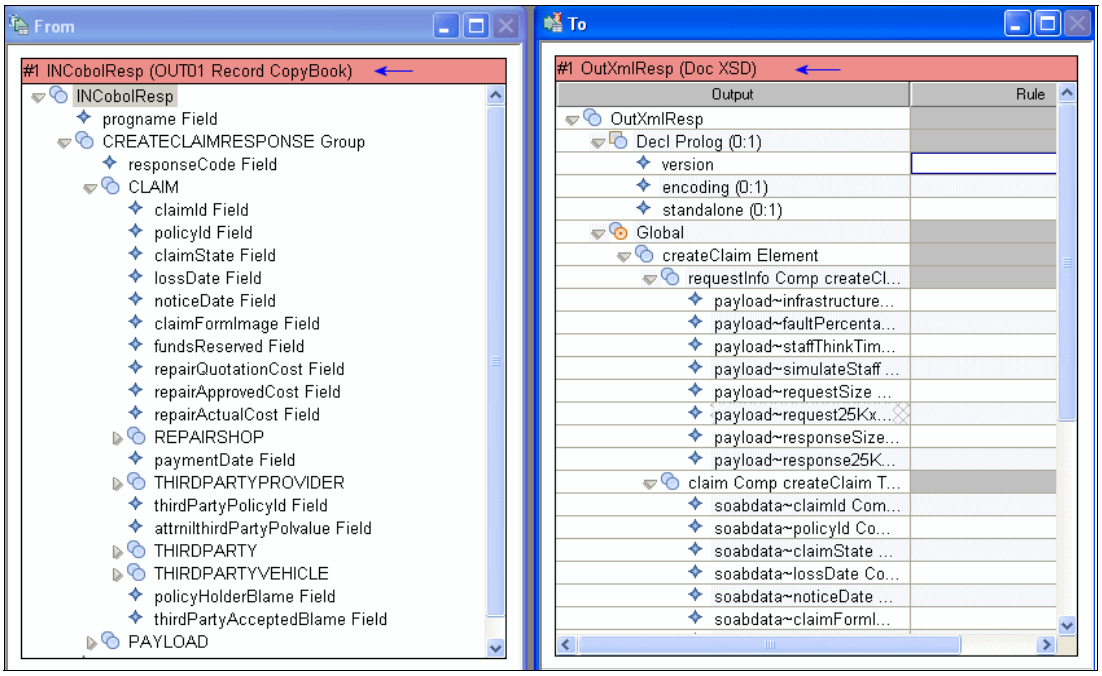


Figure 3-30 Response messages structure

9. Execute the mapping task. Follow the same steps as specified in “Request mapping” on page 65.
10. When the mapping is done, right-click **CobolToXmlResponseMap** and select **Build**. Verify that no errors are displayed.

Response map testing

Follow these steps:

1. Right-click **CobolToXmlResponseMap** and select **Settings**.
2. Change Map Runtime to `WebSphere DataPower` and click **OK**.
3. Right-click **CobolToXmlResponseMap** and select **Run** to create the files that we need to work with DataPower Device in transformation process. These are the CCOUT4K.mts and CobolToXmlResponseMap.xml files.
4. In the DataPower SOA Appliances window (Figure 3-31), verify that results are successful.

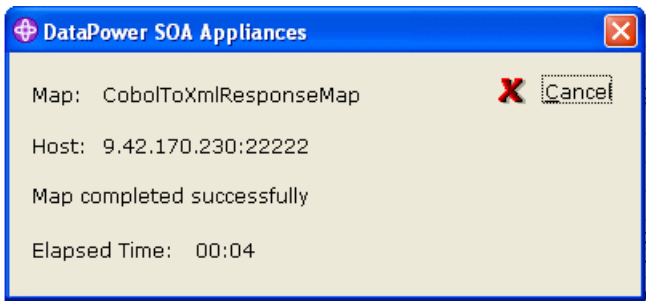


Figure 3-31 A successful process

5. Exit from Type Designer and click **Yes** to save any changes.

3.4 Deployment of the XML to COBOL transformations

At this stage, we have successfully developed the files for transformation and tested them against the DataPower run time. In this section, we illustrate how to use these transformation artifacts to create a multiprotocol gateway for providing the XML-to-COBOL service. Multiprotocol gateways are used when a protocol translation is needed in addition to a data transformation. The back-end protocol for communicating with the existing application is MQ. In this scenario, we explain how to set up the DataPower appliance for using an MQ resource. We assume that the actual MQ resource can be created or equivalent resources are available in your environment.

3.4.1 Creating the WebSphere MQ resources

In this scenario, we need to access the MQ components listed in Table 3-7. If equivalent MQ components are available in your environment, substitute their names appropriately.

Table 3-7 WebSphere MQ components

Component	Type and details	Name
Queue manager	MQ queue manager	QMDP
Queue	Local	TEST.REQUEST
Queue	Local	TEST.RESPONSE
Listener	Port: 1414	QMDP_Listener
Channel	SVRCONN	DP.CHANNEL

3.4.2 Importing transformation files developed in WebSphere Transformation Extender Studio into the DataPower appliance

To import the transformation files:

1. Log in to the DataPower WebGUI.
2. Change the domain if needed and click **Administration** → **File Management** (Figure 3-32).

Figure 3-32 File Management option

3. On the File Management page (Figure 3-33), for the local folder, click **Actions** → **Upload Files**.

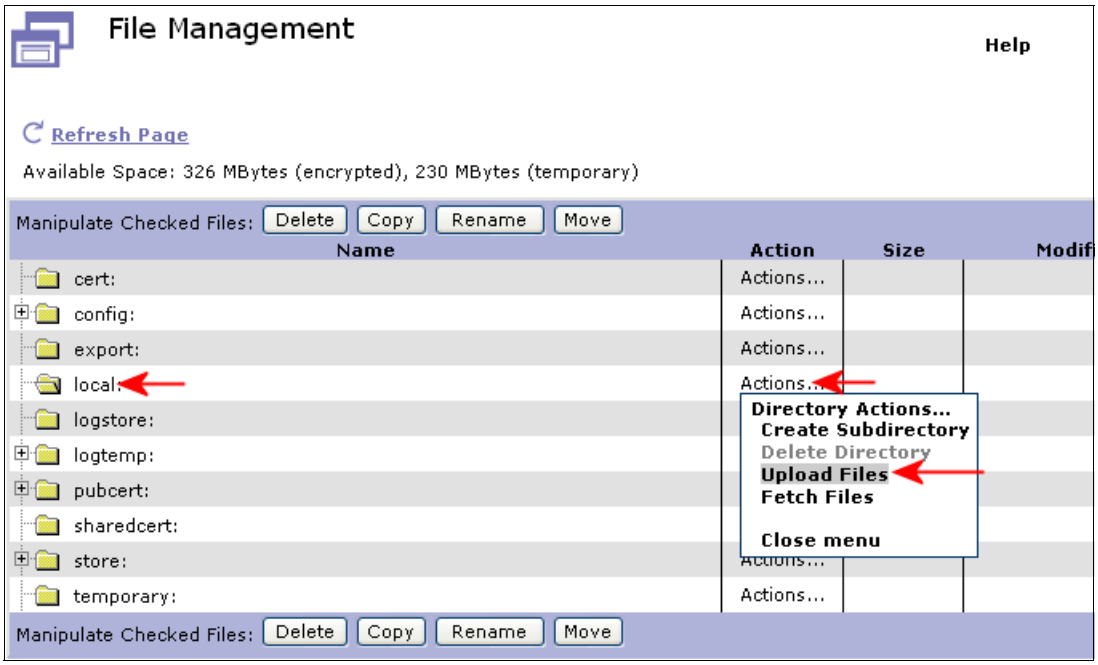


Figure 3-33 Upload Files action

4. Under File to upload (Figure 3-34), click **Browse**. Navigate to the WTXWork folder. Select the **CCINP.mts** file and click **Open**.
5. The tool returns to the File Management page. Click **Attach**.

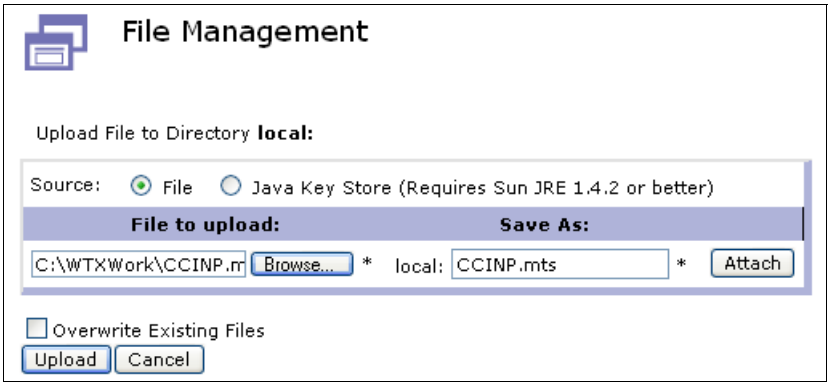


Figure 3-34 File Management page after uploading the CCINP.mts file

6. Repeat steps 4 and 5 to import the CCOUT4K.mts, CobolToXmlResponseMap.xml, and XmlToCobolRequestMap.xml files. When you are finished, click **Upload**. Figure 3-35 shows the uploaded files.

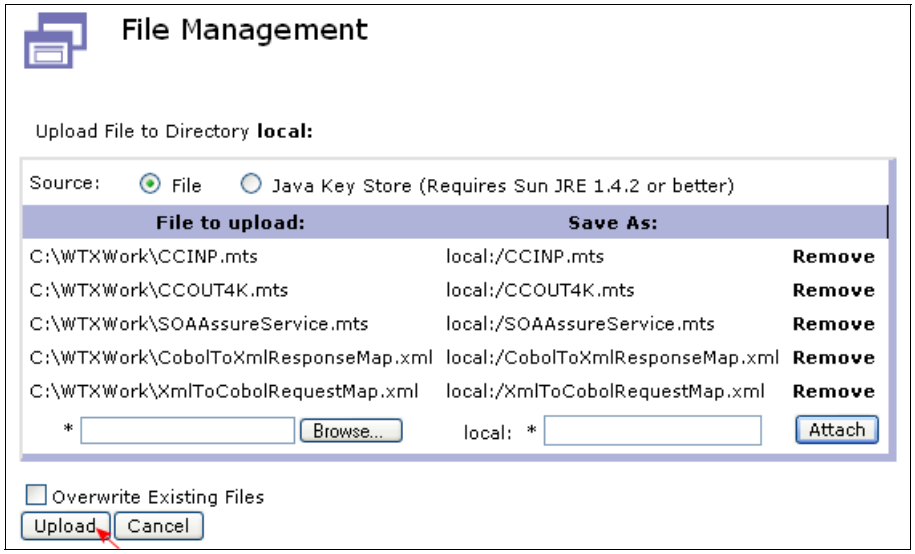


Figure 3-35 File Management page after uploading the additional files

7. Verify that all files loaded successfully as shown in Figure 3-36. Click **Continue**.

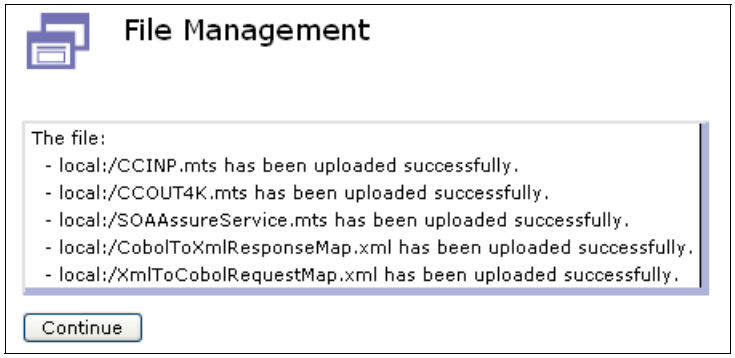


Figure 3-36 Files loaded successfully

Figure 3-37 shows the files in the local folder.

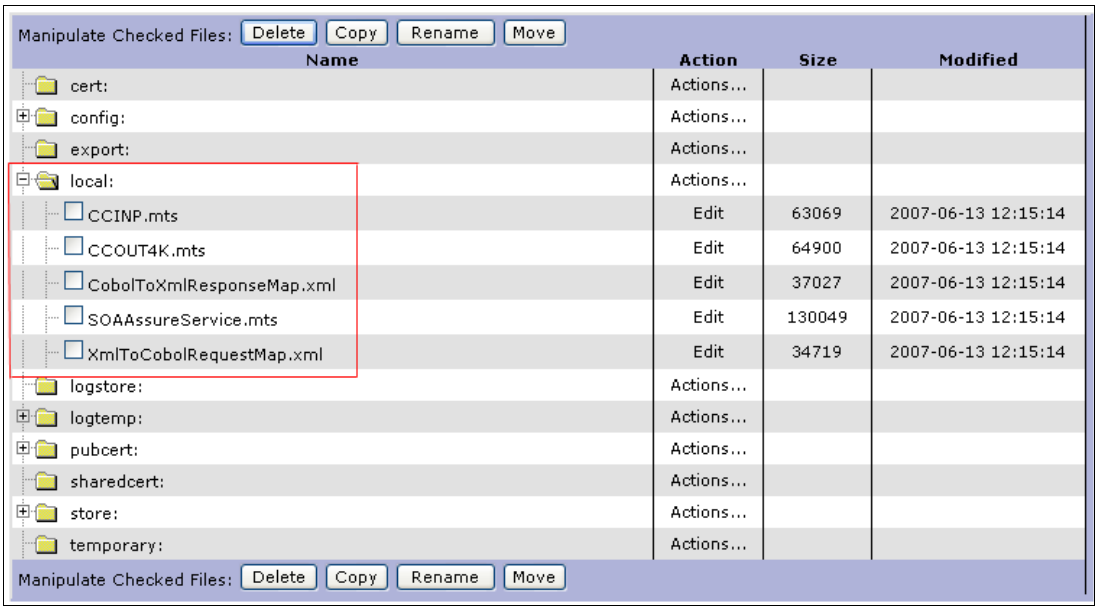


Figure 3-37 Files in the local folder

3.4.3 Creating a multiprotocol gateway

A multiprotocol gateway connects client requests that are transported over one or more protocols to a back-end service by using the same or a different protocol.

More information: You can learn more about the multiprotocol gateway in the DataPower reference guide documentation that is listed in “Other publications” on page 185.

To create a multiprotocol gateway:

1. On the Control Panel page, click **Multi-Protocol Gateway** (Figure 3-38).

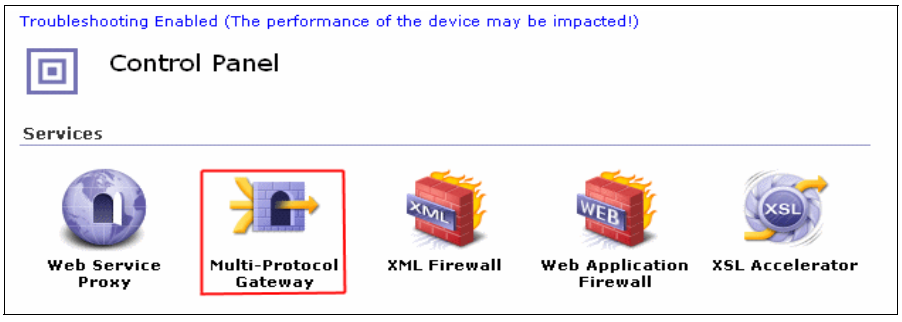


Figure 3-38 Multi Protocol Gateway Icon

2. On the Configure Multi-Protocol Gateway page, click **Add**.

3. On the **General** page, complete the following steps:
   a. In the Multi-Protocol Gateway Name field, enter `EnablingLegacyApp`.
   b. In the field summary, enter `Enabling Legacy Application Scenario`.
   c. In the Multi-Protocol Gateway Policy section, click the **+** button (Figure 3-39).

Figure 3-39 New policy button

4. In the Explorer User Prompt window (Figure 3-40), in the Processing Policy Name field, type `EnabligLegacyAppPolicy`, and click **OK**.

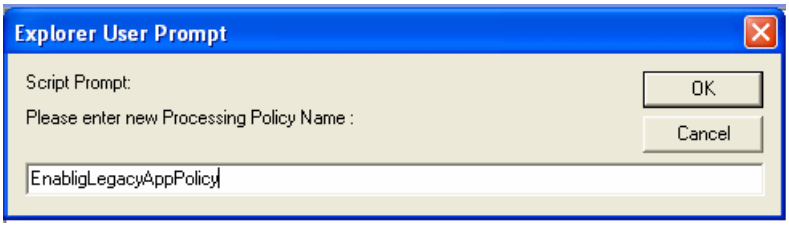


Figure 3-40 New policy name

5. In the next window, click **OK**.
6. On the Configure Multi-Protocol Gateway Protocol page (Figure 3-41), which now shows the policy icon, select **Client to Server** direction and double-click the **=** (Match) icon.

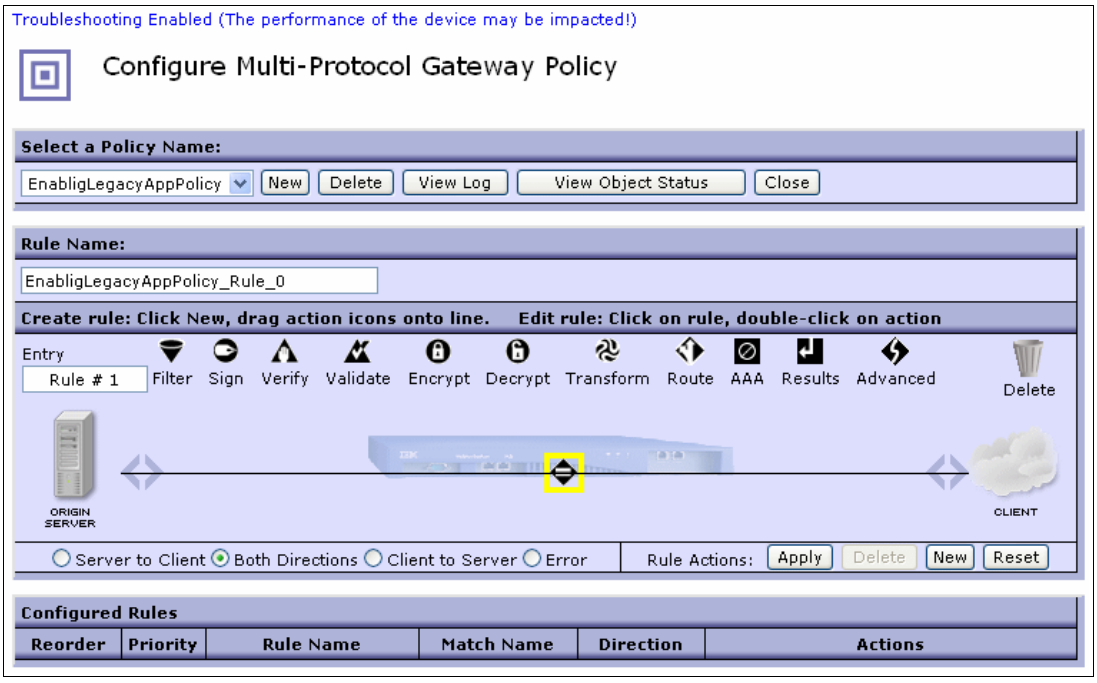


Figure 3-41 Configure Multi-Protocol Gateway page

7. On the Configure Match Action page, click the **+** button to create a new matching rule.
8. On the Configure Matching Rule page, in the Name field, type `MatchAll`.
9. Click the **Matching Rule** tab and click **Add**.

10. In the Adding new Matching Rule property window (Figure 3-42), for Matching Type, select **url**. For URL Match, type an asterisk (*) to match any request on this multiprotocol gateway. Click **Save**.

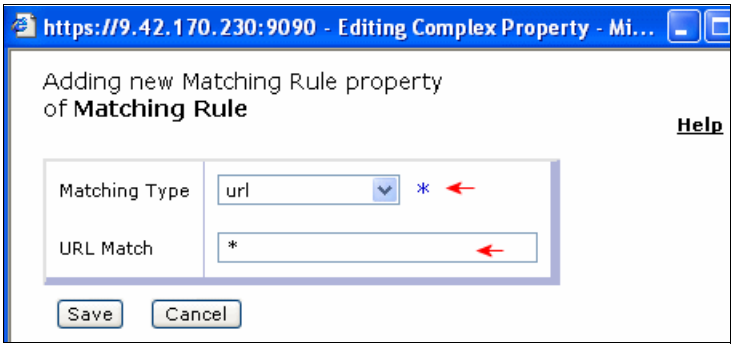


Figure 3-42 New Matching Rule parameters

11. On the Configure Matching Rule page (Figure 3-43), click **Apply**.

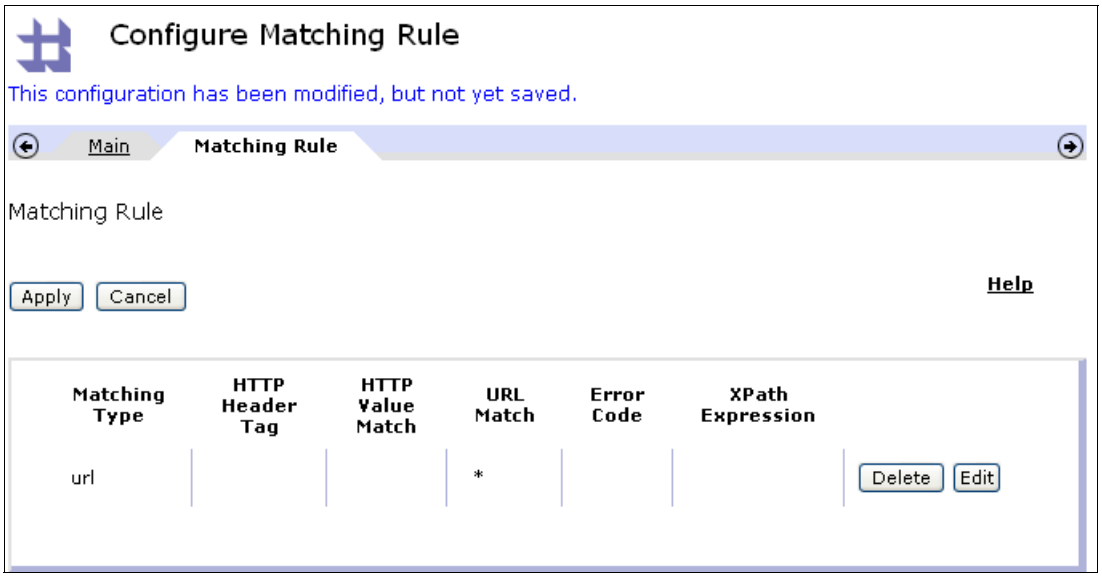


Figure 3-43 Matching rule created

12. Click **Done**. The new matching rule is active as shown in Figure 3-44.

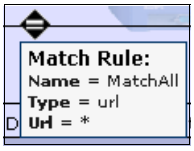


Figure 3-44 Active rule

13. Continue with rule creation by dragging the **Advanced** icon to the rule configuration path (the horizontal line). Double-click the **Advanced** icon (Figure 3-45).

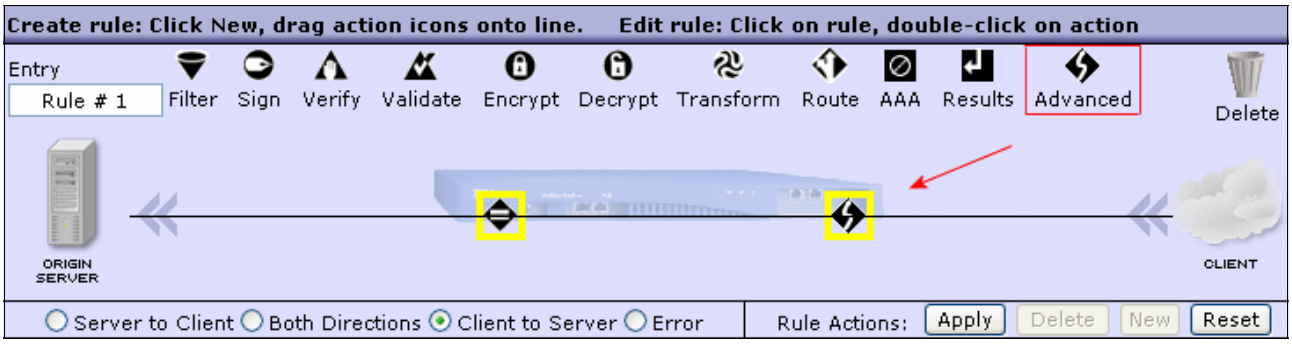


Figure 3-45 Add Advance icon

14. Select the **extract** option and click **Next** (Figure 3-46).

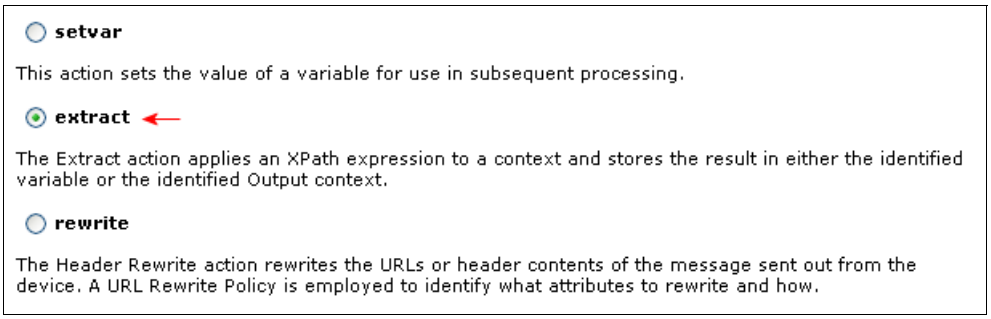


Figure 3-46 Extract option

15. On Configure Extract Using XPath Action page, in the input field, select **INPUT**.
16. Click **XPath Tool**.
17. On the Select XPath Expression for something in an XML File page (Figure 3-47 on page 85), complete the following steps:
    a. Click **Upload**.
    b. Click **Browse** and look for the **Client_Request.xml** file in WTXWork folder. Select the file and click **Open**.
    c. Back on Select XPath Expression for something in an XML File page, click **Upload** and then click **Continue**.
    d. Click **Refresh** to see the XML request message.

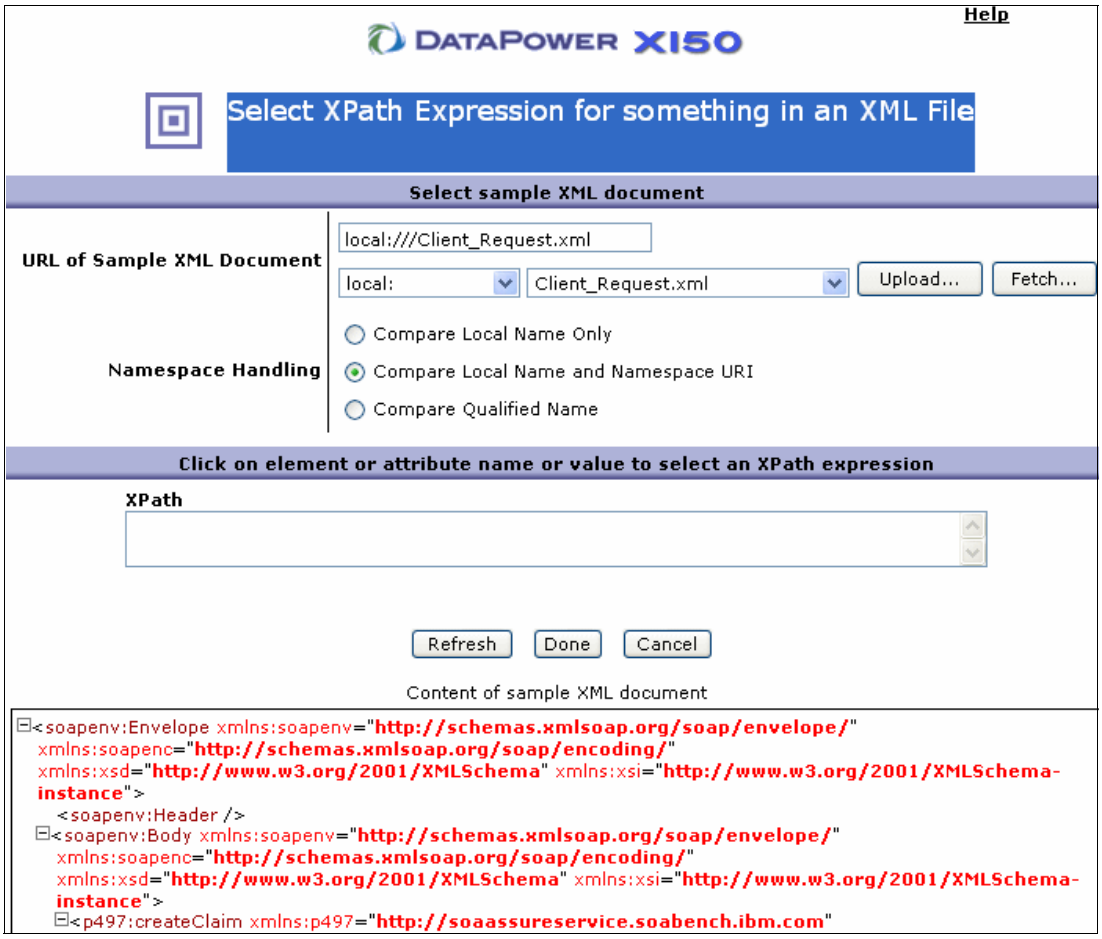


Figure 3-47 XML request message

e. In the Content of sample XML document section (Figure 3-48), click the **CreateClaim** XML element.

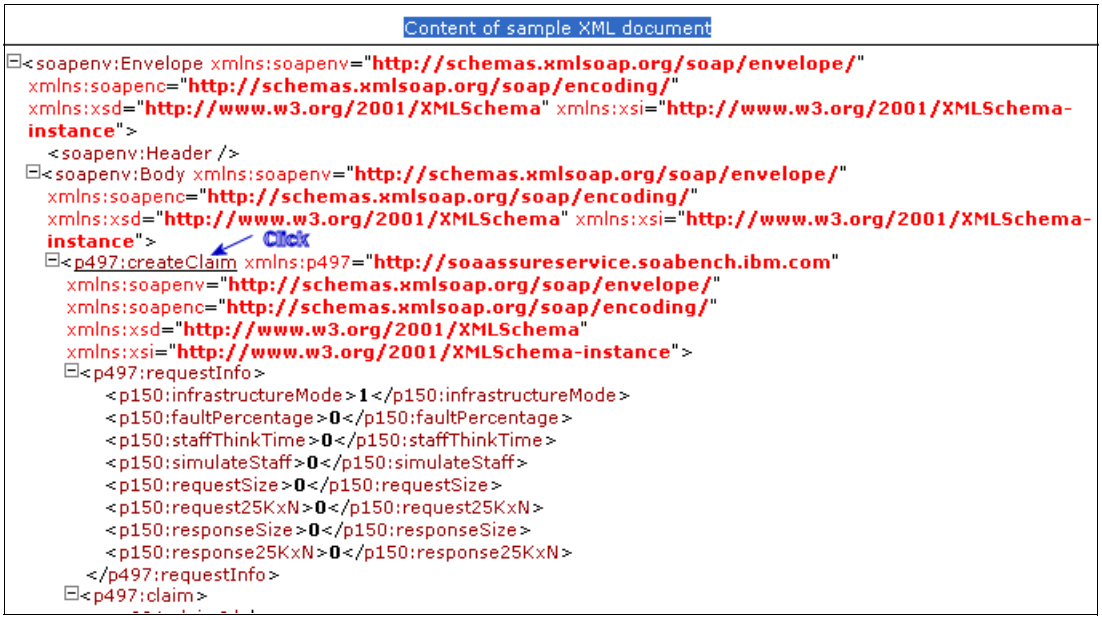


Figure 3-48 Selecting Create Claim

The page shows the XPath Information like the example in Figure 3-49.

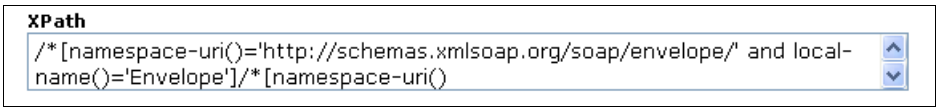


Figure 3-49 XPath details

f. Click **Done**.

g. Click **Done** to close the window.

18. Continue with the rule creation by dragging the **Transform** icon to the rule configuration path (the horizontal line). Double-click the **Transform** icon.

19. On the Configure Transform Binary Action page (Figure 3-50), complete these steps:

a. For Use Document Processing Instructions, select **Use XSLT specified in this action on a non-XML message**.

b. In the WTX Map file field, enter `local:///XmlToCobolRequestMap.xml`.

c. Click **Done**.

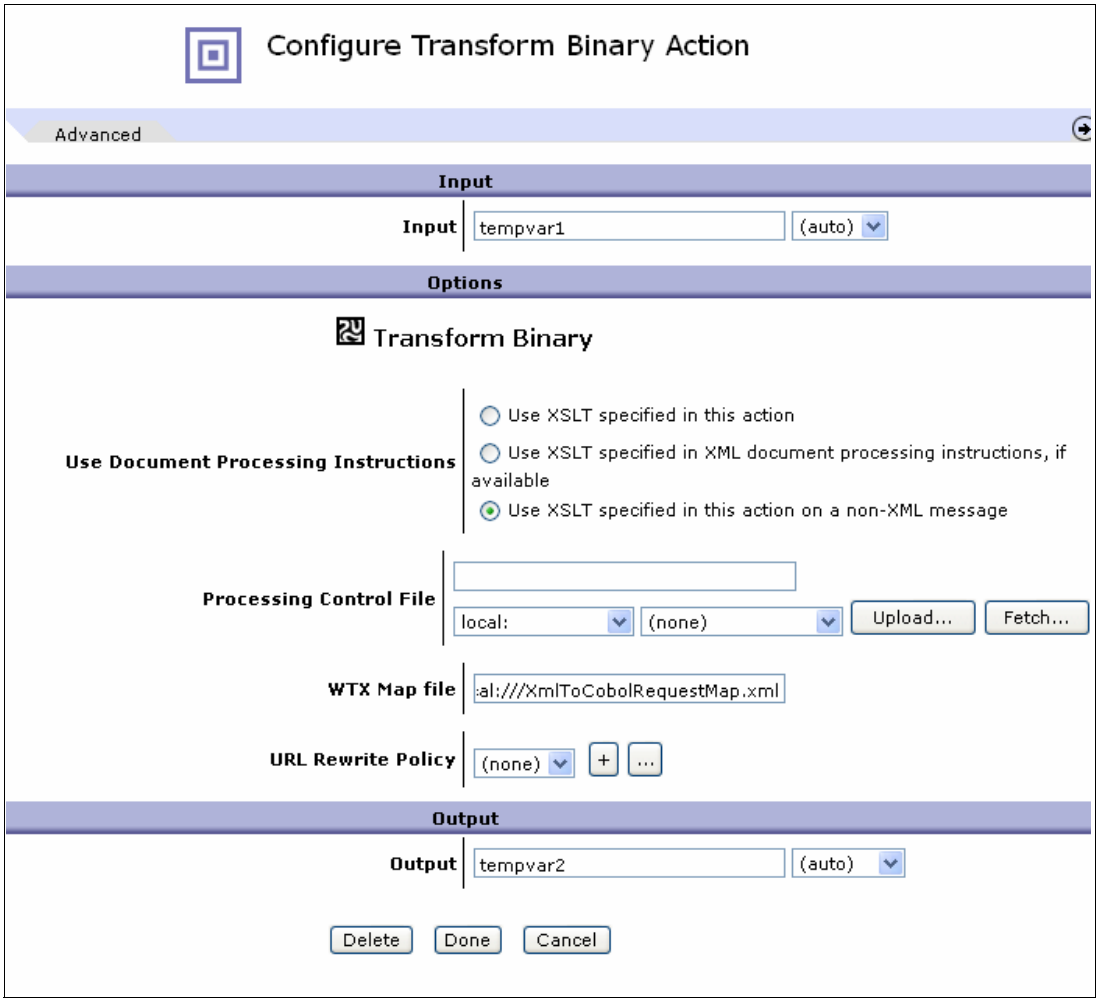


Figure 3-50 Configure Transform Binary Action details

20. Finish the request rule creation by dragging the **Results** icon to the rule configuration path (the horizontal line). Double-click the **Results** icon.
21. On the next page, click **Done**.
22. On the Configure Multi-Protocol Gateway Policy page (Figure 3-51), you see the complete rule configuration path. Click **Apply**.

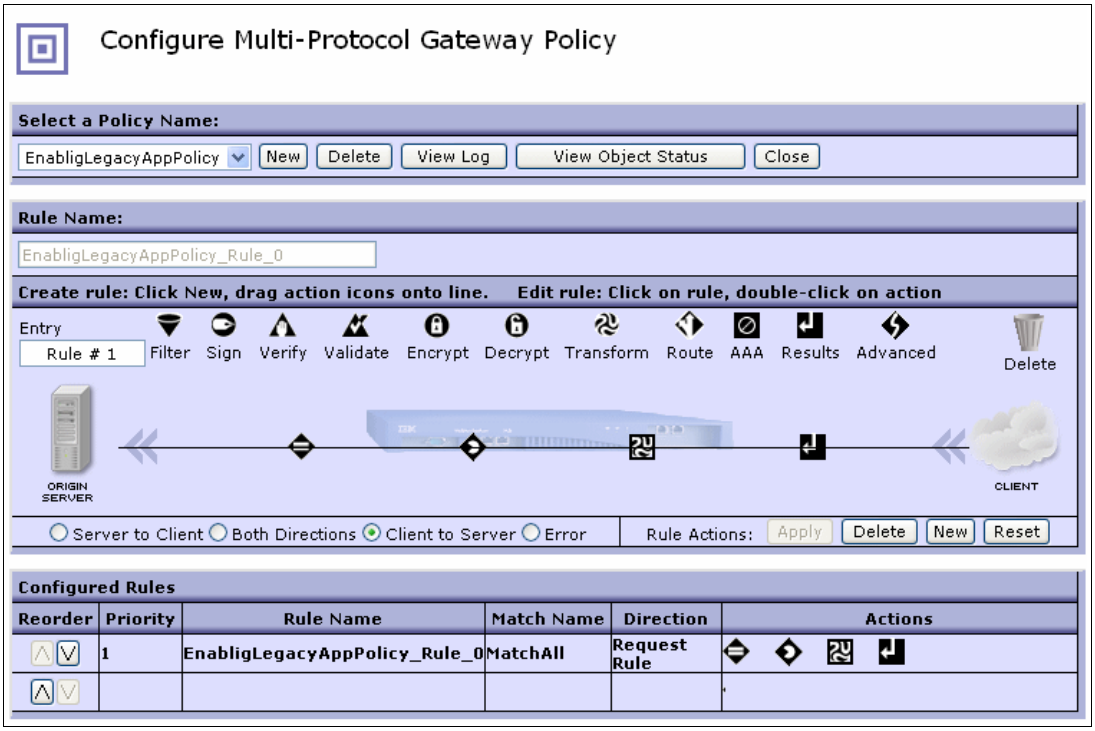


Figure 3-51 Complete rule configuration path

23. Create the response rule configuration path. Click **New** and change direction to **Server to Client** (Figure 3-52).

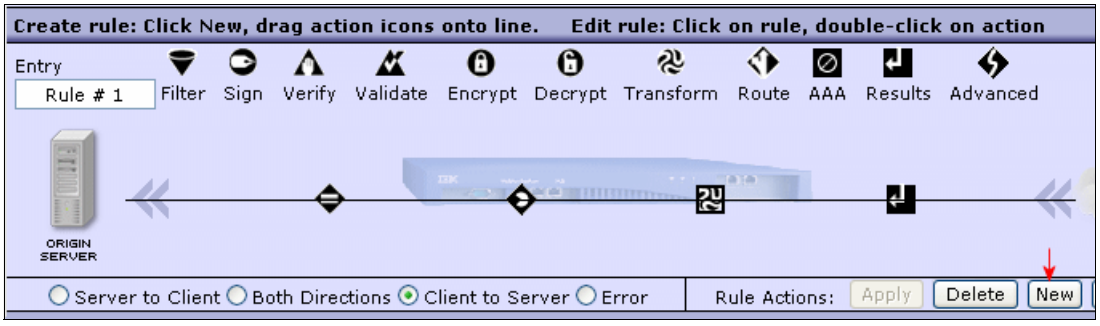


Figure 3-52 New rule configuration path

24. Double-click the **Rule** icon.
25. On the next page, in the Matching Rule field, select **MatchAll**. Then click **Done**.
26. Continue with rule creation by dragging the **Transformation** icon to the rule configuration path (the horizontal line). Double-click the **Transformation** icon.

27. On the Configure Transform Binary Action page (Figure 3-53), complete these steps:
    a. For Use Document Processing Instructions, select **Use XSLT specified in this action on a non-XML message**.
    b. For the WTX Map File, enter `local:///CobolToXmlResponseMap.xml`.
    c. Click **Done**.

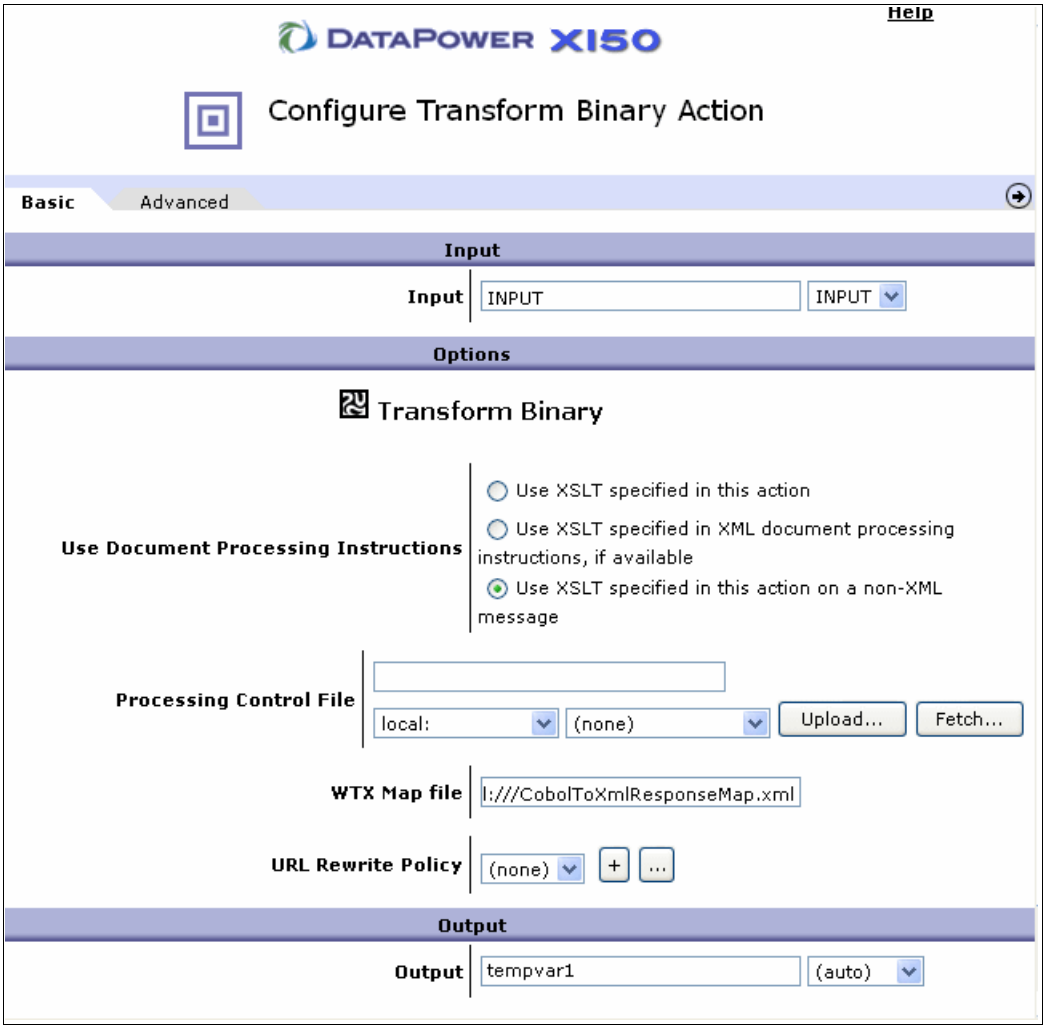


Figure 3-53 Transform Binary Action details

28. Continue with rule creation by dragging another **Transformation** icon to the rule configuration path (the horizontal line). Double-click the **Transformation** icon.
29. On the next page, in the Use Document Processing Instructions field, *s*elect **Use XSLT specified in this action**. Click **Upload**.

30. On the File Management page (Figure 3-54), click **Browse** and select the **converttosoap.xsl** file. Click **Upload** and then click **Continue**.

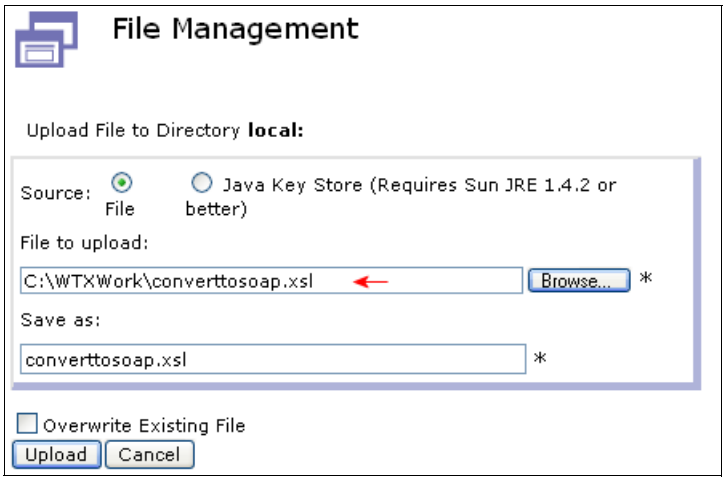


Figure 3-54 File Management page

31. On the Configure Transform Action page (Figure 3-55), click **Done**.

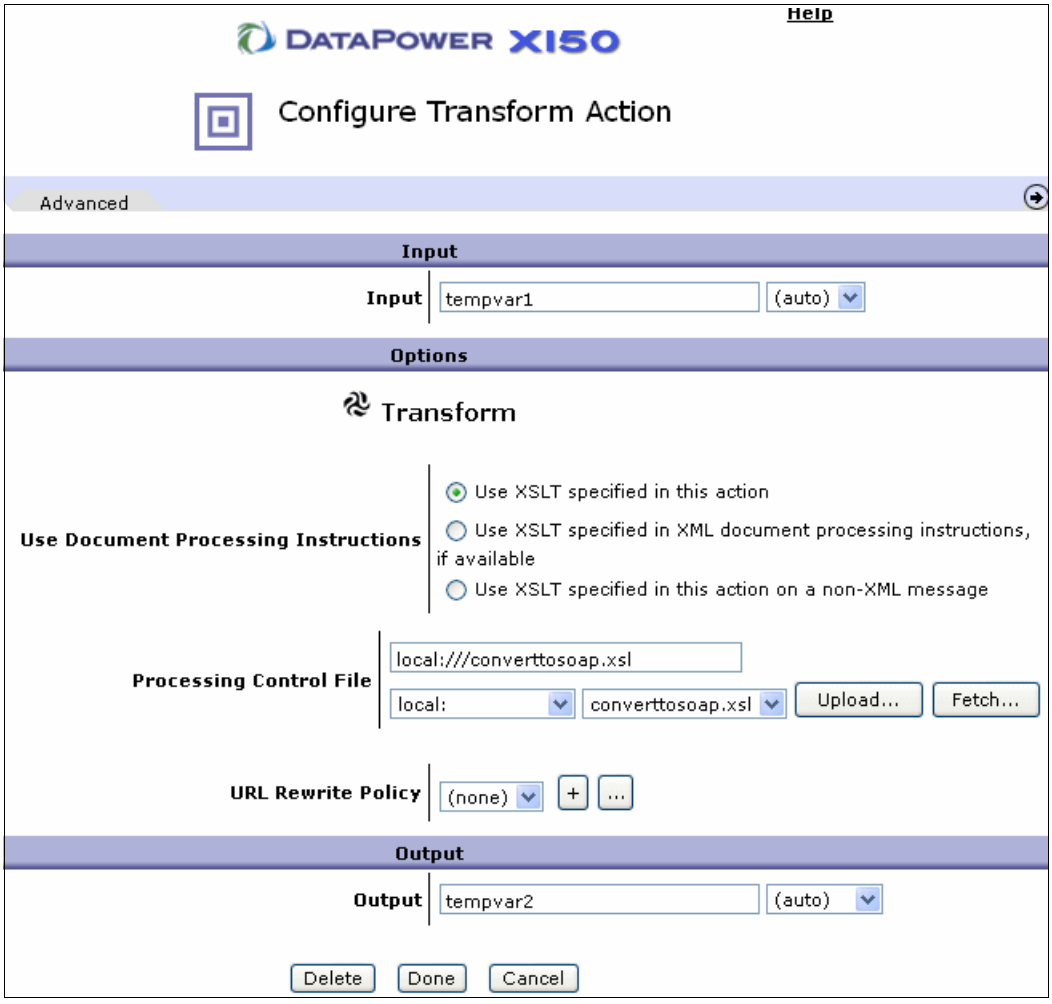


Figure 3-55 Configure Transform Action details

32. Continue with the rule creation by dragging a **Results** icon to the rule configuration path (the horizontal line). Double-click the **Results** icon.

33. On the Configure Results Action page (Figure 3-56), click **Done**.

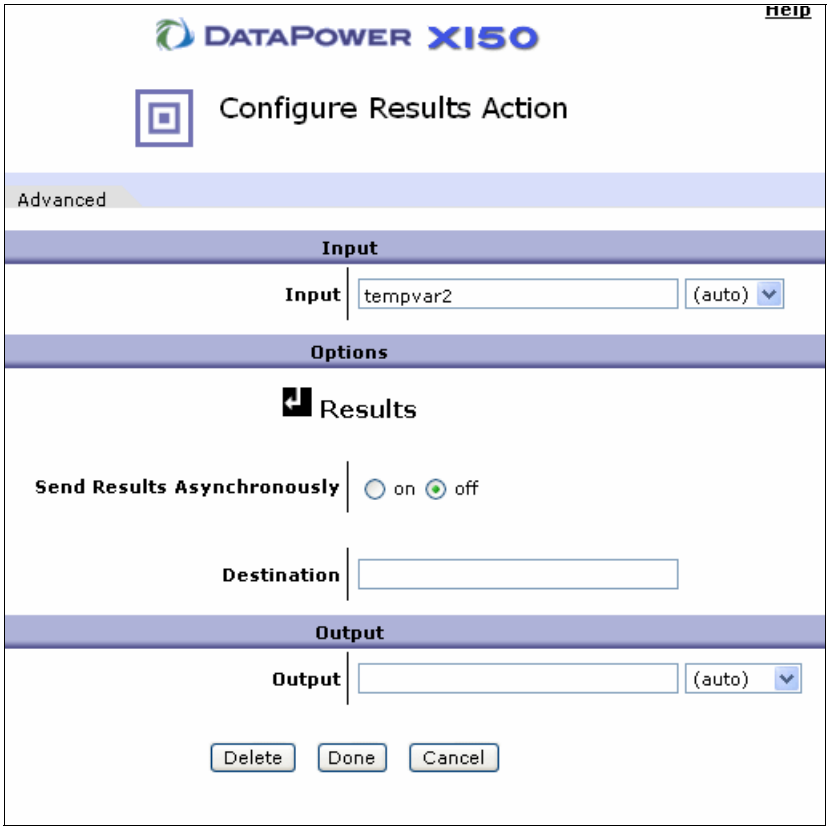


Figure 3-56 Configure Results Action page

34. The response Rule Configuration Path is displayed as shown in Figure 3-57. Click **Apply** and then click **Close**.

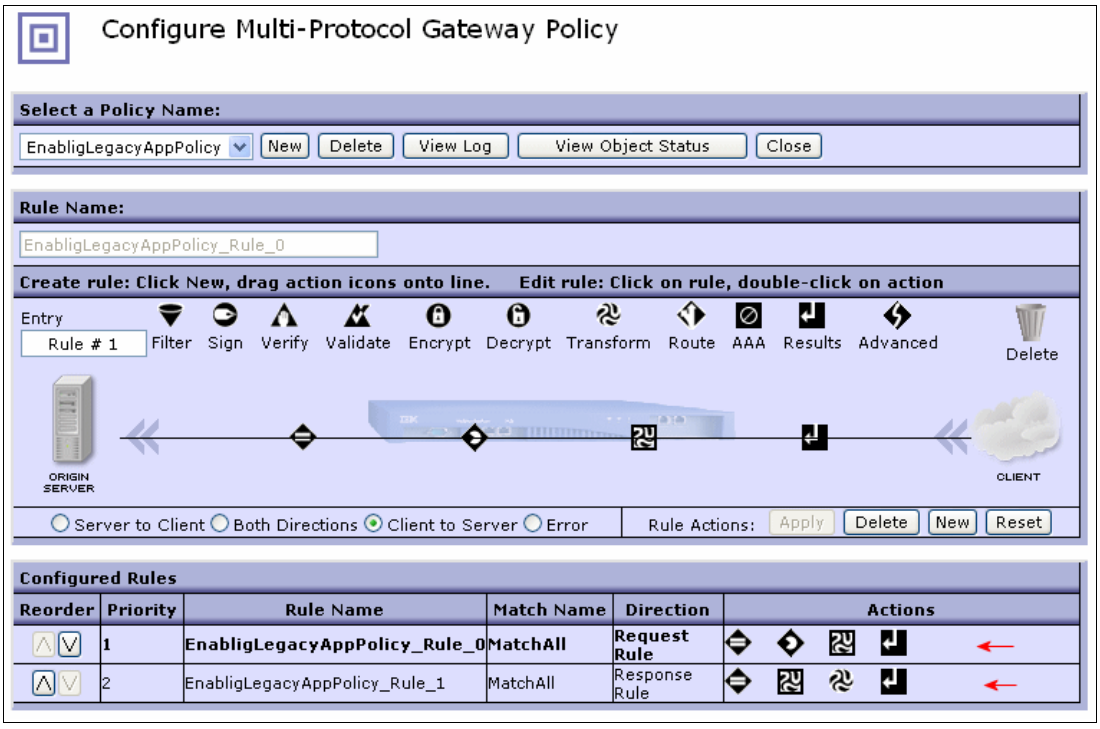


Figure 3-57 Rule configuration path

35. Configure the back side setting. In the Back side settings section (Figure 3-58), click the **MQ Helper** button.

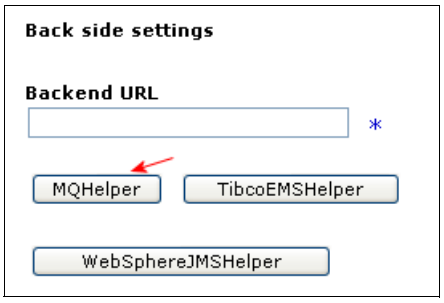


Figure 3-58 MQHelper button

36. On the MQ URL Builder page (Figure 3-59), in the Queue Manager field, click the **+** button and then select **MQ Queue Manager**.

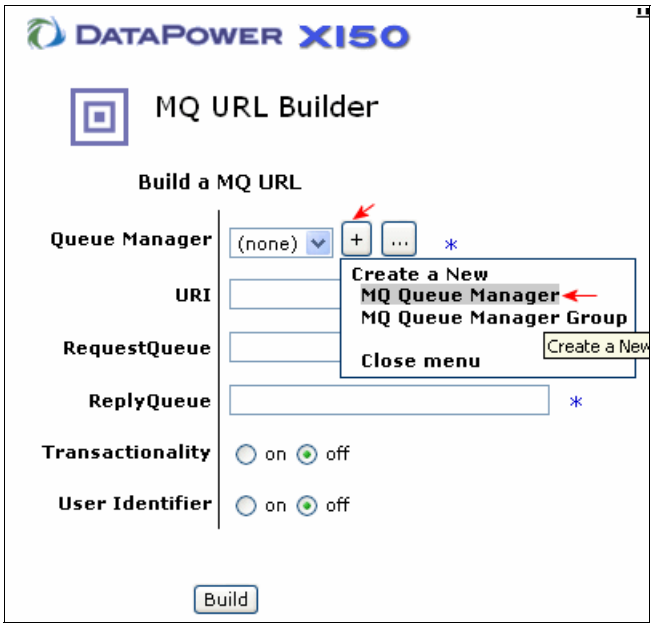


Figure 3-59 MQ URL Builder page

37. On the Configure MQ Queue Manager page, enter the values shown in Table 3-8 and click **Apply**.

Table 3-8 Configure MQ Queue Manager values

Field	Value	Explanation
Name	DataPowerQManager	DataPower MQ manager object name
Comments	DataPower Qmanager	
Host Name	dp_QM(1414)	IP address or host name of the WebSphereMQ server that hosts this queue manager
Queue Manager Name	QMDP	Name of the queue manager to connect
Channel Name	DP.CHANNEL	The MQ Channel
User Name	wmbadmin	The plaintext string that is sent to the server for identifying the client

Figure 3-60 shows the configured MQ Queue Manager settings.

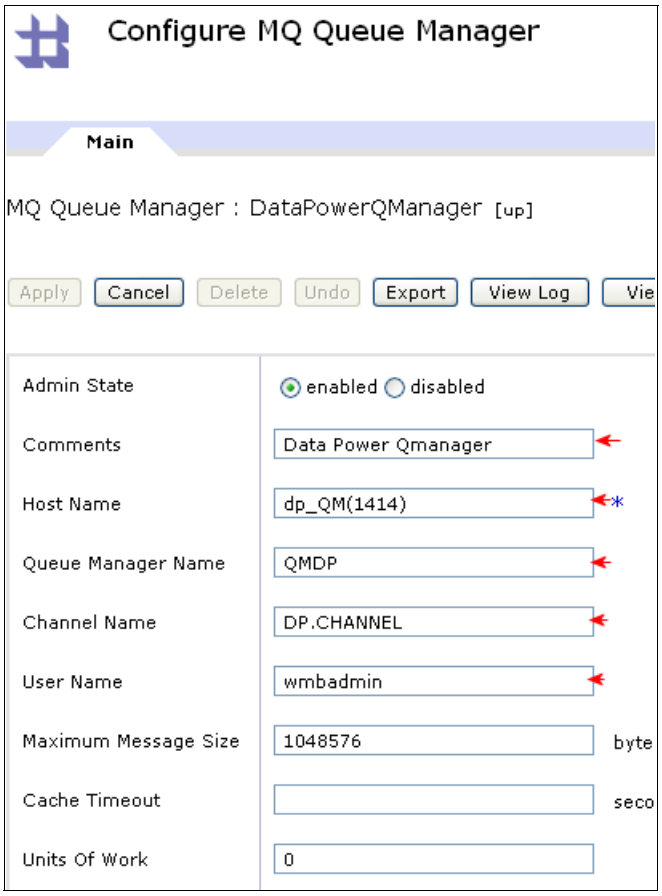


Figure 3-60 Configure MQ Queue Manager parameters

38. On the MQ URL Builder page, enter the values shown in Table 3-9 and click **Build**.

Table 3-9 MQ URL Builder values

Field	Value
Queue Manager	DataPowerQManager
RequestQueue	TEST.REQUEST
ReplyQueue	TEST.RESPONSE

Figure 3-61 shows the configured MQ URL Builder page.

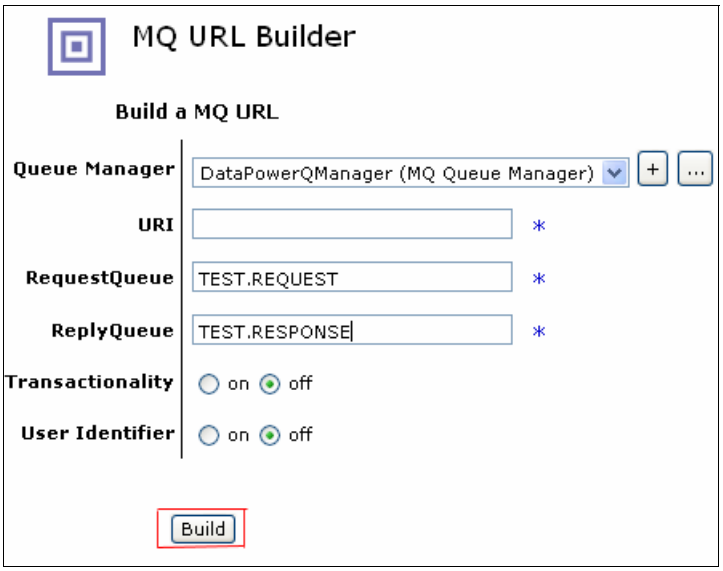


Figure 3-61 MQ URL Builder page

39. In the Back side settings section, in the Backend URL field, add the value `;ParseHeaders=true`. The last Backend URL value should read as follows:

```
dpmq://DataPowerQManager/?RequestQueue=TEST.REQUEST
;ReplyQueue=TEST.RESPONSE;ParseHeaders=true
```

40. In the Front side settings section (Figure 3-62), click the **Create new** button and select **HTTP Front Side Handler**.

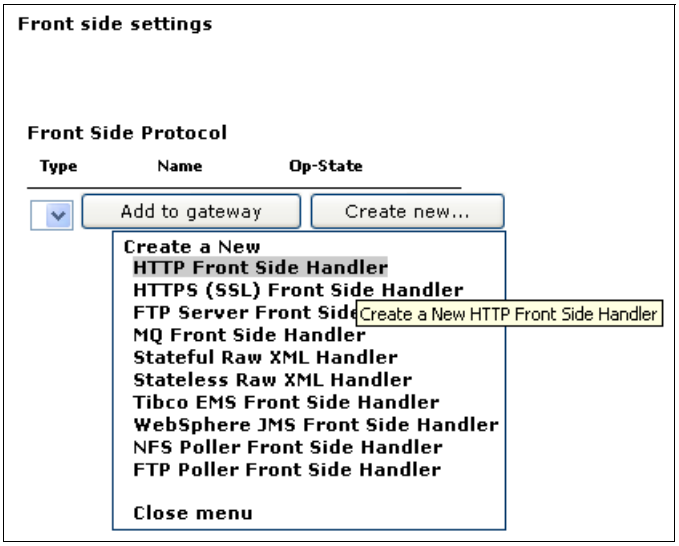


Figure 3-62 Create HTTP Front Side Handler

41. On the Configure HTTP Front Side Handler page (Figure 3-63), complete these steps:

    a. In the Name field, enter `EnableLegacyApp_HTTPFSH`.
    b. In the Port Number field, enter `8081`.
    c. Click **Apply**.

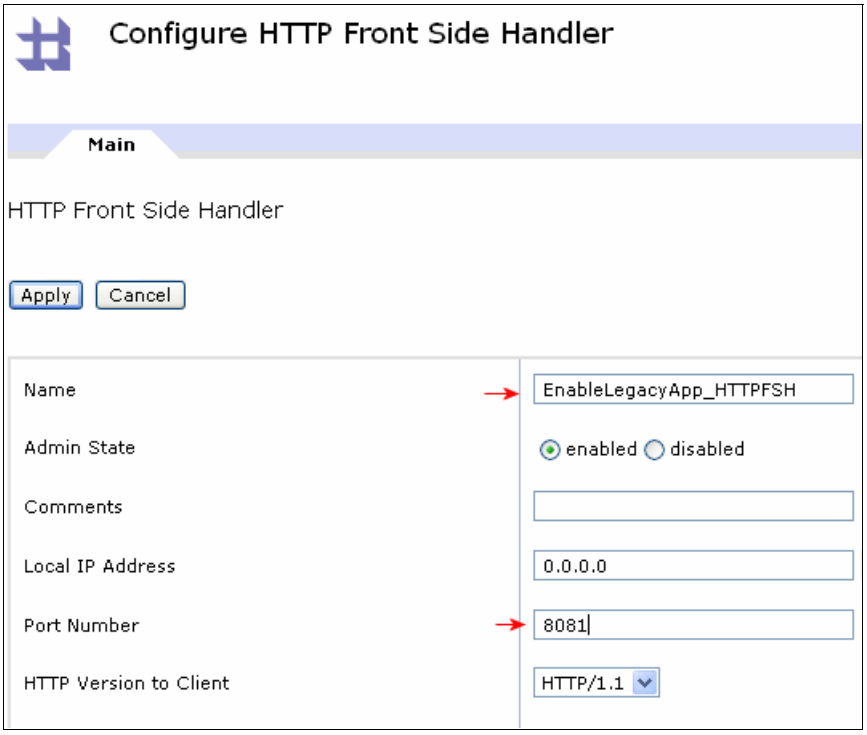


Figure 3-63 HTTP Front Side Handler values

42. Click **Apply**.

You have now created the multiprotocol gateway. If you are successful, the status is displayed as *up* as illustrated in Figure 3-64.

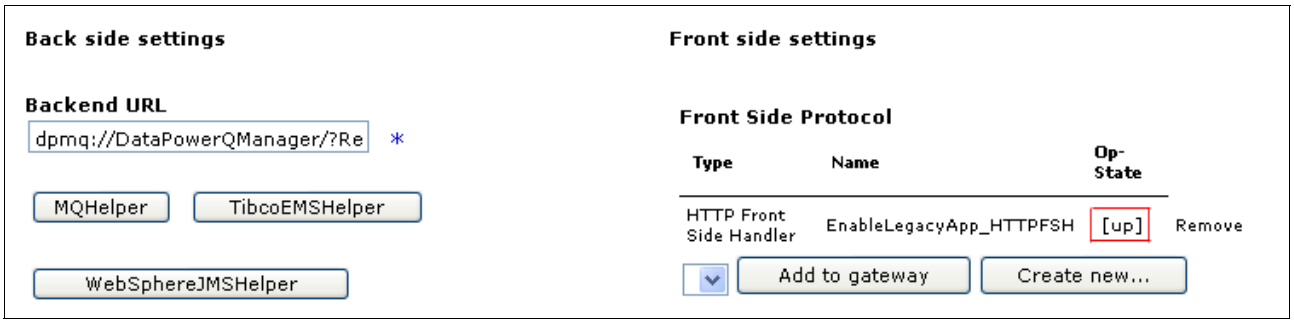


Figure 3-64 Front side settings

3.5 Running the application

To run the sample application:

1. Verify that the MQ components are running.
2. Enter the WTXWork directory and run the Host Server simulation program by using the following command:

   ```
   redbookserver -i TEST.REQUEST -o TEST.RESPONSE -m QMDP -f Host_Response.dat
   ```

   You receive a successful message indicating that redbookmqserver is running (Figure 3-65).

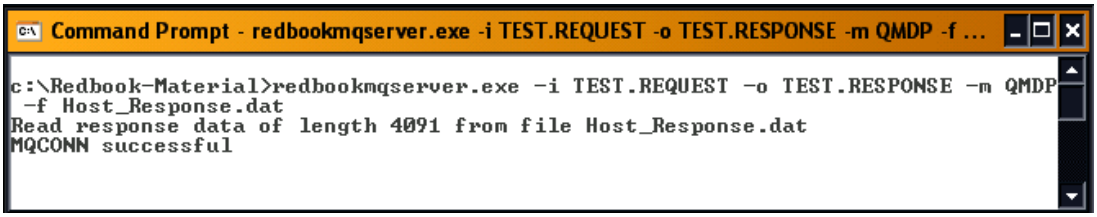


Figure 3-65 redbookmqserver running

3. In this sample scenario, we use the cURL tool as a client to send the request. Copy the cURL directory into the WTXWork folder, enter the cURL directory, and enter the following command:

   ```
   curl -v -D -X POST d@..\Client_Request.xml http://<ip address or host name of
   your datapower device>:8081/
   ```

4. Verify the result with the WTX files. The transformation results are displayed in the Command Prompt window as shown in Figure 3-66.

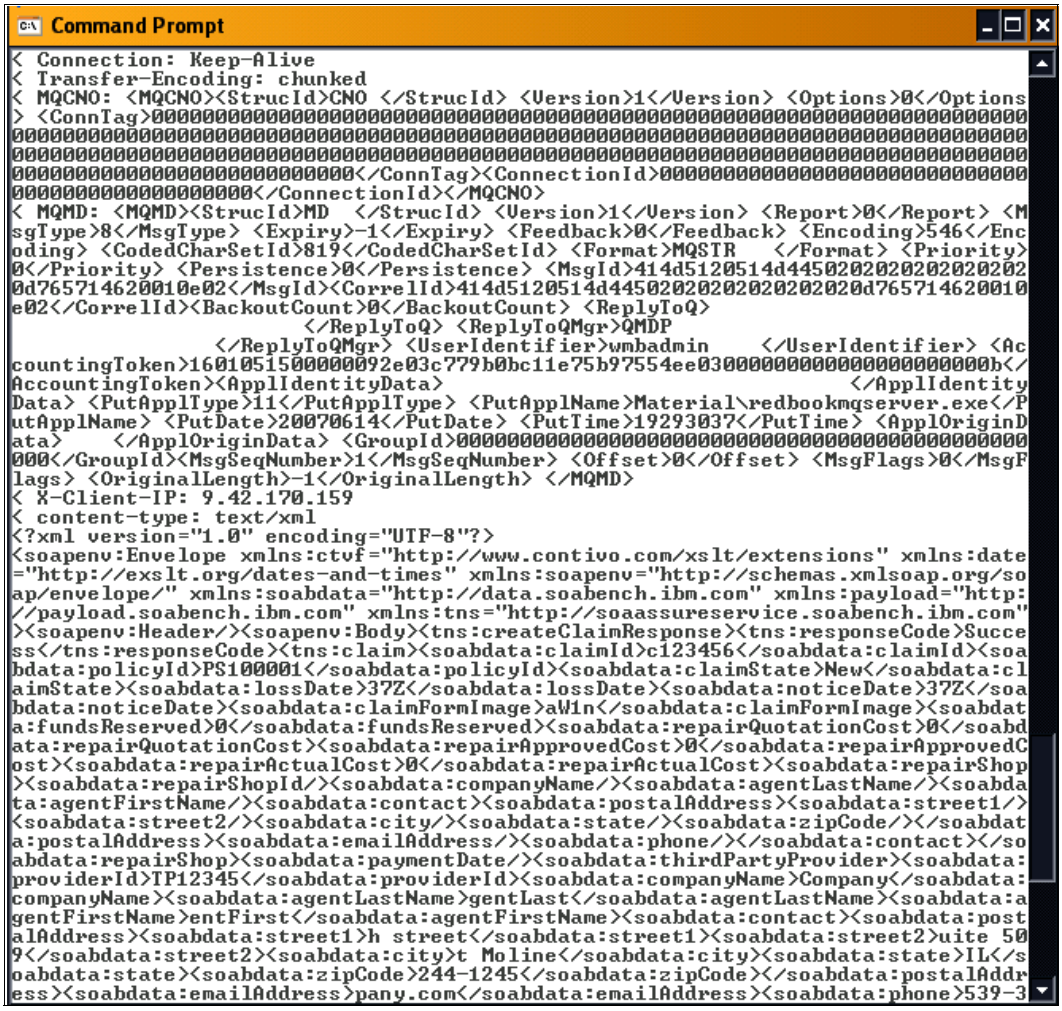


Figure 3-66 Transformation execution

Example 3-1 shows a sample of the results of using the **curl** command.

Example 3-1 Sample results

```
c:\Redbook-Material\Curl>curl -v -D -X POST -d@..\Client_Request.xml http://9.42
.170.230:8081/Test
* Could not resolve host: POST; Host not found
* Closing connection #
curl: (6) Could not resolve host: POST; Host not found
* About to connect() to 9.42.170.230 port 8081
*   Trying 9.42.170.230... connected
* Connected to 9.42.170.230 (9.42.170.230) port 8081
> POST /Test HTTP/1.1
> User-Agent: curl/7.15.0 (i586-pc-mingw32msvc) libcurl/7.15.0 OpenSSL/0.9.7e zl
ib/1.2.2
> Host: 9.42.170.230:8081
> Accept: */*
> Content-Length: 3179
> Content-Type: application/x-www-form-urlencoded
```

```
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
< X-Note: Gateway Ack
HTTP/1.1 200 OK
< X-Backside-Transport: OK OK
< Connection: Keep-Alive
< Transfer-Encoding: chunked
< MQCNO: <MQCNO><StrucId>CNO </StrucId> <Version>1</Version> <Options>0</Options
> <ConnTag>000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000</ConnTag><ConnectionId>000000000000000000000000000000000000
0000000000000000</ConnectionId></MQCNO>
< MQMD: <MQMD><StrucId>MD  </StrucId> <Version>1</Version> <Report>0</Report> <M
sgType>8</MsgType> <Expiry>-1</Expiry> <Feedback>0</Feedback> <Encoding>546</Enc
oding> <CodedCharSetId>819</CodedCharSetId> <Format>MQSTR   </Format> <Priority>
0</Priority> <Persistence>0</Persistence> <MsgId>414d5120514d4450202020202020202020
0d765714620013803</MsgId><CorrelId>414d5120514d4450202020202020202020d765714620013
803</CorrelId><BackoutCount>0</BackoutCount> <ReplyToQ>
                        </ReplyToQ> <ReplyToQMgr>QMDP
                  </ReplyToQMgr> <UserIdentifier>wmbadmin    </UserIdentifier> <Ac
countingToken>1601051500000092e03c779b0bc11e75b97554ee0300000000000000000000000b</
AccountingToken><ApplIdentityData>                                        </ApplIdentity
Data> <PutApplType>11</PutApplType> <PutApplName>Material\redbookmqserver.exe</P
utApplName> <PutDate>20070614</PutDate> <PutTime>20090998</PutTime> <ApplOriginD
ata>    </ApplOriginData> <GroupId>000000000000000000000000000000000000000000000000
000</GroupId><MsgSeqNumber>1</MsgSeqNumber> <Offset>0</Offset> <MsgFlags>0</MsgF
lags> <OriginalLength>-1</OriginalLength> </MQMD>
< X-Client-IP: 9.42.170.159
< content-type: text/xml
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:ctvf="http://www.wtx.com/xslt/extensions" xmlns:date
="http://exslt.org/dates-and-times" xmlns:soapenv="http://schemas.xmlsoap.org/so
ap/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>Succe
ss</tns:responseCode><tns:claim><soabdata:claimId>c123456</soabdata:claimId><soa
bdata:policyId>PS100001</soabdata:policyId><soabdata:claimState>New</soabdata:cl
aimState><soabdata:lossDate>37Z</soabdata:lossDate><soabdata:noticeDate>37Z</soa
bdata:noticeDate><soabdata:claimFormImage>aW1n</soabdata:claimFormImage><soabdat
a:fundsReserved>0</soabdata:fundsReserved><soabdata:repairQuotationCost>0</soabd
ata:repairQuotationCost><soabdata:repairApprovedCost>0</soabdata:repairApprovedC
ost><soabdata:repairActualCost>0</soabdata:repairActualCost><soabdata:repairShop
><soabdata:repairShopId/><soabdata:companyName/><soabdata:agentLastName/><soabda
ta:agentFirstName/><soabdata:contact><soabdata:postalAddress><soabdata:street1/>
<soabdata:street2/><soabdata:city/><soabdata:state/><soabdata:zipCode/></soabdat
a:postalAddress><soabdata:emailAddress/><soabdata:phone/></soabdata:contact></so
abdata:repairShop><soabdata:paymentDate/><soabdata:thirdPartyProvider><soabdata:
providerId>TP12345</soabdata:providerId><soabdata:companyName>Company</soabdata:
companyName><soabdata:agentLastName>gentLast</soabdata:agentLastName><soabdata:a
gentFirstName>entFirst</soabdata:agentFirstName><soabdata:contact><soabdata:post
alAddress><soabdata:street1>h street</soabdata:street1><soabdata:street2>uite 50
9</soabdata:street2><soabdata:city>t Moline</soabdata:city><soabdata:state>IL</s
oabdata:state><soabdata:zipCode>244-1245</soabdata:zipCode></soabdata:postalAddr
```

ess><soabdata:emailAddress>pany.com</soabdata:emailAddress><soabdata:phone>539-3
434</soabdata:phone></soabdata:contact></soabdata:thirdPartyProvider><soabdata:t
hirdPartyPolicyId>2435-235</soabdata:thirdPartyPolicyId><soabdata:thirdParty><so
abdata:lastName>DeGuzman</soabdata:lastName><soabdata:firstName>lexander</soabda
ta:firstName><soabdata:dateOfBirth>37Z</soabdata:dateOfBirth><soabdata:gender>Ma
le</soabdata:gender><soabdata:maritalStatus>Married</soabdata:maritalStatus><soa
bdata:contact><soabdata:postalAddress><soabdata:street1>h street</soabdata:stree
t1><soabdata:street2/><soabdata:city>t Moline</soabdata:city><soabdata:state>IL<
/soabdata:state><soabdata:zipCode>244-1245</soabdata:zipCode></soabdata:postalAd
dress><soabdata:emailAddress>pany.com</soabdata:emailAddress><soabdata:phone>539
-3434</soabdata:phone></soabdata:contact></soabdata:thirdParty><soabdata:thirdPa
rtyVehicle><soabdata:vin>389OK343</soabdata:vin><soabdata:registrationMark>TQF-3
345</soabdata:registrationMark><soabdata:manufacturer>Ford</soabdata:manufacture
r><soabdata:model>plorer V</soabdata:model><soabdata:year>2OO2</soabdata:year><s
oabdata:estimatedValue>O.67</soabdata:estimatedValue><soabdata:estimatedValueDat
e>37Z</soabdata:estimatedValueDate></soabdata:thirdPartyVehicle><soabdata:policy
HolderBlame>O</soabdata:policyHolderBlame><soabdata:thirdPartyAcceptedBlame>1</s
oabdata:thirdPartyAcceptedBlame></tns:claim><tns:payload><payload:customerDetail
><payload:firstname/><payload:lastname/><payload:custid/><payload:accnum>O</payl
oad:accnum><payload:hasChild>O</payload:hasChild><payload:policyAmt1>O</payload:
policyAmt1><payload:policyAmt2>O</payload:policyAmt2><payload:netAmt>O</payload:
netAmt><payload:initDate/><payload:markerID>O</payload:markerID><payload:pNest><
payload:recurID/><payload:recurNum/><payload:markerID>O</payload:markerID></payl
oad:pNest><payload:policyDetail><payload:typeID/><payload:polName/><payload:comp
Amt>O</payload:compAmt><payload:markerID>O</payload:markerID></payload:policyDet
ail><payload:vehicleImage/></payload:customerDetail><payload:baseCheckSum>O</pay
load:baseCheckSum><payload:accHistCheckSum>O</payload:accHistCheckSum></tns:payl
oad></tns:createClaimResponse></soapenv:Body></soapenv:Envelope>* Connection #O
to host 9.42.170.230 left intact
* Closing connection #O

3.6 Adding XML schema validation

We successfully provided a Web service facade to the back-end application without modifying the back-end application in any way. As a first step toward robust compliance to SOA, we validate the input data against its XSD:

1. Load the files into DataPower. From the navigation bar, click **ADMINISTRATION** → **File Management**.
2. On the File Management page (Figure 3-67), for the local folder, click **Actions** → **UploadFiles**.

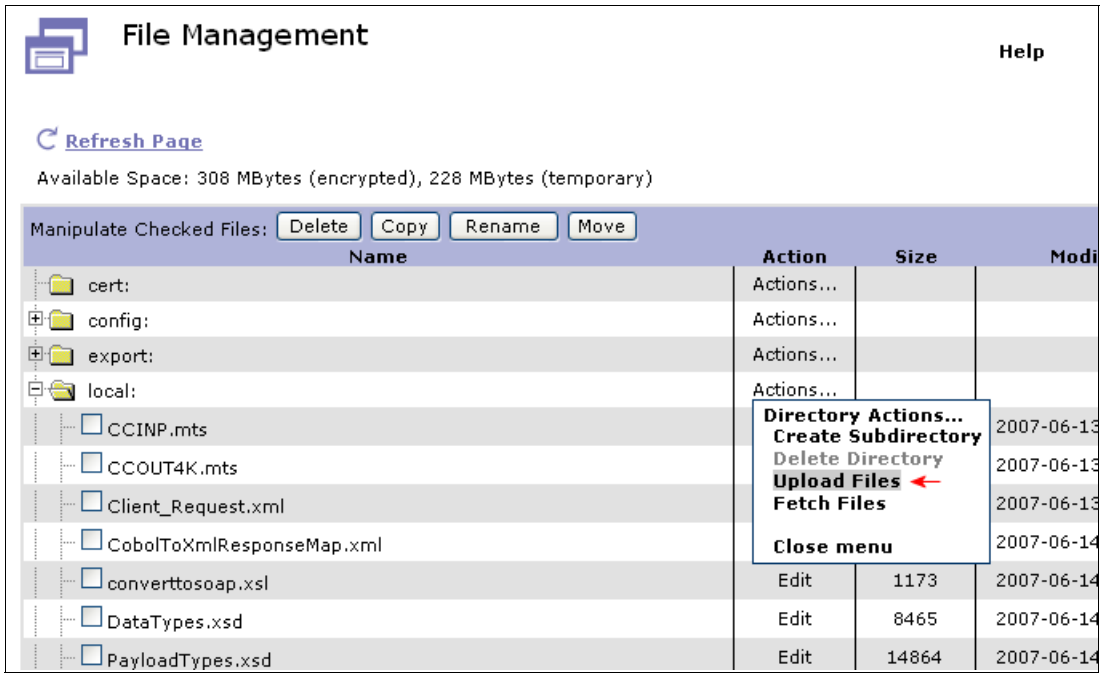


Figure 3-67 Upload Files

3. Browse for the **DataTypes.xsd** file. Click **Attach** → **Upload** → **Continue**. Repeat the same steps for the PayloadTypes.xsd and SOAAssureService.xsd files.
4. Go to the Multi-Protocol Gateway.
5. On the Configure Multi-Protocol Gateway page (Figure 3-68), click **EnablingLegacyApplication**.

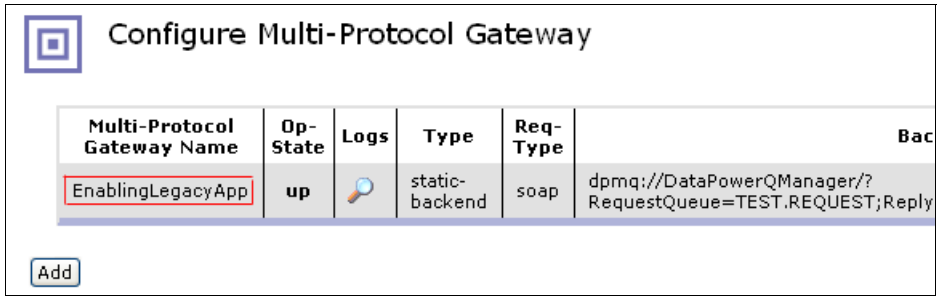


Figure 3-68 Selecting EnablingLegacyApplication

6. On the General page (Figure 3-69), for Multi-Protocol Gateway Policy, select **EnablingLagacyAppPolicy** and click the **...** button.

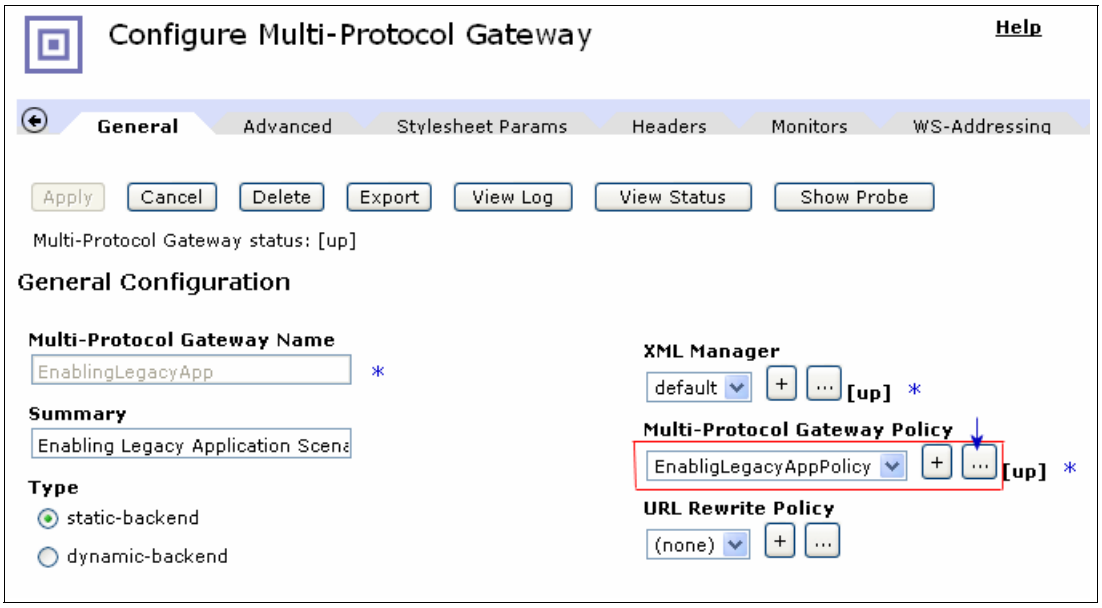


Figure 3-69 Edit EnablingLegacyAppPolicy

7. Drag the **Validate** icon to the rule configuration path (the horizontal line), and drop it as the second icon after the Match (=) icon. Double-click the **Validate** icon (Figure 3-70).

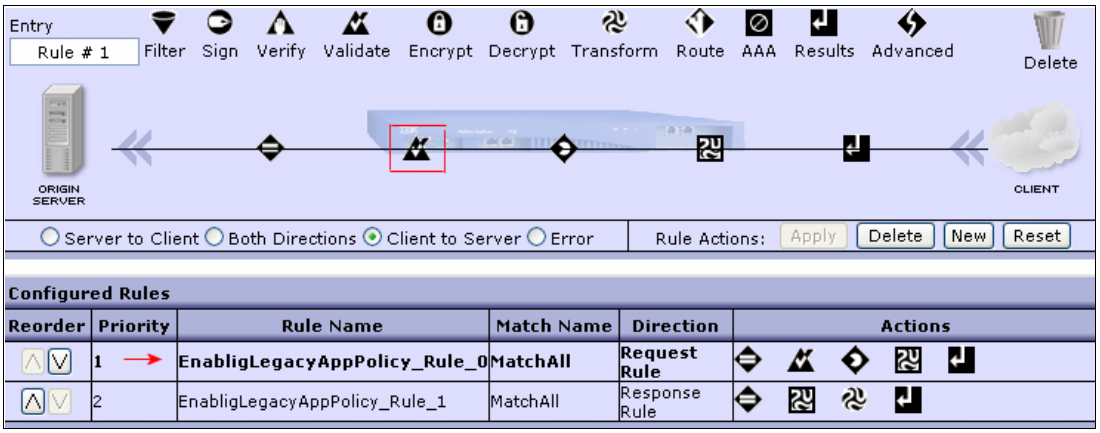


Figure 3-70 Adding the Validate icon

8. On the Configure Validate Action page (Figure 3-71), for Schema Validation Method, select **Validate Document via Schema URL**. Load the SOAAssureService.xsd file. Then click **Done**.

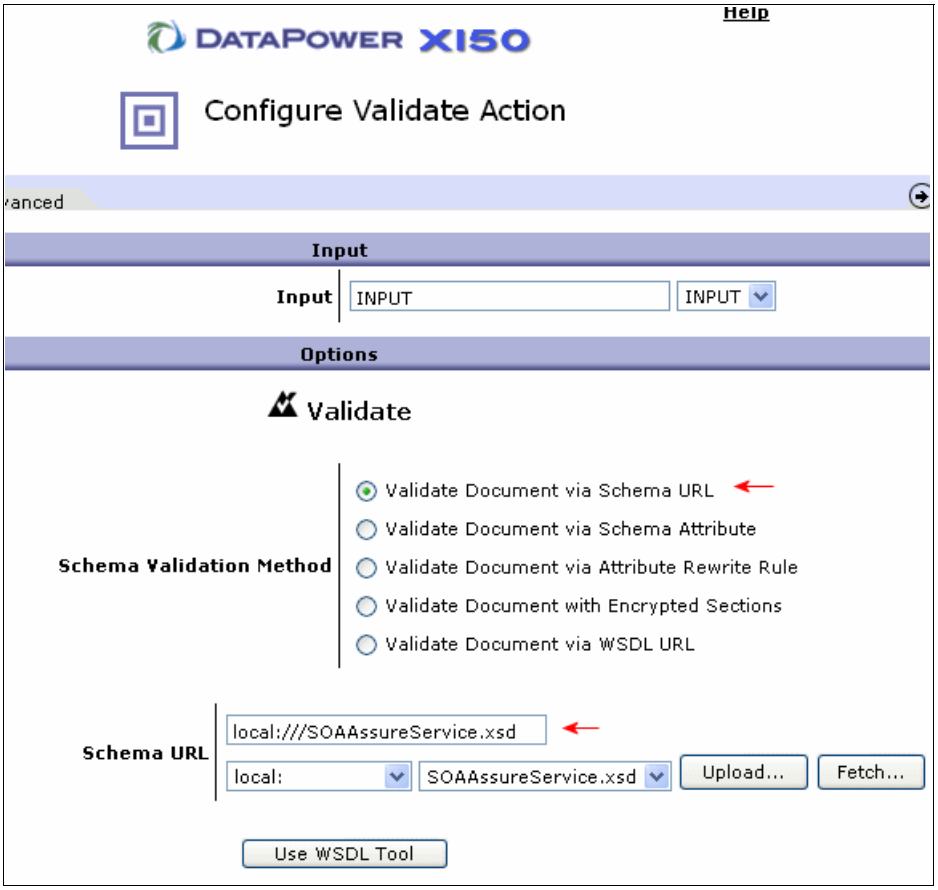


Figure 3-71 Configure Validate Action page

9. On the Configure Multi-Protocol Gateway Policy page, click **Apply** and then click **Close**.

3.7 Running the XML schema validation

To run the validation:

1. On the Configure Multi-Protocol Gateway page (Figure 3-72), in the Request Type section, select **XML** and execute the test.

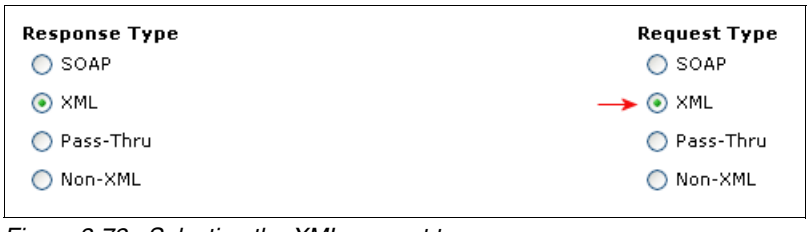


Figure 3-72 Selecting the XML request type

2. Check the validation execution as shown Figure 3-73.

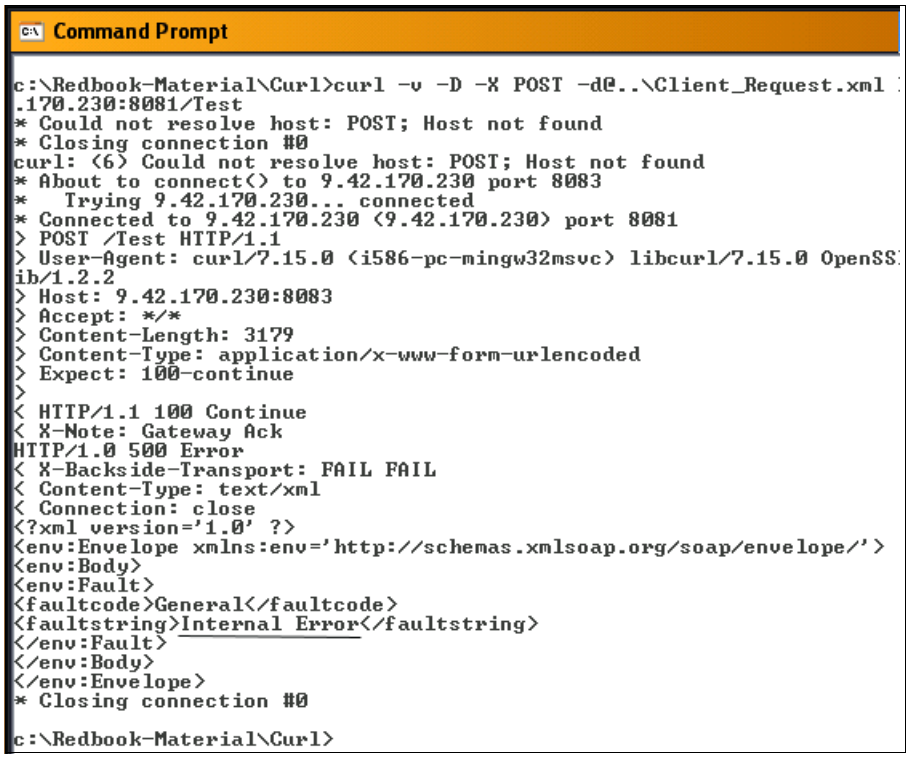


Figure 3-73 Validation execution

3. Check the validation execution in the Probe section (Figure 3-74).

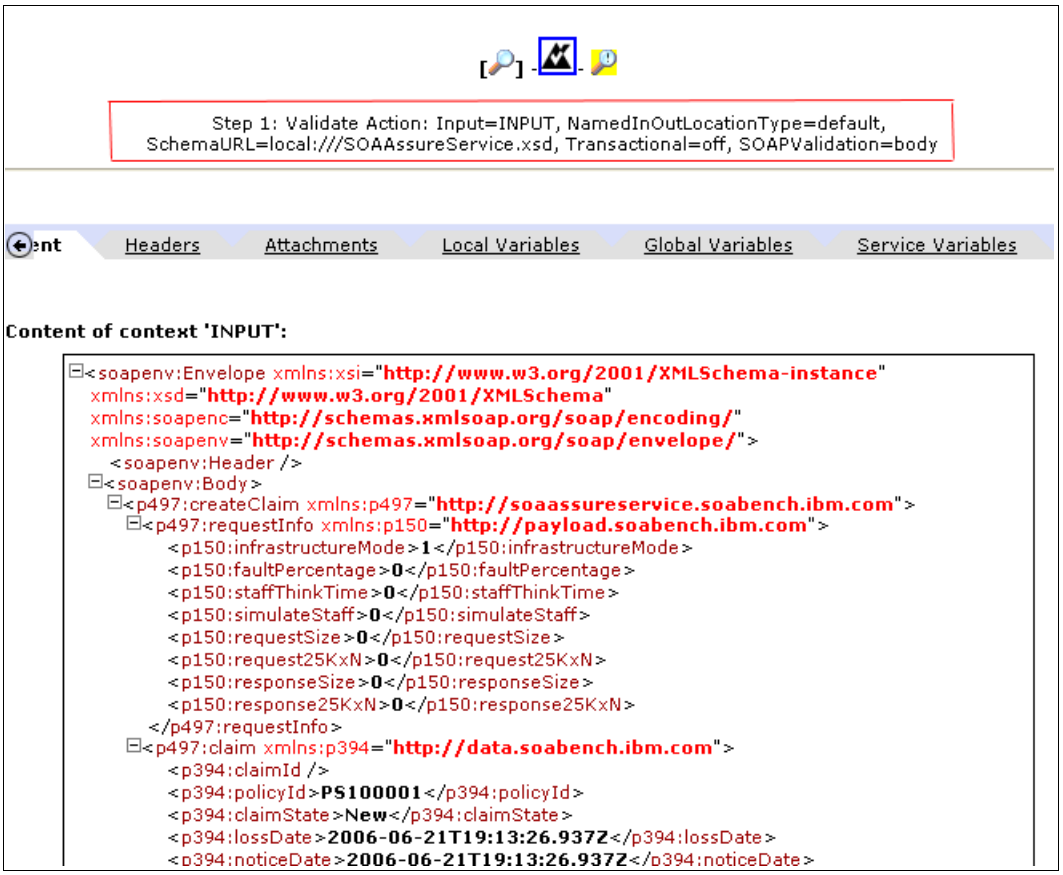


Figure 3-74 Validation Execution from Probe

4. Change the Request Type to **SOAP** (Figure 3-75) and execute the test.

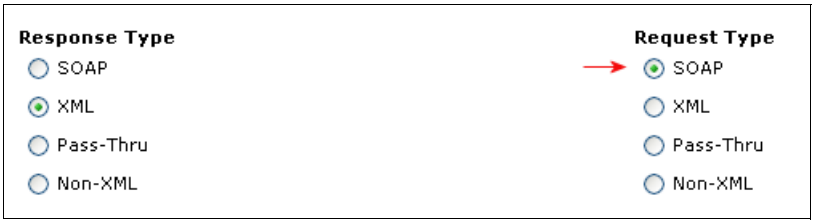


Figure 3-75 Selecting the SOAP request type

3.8 Summary

In this chapter, we illustrated the main steps to build a typical scenario with the DataPower appliance for clients who want to XML-enable arbitrary back-end applications. In this scenario, we went one step further and provided a Web service facade to a back-end COBOL application.

We use this scenario as a building block in the following chapters. We add layers, such as security and authentication, authorization, and audit (AAA), to make the scenario robust in keeping with SOA standards.

4

Securing communication channels with SSL

In this chapter, we explain how you secure and encrypt the communication channels between the DataPower service and the requesting client using Secure Sockets Layer (SSL) by using the secure HTTP protocol, *HTTPS*.

4.1 SSL for transport level security

In order to enable SSL, you must create the following objects in the DataPower appliance:

- Crypto profile
  - Crypto Identification Credentials
  - Crypto Validation Credentials
  - Crypto Key
  - Crypto Certificate
- SSL proxy profile

4.1.1 Crypto profile

A crypto profile identifies a collection of SSL resources that support SSL connections with remote peer devices. To configure a crypto profile:

1. From the navigation bar, select **OBJECTS** → **Crypto Menu** → **Crypto Profile**.
2. On the Crypto Profile Catalog page (Figure 4-1 on page 107), which provides a list of all current crypto profiles, specify the following values:
   a. For Identification Credentials, optionally from the list, select the Identification Credentials Set that is assigned to this crypto profile.

      The Identification Credentials Set provides the public key infrastructure (PKI) certificate or key pair that is used to authenticate the device during the SSL handshake. Retain the default value (none) if are not assigning an Identification Credentials Set to the crypto profile. Click the **+** and **...** buttons to create a new Identification Credentials Set or to edit an Identification Credentials Set.

      In this crypto profile, we use the name **itsoIDCred**.

   b. For Validation credentials, optionally from the list, select the Validation Credentials List that is assigned to this crypto profile. In our crypto profile, we used **itsovalid**.

      Reverse SSL proxy: Assignment of a Validation Credentials List is only meaningful when the crypto profile supports a reverse (or server) SSL proxy. The assignment of a Validation Credentials List to a reverse SSL proxy forces the proxy to require a certificate from all requesting clients.

   c. For Ciphers, identify the symmetric key encryption algorithms that are supported by this crypto profile. Refer to the DataPower reference documentation listed in “Other publications” on page 185 for more information.
   d. For Options, select the appropriate check box to disable support for SSL versions and variants. By default, SSL Versions 2 and 3 are supported, along with Transaction Level Security (TLS) Version 1.
   e. For Send Client CA List, select On to enable transmission of a Client CA List during the SSL handshake. The default is Off, which disables transmission of a Client CA List.

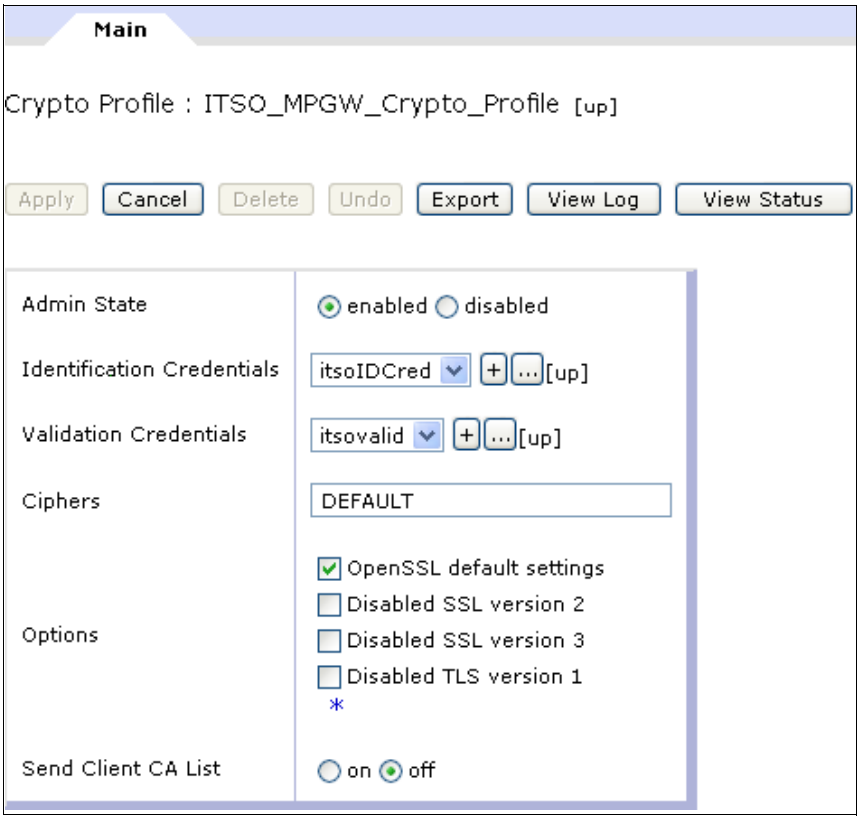


Figure 4-1 Configuration of the crypto profile

Crypto Identification Credentials

A Crypto Identification Credentials set consists of a Crypto Key object and a Crypto Certificate object. An identification credentials set identifies the matched public key cryptography public and private keys that are used by an entity for SSL authentication. An identification credentials set may also be used in document encryption, document decryption, and digital signature operations as shown in the following example:

1. From the navigation bar, select **OBJECTS** → **Crypto** → **Crypto Identification Credentials**. The Crypto catalog (Figure 4-2) lists all current identification credentials sets.

Figure 4-2 Crypto objects menu

2. Click **Add.**
3. On the Configure Crypto Identification Credentials page (Figure 4-3), specify the following values:
   a. For Crypto Key, from the list, select the Crypto Key object that is used by this identification credentials set. You can click the **+** and **...** buttons to create a new Crypto Key object or to edit a Crypto Key object.
   b. For Certificate, from the list, select the Crypto Certificate object that is used by this identification credentials set. You can click the **+** and **...** buttons to create a new Crypto Certificate object or to edit a Crypto Certificate object.
   c. For Intermediate CA Certificate, if necessary, click the **Delete** and **Add** buttons, in conjunction with a list of available Crypto Certificate objects, to establish a verifiable trust chain that consists of one or more Certification Authority (CA) certificates.

      The trust chain provides a linked path from the certificate that is contained in the Identification Credentials Set to a CA that is trusted by a remote device, thus enabling the device to authenticate the certificate.

      Intermediate CA certificates might be necessary when the CA that is signing this certificate is not widely-recognized. If the intermediate CA certificate is also signed by a less recognized CA, an additional intermediate CA certificate might be required for that CA. You can specify as many intermediate certificates as may be required.

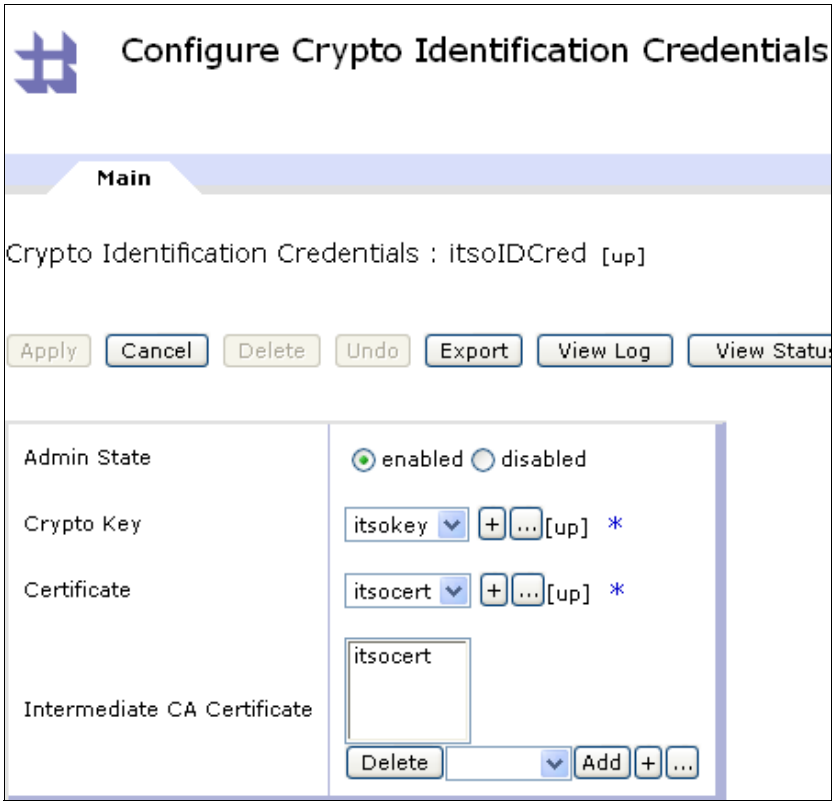


Figure 4-3 Configure Crypto Identification Credentials page

Crypto Validation Credentials

A Crypto Validation Credentials list consists of a list of Crypto Certificate objects. Validation credentials lists are used to validate the authenticity of received certificates and digital signatures. From the navigation bar, select **OBJECTS** → **Crypto** → **Crypto Validation Credentials**. The Crypto catalog provides a list of all current validation credentials.

HTTPS Front Side Handler

After you create a crypto profile, you must create an HTTPS (SSL) Front Side Handler to handle HTTPS protocol communications with multiprotocol gateway clients. To configure the HTTPS Front Side Handler:

1. From the navigation bar, select **OBJECTS** → **Protocol Handlers** → **HTTPS Front Side Handler**. On the HTTPS (SSL) Front Side Handler catalog, you see a list of all current HTTPS Front Side Handlers.
2. Click **Add**.
3. On the HTTPS (SSL) Front Side Handler Configuration page (Figure 4-4), complete the following steps:
   a. Accept the defaults for all parameters, except for SSL Proxy, for which you must create a new SSL proxy object.
   b. For Port Number, enter the port that you want this front-side handler to run.
   c. For Local IP Address, specify the address on which the service listens. The default of 0.0.0.0 indicates that the service is active on all addresses. Click **Select Alias** to use an alias for this value. Local host aliases help to ease migration tasks between machines.
   d. Click **Apply** to write HTTPS Front Side Handler properties to the running configuration. You return to the HTTPS (SSL) Front Side Handler Catalog (Figure 4-4 on page 109), which now lists the newly configured front side handler.
4. Click **Save Config** to save the HTTPS Front Side Handler properties to the persistent startup configuration.

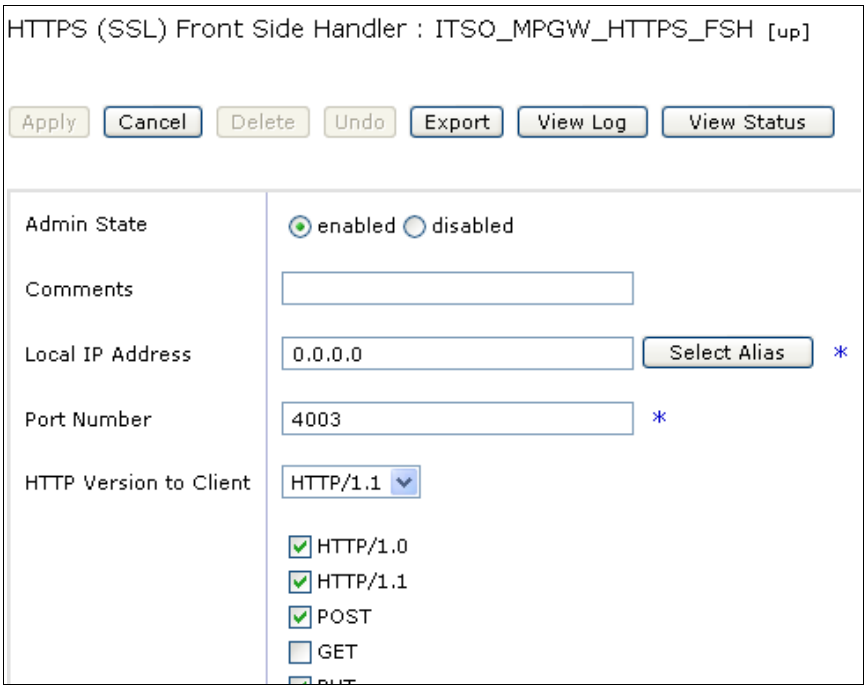


Figure 4-4 Configuring the HTTPS Front Side Handler

4.1.2 SSL proxy profile

An SSL proxy defines the level of service. To define an SSL proxy profile:

1. From the navigation bar, select **OBJECTS** → **Crypto** → **SSL Proxy Profile**.
2. On the Configure SSL Proxy Profile page (Figure 4-5), enter the following information:
   a. For Direction, select one of the following options:
      - *Forward* specifies a mode in which the SSL Proxy functions as an SSL client. In client mode, SSL is used over the device-to-server connection.
      - *Reverse* specifies a mode in which the SSL Proxy functions as an SSL server. In server mode, SSL is used over the device-to-client connection.
      - *Two-way* specifies a mode in which the SSL proxy functions both as an SSL client and as an SSL server. In two-way mode, SSL is used over both the device-to-server connection and over the device-to-client connection.
   b. For Reverse (Server) Crypto Profile, from the list, select the crypto profile that defines the SSL service level between the device (acting as an SSL server) and front-end SSL clients. This setting is relevant when the SSL Proxy Profile operational mode is either reverse or two-way. Select the crypto profile that defines SSL service to front-end clients. Retain the default value (none) if the operational mode is forward. You can use the **+** and **...** buttons to create a new crypto profile or to edit a crypto profile.

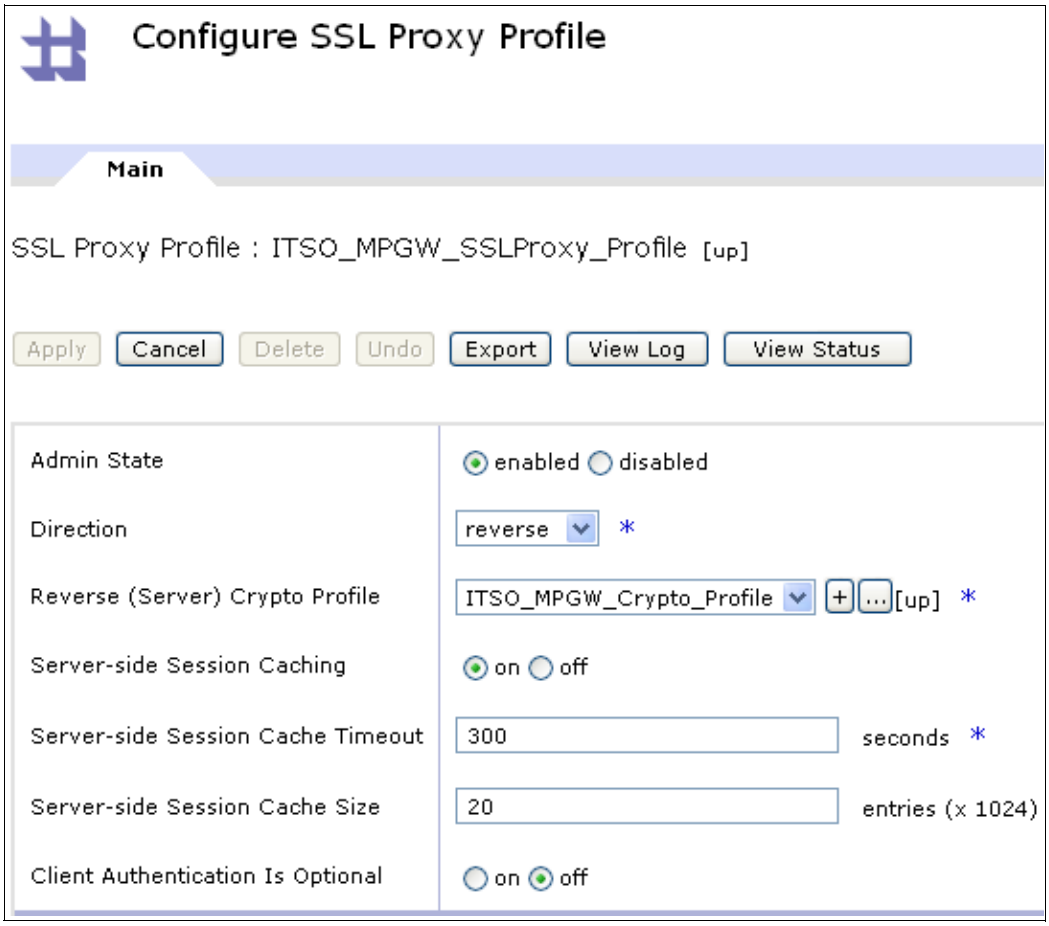


Figure 4-5 Configure SSL Proxy Profile page

c. For Server-side Session Caching, select **On** to enable server side caching or **Off** to disable server side caching.

   By default, SSL server implementations cache SSL session-specific state data, such as the session ID, the peer certificate, compression method, and crypto specs and secrets.

d. For Client Authentication is Optional, this property is meaningful when SSL client authentication is enabled in the server cryptographic profile. When both properties are enabled, SSL client authentication is optional (a setting of **yes**). The request does not fail when there is no client certificate. When disabled (the default setting of **no**), SSL client authentication is required by the application server.

4.1.3 Enabling the Probe for encrypted SSL request messages

SSL provides link-level security and uses the multiprotocol gateway with a specific front side handler. An HTTPS Front Side Handler must be created as described in “HTTPS Front Side Handler” on page 109. The client device that connects to the DataPower appliance, the cURL program in the sample scenario described in this chapter, uses a certificate and a private key to send the encrypted messages.

You can use the following command in this case:

```
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
```

4.2 Summary

In this chapter, we explained how to secure communications channels to the DataPower device by securing the transport communication layer with SSL.

5

Logging capabilities the in DataPower appliance

In this chapter, we discuss the logging capabilities that are provided by the DataPower XI50 appliances. We explain the DataPower logging capabilities and error handling.

5.1 DataPower logging capabilities

In this section, we discuss the logging capabilities of the DataPower appliance.

5.1.1 Log target, category, and level

The DataPower logging capabilities are based on the following objects:

- Log target
- Log category
- Log level

Log target

A log target object captures logging messages that are posted by the different enabled services or objects of a DataPower device. Then it forwards the messages to a specific physical target. This target may be located inside or outside the DataPower device.

Cache or file: Because no hard-disk-based file system is on a DataPower device, do not use a cache or file as log targets. Instead, log outside the DataPower device.

To access the DataPower log targets, select **ADMINISTRATION** → **Manage Log Targets**.

You can filter messages that are destined to a log target by using the following options:

- Event categories

  These categories refer to log category objects, which we explain in the following section.

- Event subscription filters

  By using these filters, only log messages that contain the configured event codes can be written to a log target.

- Event suppression filters

  Suppression filters suppress log messages that contain the configured event codes to be written to a log target.

- Object filters

  With these filters, only log messages that are generated by selected configuration objects can be written to a log target. Object filters are based on object classes, such as processing actions and multiprotocol gateways.

Log category

A log category object represents a type of event. Basically, a DataPower device contains predefined categories. Each category is associated to a specific type of event such as aaa, xslt, mq, auth, and mgmt.

To access the Log Category Configuration page, from the navigation bar, select **ADMINISTRATION** → **Configure Log Categories**.

Log level

A log target must also define a log level. Log levels are hierarchical, with the lowest level (debug) at the bottom of the list and the highest level (emergency) at the top. The following log levels are available:

1. Emergency
2. Alert
3. Critical
4. Error
5. Warning
6. Notice
7. Info
8. Debug

A log target is configured to capture log messages that are at or above the configured level.

Default log target and system logs

For each domain, a default log target of `default-log` exists.

The default system log captures messages that are at or above the default log level. To modify the default system log level, from the Control Panel, click **Troubleshooting**, and select Set Log Level. You set the level as shown in Figure 5-1.

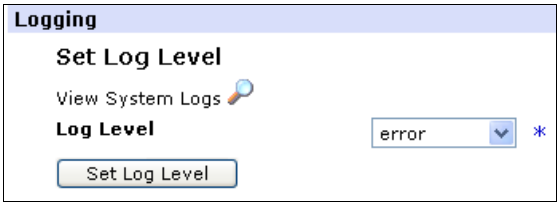


Figure 5-1 Setting the default system log level

System logs (default and others) are displayed on the System Log page. To access this page, select **STATUS** → **System logs** from the navigation bar.

Figure 5-2 shows an example of log entries as displayed on the System Log page.

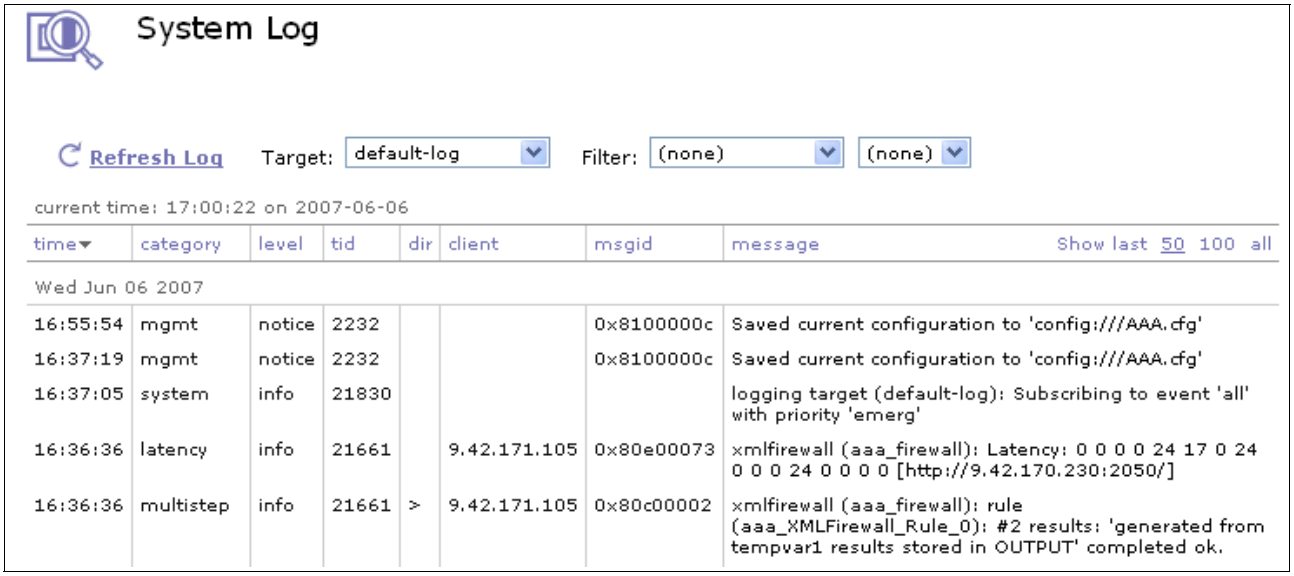


Figure 5-2 System log entries

5.1.2 Configuring a system log

In this section, we discuss a system log configuration for our Multi-Protocol Gateway Service. We add a log target with the following characteristics:

- The log target captures messages that deal with crypto treatments at the notice level.
- The log target must be available only on our multiprotocol gateway service object.
- The name of our log target is crypto_target.
- The log target uses the filestore `logtemp:///crypto.log` to log entries in an XML format.

You can add a DataPower log target by selecting **ADMINISTRATION** → **Manage Log Targets** from the navigation bar. Figure 5-3 shows the Configure Log Target page.

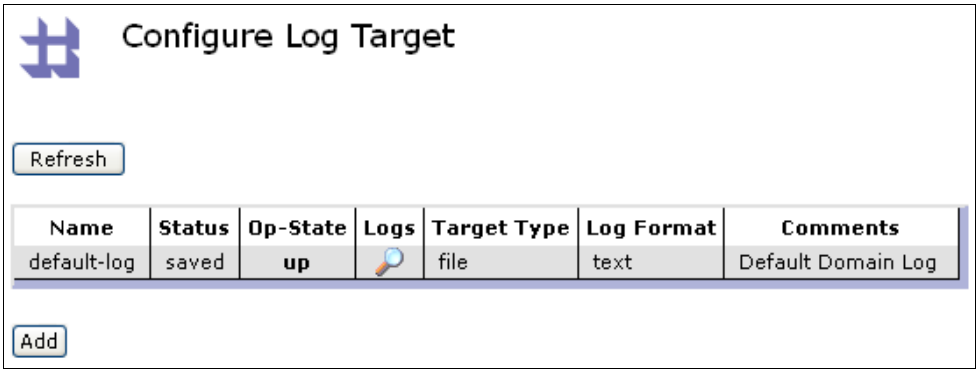


Figure 5-3 Configure Log Target page

Figure 5-4 shows the parameters that we use to create our log target.

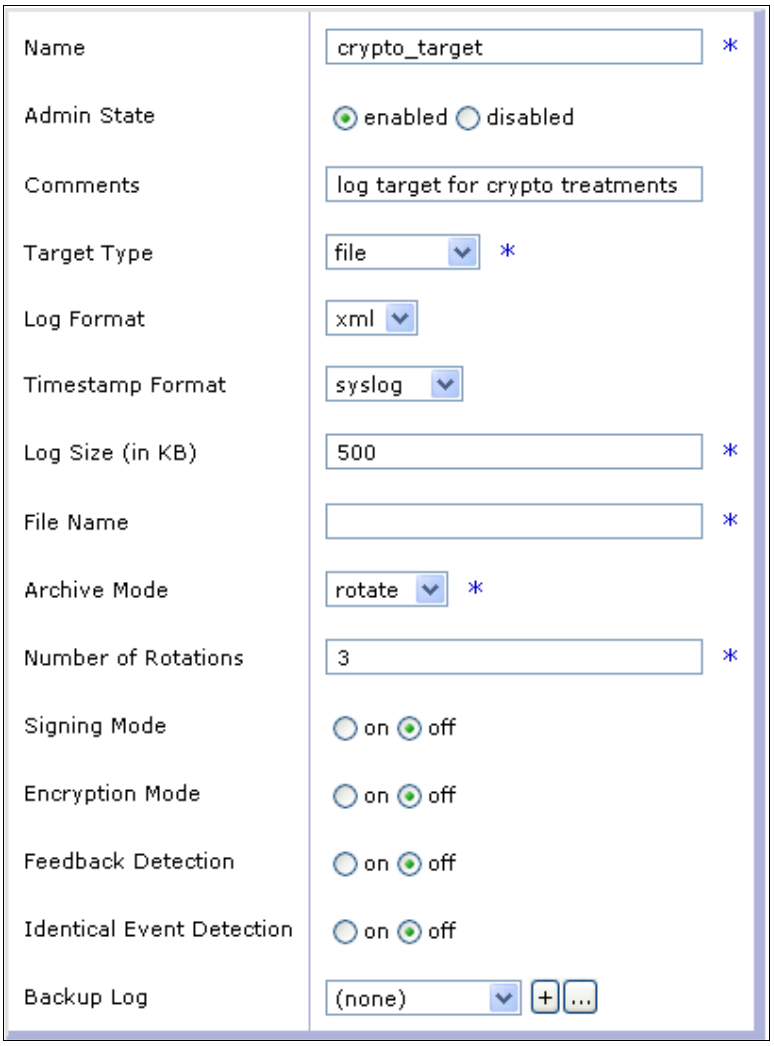


Figure 5-4 Log target creation for crypto treatments

We click the **Object Filters** tab to specify that the crypto_target log target only captures messages from our multiprotocol gateway (Figure 5-5).

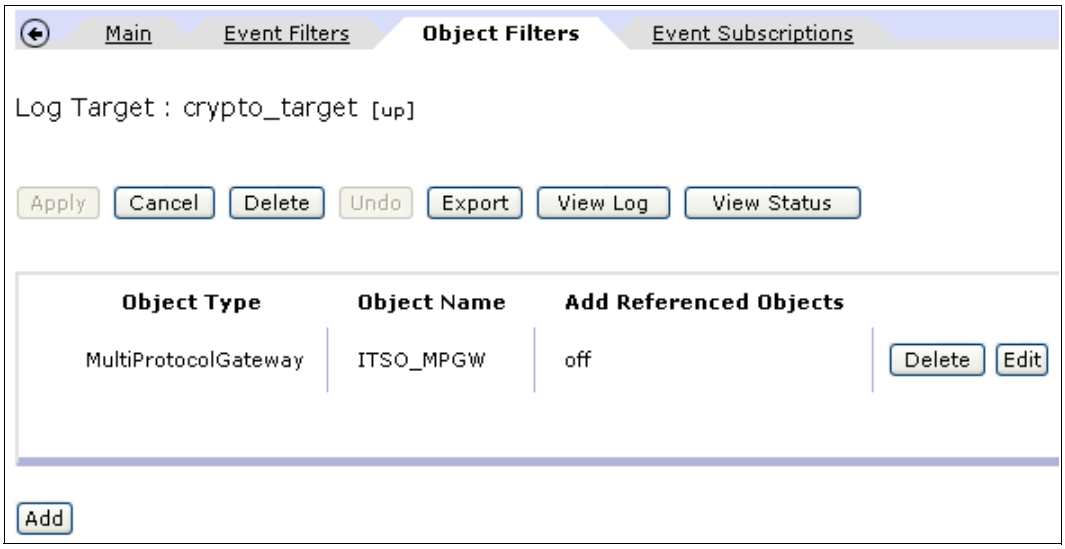


Figure 5-5 Object filters of the log target

Finally we click the **Event Subscriptions** tab to define the category and the log level that we use. Figure 5-6 shows how we define the category and log level.

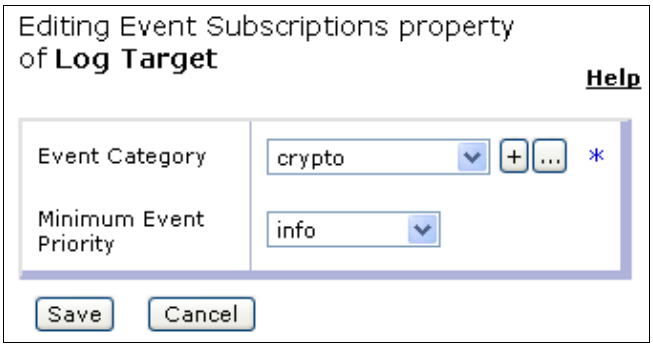


Figure 5-6 Adding a new event subscription property

Figure 5-7 shows the added event category.

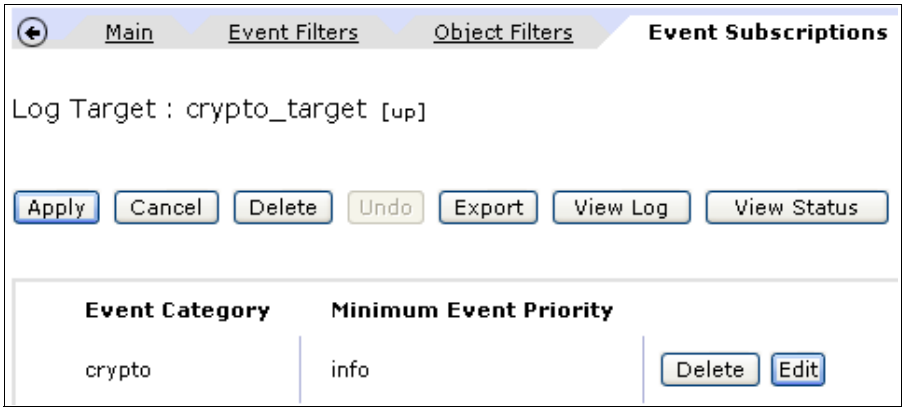


Figure 5-7 Event category of the log target

While sending a request to our multiprotocol gateway, the log entries (samples) shown in Example 5-1 are added to the logtemp:///crypto.log file. These entries concern both request and response processing rules.

Example 5-1 Log entries in the logtemp:///crypto.log file

```
<log-entry serial='0' domain='SecurityProcessing'>
<date>Wed Jun 06 2007</date>
<time utc='1181168414803'>18:20:14</time>
<date-time>2007-06-06T18:20:14</date-time>
<type>crypto</type>
<class>mpgw</class>
<object>ITSO_MPGW</object>
<level num='6'>info</level>
<transaction-type>request</transaction-type>
<transaction>26240</transaction>
<client>9.42.171.105</client>
<code>0x00000000</code>
<file></file>
<message>Ephemeral key decryption succeeded</message>
</log-entry>
...
<message>Data decryption succeeded</message>
...
<message>Signature verification succeeded</message>
...
<message>Signature generation succeeded</message>
...
<message>Ephemeral key generation succeeded</message>
...
<message>Ephemeral key encryption succeeded</message>
...
<message>Data encryption succeeded</message>
...
```

We can see that every step that is associated with the crypto category is logged in the logtemp:///crypto.log file. The log target can use various target types. The following list shows the possible targets:

cache Uses system memory to log entries.

file Writes log entries on the DataPower device flash memory.

nfs Writes log entries to a file on a remote Network File System (NFS) server.

smtp Forwards log entries to e-mail addresses.

snmp Forwards log entries as Simple Network Management Protocol (SNMP) traps issued to all configured recipients of SNMP traps.

soap Forwards log entries as SOAP messages.

syslog Forwards log entries to a remote syslog daemon, over User Datagram Protocol (UDP).

syslog-ng Forwards log entries to a remote syslog daemon, over TCP. In this case, an SSL connection to the syslog server can be used.

5.1.3 Log action

A log action may be used on processing rules. The target of this log must be located outside the DataPower device.

The log action is accessible through an advanced action. You drag an advanced action on a processing rule (request, response or error), double-click this action, and then select the **log** operation (Figure 5-8).

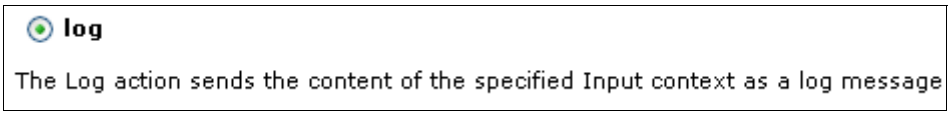


Figure 5-8 Creating a log action

On the Configure Log Action page (Figure 5-9), configure the log action. In the Destination text field, you must declare a target. For Log Type, select the desired category of the target.

For Output, define the log action as **NULL**. In this case, a log action failure does not generate a SOAP fault message. A log action failure might occur, for instance, in case of a problem with the external log target. Moreover, choosing a NULL output implies an asynchronous treatment of the log action, which may be preferred for performance enhancements.

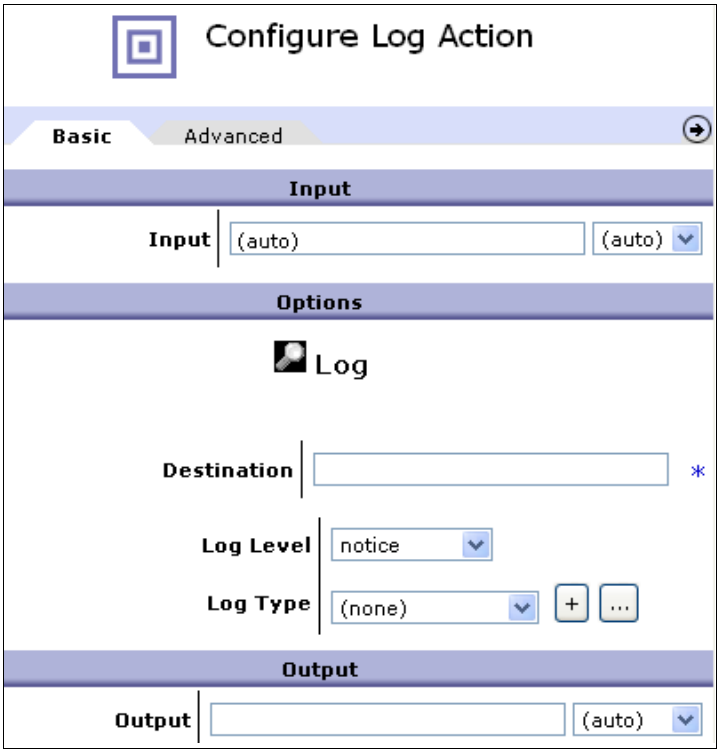


Figure 5-9 Configuring a log action

5.1.4 Logging from a custom template

Log messages can be sent from an Extensible Stylesheet Language (XSL) stylesheet, by using event category and log level as attributes of the `<xsl:message>` element. For a detailed example, see Chapter 6, “XSLT programming” on page 123.

5.2 Error handling

Error handling may be performed with various DataPower objects, which are the on-error processing action and the on-error processing rule.

5.2.1 On-error processing action

An on-error processing action provides the following possibilities:

- Abort or continue the current processing
- Execute a named rule to handle the error

A processing rule might use one or more on-error actions. Each on-error action defines error handling for preceding actions until a new on-error action is found on the same processing rule.

5.2.2 On-error processing rule

An on-error processing rule can be defined in every processing policy. This error rule executes each time an error occurs during request or response processing.

On-error action: The presence of an on-error action prevents an on-error rule from executing.

An on-error processing rule might also use various actions to define a specific and complex error handling. For instance, it is possible to define a transformation action based on a custom template to generate an error message that is related to error and sub-error codes that are generated by the DataPower appliance.

Refer to Chapter 6, “XSLT programming” on page 123, for an example of using an XSL stylesheet to create an error message.

5.3 Summary

Logging outside a DataPower device is definitely a best practice, since there is no file system in a DataPower device.

6

XSLT programming

In this chapter, we describe the XML Stylesheet Language Transformation (XSLT) programming enhancements that are provided in the DataPower appliance thanks to the following extension modules:

- Extended XSLT (EXSLT)

  This module consists of a set of extension functions to common XSLT1.0 capabilities.

- DataPower extension functions and elements

  This module contains a set of functionalities and elements (or tags) dedicated to DataPower treatments.

Both EXSLT and DataPower extensions can be used in XSL stylesheets in order to achieve specific treatments. In this chapter, we introduce these basic extension functions and elements, and we provide practical code examples that can be easily reused. We discuss the following topics:

- XSL stylesheet namespace requirements to use EXSLT and DataPower extension functions and elements
- Basic examples on using both EXSLT and DataPower extension functions and elements
- Examples of custom templates implementation, based on the scenario described in Chapter 2, “Getting started” on page 17
- Examples of custom template implementations, based on a specific scenario
- XSLT debugging and logging in the DataPower appliance

More information: The goal of this chapter is to describe the way in which you can use the enhancements of the extension function and provided elements in the context of a specific XSL stylesheet implementation on the DataPower appliance.

For details about the DataPower domain, SOAP clients, and requests that are used to configure and test these examples, see Appendix A, “XSL programming issues” on page 179.

6.1 XSL stylesheet namespace requirements

EXSLT and DataPower extension functions or elements use specific namespaces that must be added in the <xsl:stylesheet> element of a custom template. We describe the different possible values of these namespaces and the way in which they are declared.

6.1.1 Namespace declarations for DataPower extensions

In this section, we discuss the namespace declarations for DataPower extension functions and elements.

Namespace in case of an extension element use

Example 6-1 shows the namespace declaration that you must use in an XSL stylesheet that implements at least one DataPower extension element.

Example 6-1 template1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
exclude-result-prefixes="dp">

<!-- *** HERE COMES THE XSL STYLESHEET CODE *** -->

</xsl:stylesheet>
```

The xsl:stylesheet element may use other complementary namespaces in addition to these required declarations.

Namespace prefix: The namespace prefix that we use in our example is dp. We could have defined another value. This prefix value must be used in every reference to an extension element.

The extension-element-prefixe attribute is a space-separated list of namespace prefixes that are related to extension elements. In the previous example, only dp belongs to this list.

The exclude-result-prefixes attribute is a space-separated list of namespace prefixes that must not be included in the output content. In the previous example, only dp belongs to this list.

Both the extension-element-prefixes and exclude-result-prefixes attributes are optional.

Namespace in case of an extension function use

The namespace used in case of DataPower extension function use is strictly the same as the one that is used for an extension element. The only difference resides in the fact that the extension-element-prefixe and exclude-result-prefixes attributes are useless because we do not need to implement an element, but rather to implement a function.

Therefore, the namespace declaration shown in Example 6-2 must be used in an XSL stylesheet that needs reference to at least one extension function.

Example 6-2 template2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:dp="http://www.datapower.com/extensions">

<!-- *** HERE COMES THE XSL STYLESHEET CODE *** -->

</xsl:stylesheet>
```

The <xsl:stylesheet> element may use other complementary namespaces in addition to this required declaration.

Namespace prefix: The namespace prefix that we use in our example is dp. Another value could have been defined. This prefix value must be used in every reference to an extension function.

6.1.2 Namespace declarations for EXSLT extension functions

EXSLT contains a wide range of functionality that is grouped by module. Each module uses its own namespace. These namespaces must be added to the XSL stylesheet only if at least one extension function of a module is used.

For DataPower XSL stylesheet implementation, Table 6-1 shows the main modules and their related namespaces and functions.

Table 6-1 EXSLT extension functions

Module	Namespace	Functions
Common	xmlns:exsl="http://exslt.org/common"	object-type()
Date and time	xmlns:date="http://exslt.org/dates-and-times"	add() add-duration() date() date-time() day-abbreviation() day-in-month() day-in-week() day-in-year() day-name() day-of-week-in-month() difference() duration() hour-in-day() lap-year() minute-in-hour() month-abbreviation() month-in-year() month-name() second-in-minute() seconds()

Module	Namespace	Functions
date and time (description continued)	xmlns:date="http://exslt.org/dates-and-times"	time() week-in-month() week-in-year() year()
Regular expression	xmlns:regExp="http://exslt.org/regular-expressions"	match() replace() test()
Set	xmlns:set="http://exslt.org/sets"	difference() distinct() has-same-node() intersection() leading() trailing()
String	xmlns:str="http://exslt.org/strings"	concat() decode-uri() encode-uri() padding() split() tokenize()

For instance, Example 6-3 shows the namespace declaration to use if you must use both the string and date modules.

Example 6-3 template3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:str="http://exslt.org/strings"
xmlns:date="http://exslt.org/dates-and-times">

<!-- *** HERE COMES THE XSL STYLESHEET CODE *** -->

</xsl:stylesheet>
```

For details about the extension functions, refer to the latest edition of the *DataPower Extension Functions Catalog* on the Web at the following address:

https://www14.software.ibm.com/webapp/iwm/web/reg/download.do?source=swg-datapower&S_PKG=xi50_9003_s2&dlmethod=http

6.2 Using namespaces

In this section, we provide basic examples on how to use namespaces with extension functions and elements.

6.2.1 A DataPower extension element

In the first template (Example 6-4), we introduce the use of the DataPower extension element <set-variable>, which you might implement to assign a value to a multiple-step variable.

Example 6-4 template4.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:dp="http://www.datapower.com/extensions"
extension-element-prefixes="dp"
exclude-result-prefixes="dp">

<xsl:template math="/">
<dp:set-variable name="'var://context/itso/myItsoVariable'" value="'Hello'"/>
<xsl:apply-templates select="*"/>
</xsl:template>

<!-- *** HERE COMES THE FURTHER XSL STYLESHEET CODE *** -->

</xsl:stylesheet>
```

The result of this XSL stylesheet is to assign a value of `Hello` to the variable `var://context/itso/myItsoVariable`.

Namespace prefix: The namespace prefix `dp` is declared in the `xsl:stylesheet` element and used while implementing the `set-variable` element.

6.2.2 A DataPower extension function

In this second template (Example 6-5), we introduce the use of the DataPower extension function `variable()`, which you can use to retrieve the value of a specified variable. We assume that a multi-step variable `var://context/itso/myItsoVariable` has previously been set with the value `Hello`.

Example 6-5 template5.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:dp="http://www.datapower.com/extensions">

<xsl:output method="xml" indent="yes" encoding="UTF-8"/>

<xsl:template match="/">
<result>
<xsl:value-of select="dp:variable('var://context/itso/myItsoVariable')"/>
</result>
</xsl:template>

</xsl:stylesheet>
```

Example 6-6 shows the result of this XSL stylesheet.

Example 6-6 template6.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
Hello
</result>
```

We create a <result> output element in which we place a text value.

Namespace prefix: The namespace prefix dp is declared in the xsl:stylesheet element and is used while implementing the variable() function.

6.2.3 An EXSLT extension function

In this third template, we introduce the use of the EXSLT extension function date-time(), which belongs to the date and time module. You might use this function to get the current date and time, as a string value, with a specific format as shown in Example 6-7.

Example 6-7 template7.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:date="http://exslt.org/dates-and-times">

<xsl:output method="xml" indent="yes" encoding="UTF-8"/>

<xsl:template match="/">
<result>
<xsl:value-of select="date:date-time()"/>
</result>
</xsl:template>

</xsl:stylesheet>
```

Example 6-7 show the result of this XSL stylesheet.

Example 6-8 template8.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
2007-06-04T13:21:06
</result>
```

We create a <result> output element in which we place the current date and time.

Namespace prefix: The namespace prefix date is declared in the xsl:stylesheet element and used while implementing the date-time() function.

6.3 Example 1: AAA policy based on custom templates

In this example, we present a full authentication, authorization, and audit (AAA) processing policy based on custom templates. This example is based is the scenario that we described in Chapter 2, “Getting started” on page 17. This scenario is based on the Web service proxy.

For details about the AAA capabilities of the DataPower appliance, see *IBM WebSphere DataPower SOA Appliances Part II: Authentication and Authorization*, REDP-4364.

6.3.1 Objectives and presentation

The goal of this first example is to answer the following questions:

- How can an identity be extracted from an incoming request?
- How is identity data transmitted to the authentication step of a AAA policy?
- How can authentication be performed by using an XML file?
- How is data transmitted to the authorization step?
- How can authorization be performed by using an XML file?

The following XSL stylesheets are used:

- For extract identity, the itso.aaa-extract_info.xsl stylesheet is used.
- For authentication, the itso.aaa-authenticate.xsl stylesheet is used.
- For authorization, the itso.aaa-authorize.xsl stylesheet is used.

These custom templates are loaded in the local directory of the DataPower device. Neither credential nor resource mapping is used in this example.

The XML file AAA-repository.xml is the authority that is used to check both credentials and resources. This file contains the following information:

- Authenticated users, described in an `<authenticate>` element with tags:
  - `<user>`: The user’s name who requests the service. Each user is identified by a unique identifier (id attribute).
  - `<password>`: The user’s password.
  - `<client-ip>`: The user’s IP address.
- Local names of the possible requests, linked to user IDs

  If a user is included in a `<request>` element, then this user is authorized to request the related service.

Example 6-9 shows the AAA-repository XML file content.

Example 6-9 AAA-repository XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<aaa-data>
<!-- Authentication data-->
<authenticate>
<user id="001">fred</user>
<password>flintstone</password>
<client-ip>9.42.171.105</client-ip>
</authenticate>
<authenticate>
<user id="002">greg</user>
<password>mccarty</password>
```

```
<client-ip>9.42.171.175</client-ip>
</authenticate>

<!-- Authorization data-->
<authorize>
<request name="getPrime">
<user id="001" start-date="100"/>
</request>
</authorize>
<authorize>
<request name="getOdd">
<user id="001" start-date="200"/>
<user id="002" start-date="100"/>
</request>
</authorize>
</aaa-data>
```

The possible users are *fred flintstone* and *greg mccarty*. They are authenticated if they send a request from a client whose IP address is one of those indicated in `<client-ip>` tags. They both can use the `getOdd` service, but only frank (`id=001`) is allowed to use the `getPrime` service.

The `start-date` attribute indicates the date, in milliseconds (since January 1, 1970), from which users are authorized to submit the appropriate request.

6.3.2 DataPower configuration

Figure 6-1 shows the request rule where a AAA action will be added.

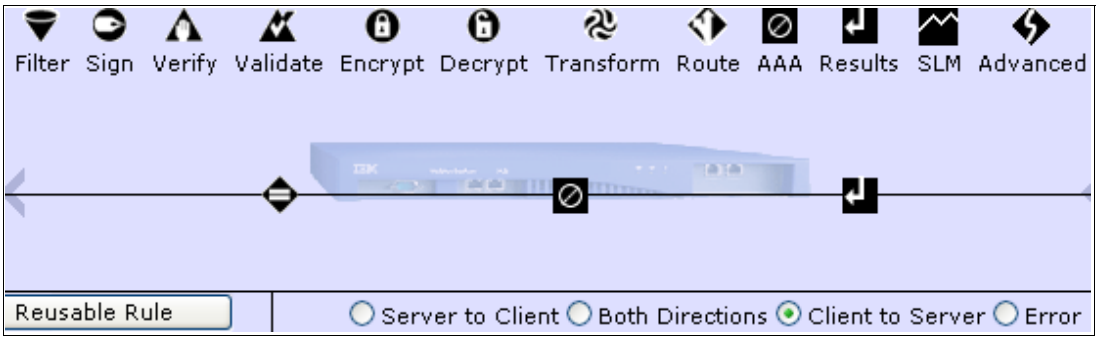


Figure 6-1 Request rule with a AAA action

The AAA policy has the following properties:

- Identity extraction is completed by using the custom template `itso.aaa-extract_info.xsl`.
- Authentication is completed by using the custom template `itso.aaa-authenticate.xsl`.
- Resource extraction is based on the local name of the incoming request.
- Authorization is completed by using the custom template `itso.aaa-authorization.xsl`.
- Audit is completed by using the default values that are proposed while creating the AAA policy.

6.3.3 Incoming SOAP message

The incoming request is a SOAP message that uses WS-Security UsernameToken to define a required user name and password, as shown in Example 6-10.

Example 6-10 getPrime_EX1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:q0="http://com.itso"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-sec
ext-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>
<q0:getPrime>
<numDigits>24</numDigits>
</q0:getPrime>
</soapenv:Body>
</soapenv:Envelope>
```

6.3.4 XSL stylesheet details

In this section, we introduce the three steps of the AAA policy that use custom templates:

1. Identity extraction
2. Authentication
3. Authorization

Identity extraction

The `itso.aaa-extract_info.xsl` custom template (Example 6-11 on page 132) extracts the following values:

- X-Client-IP, from the HTTP header

> **Note:** The HTTP header X-Client-IP value is extracted by using the `dp:http-request-header()` extension function.

- User name and password, from the incoming SOAP header

Example 6-11 itso.aaa-extract_info.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<!--
+
|*****************************************************************************
|*** Author: ITSO
|*** file: itso.aaa-extract_info.xsl
|*** Description: This XSL is responsible for extracting AAA information from the incoming
message
|*** Revision : 1.0 : initial version
|*****************************************************************************
+ -->
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
    xmlns:itso="http://com.itso"
    extension-element-prefixes="dp      "
    exclude-result-prefixes="dp itso">

<!--  +
   |*********************************************************************
   |*** Matching Template
   |*** Element: /*[local-name()='Envelope']/*[local-name()='Body']/*[1]
   |*********************************************************************
   + -->
<!--  Match the first element contained in the body of the soap message
   Input soap message has already been validated when this matching is processed
   -->
 <xsl:template match="/*[local-name()='Envelope']/*[local-name()='Body']/*[1]">

  <!-- Get value of the HTTP header field XClient-IP -->
  <xsl:variable name="vClientIP" select="dp:http-request-header('X-Client-IP')"/>

  <!-- Get local name of the current service request and store it in DP transaction -->
  <xsl:variable name="vRequestName" select="local-name()"/>
  <dp:set-variable name="'var://context/txn-info/requestName'" value="string($vRequestName)"/>

<!-- ***
   Here we build the output of the identity extraction.
   Next step is authentication, that is going to use these data to perform user authentication
   ***
  -->
  <user>
   <!-- Get value of Username element contained in SOAP header's WS-Security token -->
   <xsl:value-of
select="/*[local-name()='Envelope']/*[local-name()='Header']/*[local-name()='Security']/*[local-
name()='UsernameToken']/*[local-name()='Username']"/>
  </user>
  <password>
   <!-- Get value of Password element contained in SOAP header's WS-Security token -->
   <xsl:value-of
select="/*[local-name()='Envelope']/*[local-name()='Header']/*[local-name()='Security']/*[local-
name()='UsernameToken']/*[local-name()='Password']"/>
  </password>
  <clientIP>
```

```
   <xsl:value-of select="$vClientIP"/>
  </clientIP>

 </xsl:template>
</xsl:stylesheet>
```

After the user name, password, and X-Client-IP are extracted, we build XML content, which is used to transmit the three values to the authentication step. Figure 6-2 shows the XML content of the output of the identity extraction.

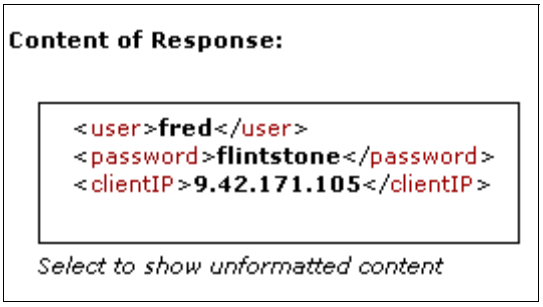


Figure 6-2 Output of the identity extraction step

Authentication

Figure 6-3 shows the XML content of the input of the authentication step.

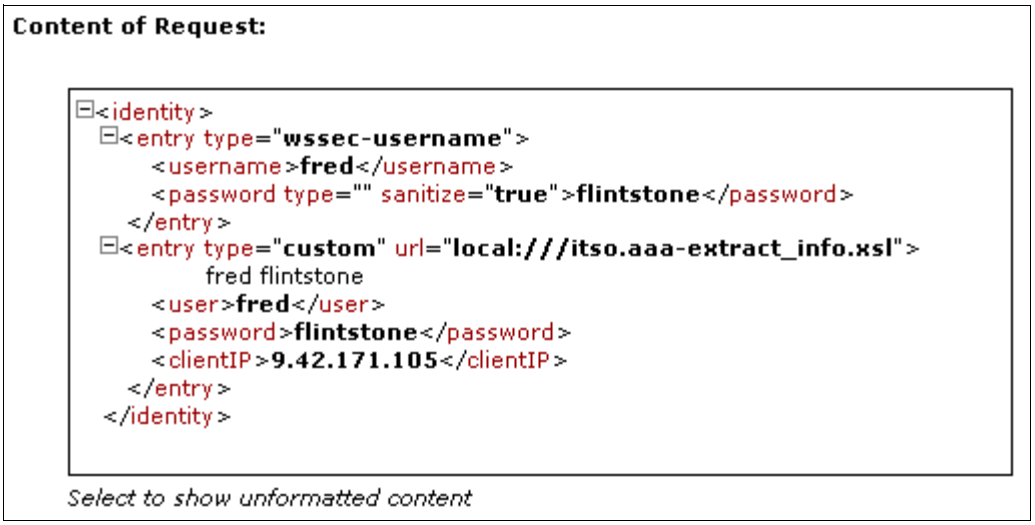


Figure 6-3 Input of the authentication step

In the `itso.aaa-authenticate.xsl` template (Example 6-12 on page 134), we see the extracted information and verify that an entry exists in the AAA-repository XML file, for the triplet {user,password,clientIP}. We recover the value of the `id` attribute of a `<user>` element, in the `<authenticate>` section of the AAA-repository XML file, as implemented in the XSL stylesheet.

Example 6-12 itso.aaa-authenticate.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<!--
+ |****************************************************************************
|*** Author: ITSO
|*** file: itso.aaa-authenticate.xsl
|*** Description: This XSL is responsible for authentication. Data used are those transmitted by
the
|*** identity extraction step. Here is an example of the incoming XML content:
|*** <identity>
|***  <entry type='custom'>
|***   <user>...</user>
|***   <password>...</password>
|***   <clientIP>...</clientIP>
|***  </entry>
|*** </identity>
|*** Revision : 1.0 : initial version
|****************************************************************************
+ -->

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
    xmlns:itso="http://com.itso"
    extension-element-prefixes="dp"
    exclude-result-prefixes="dp itso">

 <!-- Import XSL stylesheet which provides debugging enhancements-->
 <xsl:import href="local:///itso-common-debug.xsl"/>

 <!--+
  |*********************
  |*** Matching Template
  |*** Element: ROOT
  |*********************
  + -->
 <xsl:template match="/">
  <xsl:apply-templates select="identity"/>
 </xsl:template>

 <!--+
  |*********************
  |*** Matching Template
  |*** Element: identity
  |*********************
  + -->
 <xsl:template match="identity">
  <xsl:apply-templates select="entry"/>
 </xsl:template>

 <!--+
  |*********************
  |*** Matching Template
  |*** Element: entry
  |*********************
```

```
 + -->
<xsl:template match="entry[@type='custom']">

 <!-- Get login value -->
 <xsl:variable name="vUser" select="user"/>

 <!-- Get password value -->
 <xsl:variable name="vPassword" select="password"/>

 <!-- Get client-IP value -->
 <xsl:variable name="vClientIP" select="clientIP"/>

 <!-- Call the 'authenticate' named template that performs the authentication step -->
 <xsl:call-template name="authenticate">
  <xsl:with-param name="aUser">
   <xsl:value-of select="$vUser"/>
  </xsl:with-param>
  <xsl:with-param name="aPassword">
   <xsl:value-of select="$vPassword"/>
  </xsl:with-param>
  <xsl:with-param name="aClientIP">
   <xsl:value-of select="$vClientIP"/>
  </xsl:with-param>
 </xsl:call-template>

</xsl:template>

<!--+
 |************************
 |*** Named Template
 |*** Name: authenticate
 |************************
 + -->
<!-- Verify authentication for a specific {user,password,clientIP} triplet -->
<xsl:template name="authenticate">
 <xsl:param name="aUser"/>
 <xsl:param name="aPassword"/>
 <xsl:param name="aClientIP"/>

 <!-- define a variable for AAA repository XML file -->
 <xsl:variable name="vAuthenticationFile" select="document('local:///AAA-repository.xml')"/>

 <!-- Here we perform authentication -->
 <xsl:choose>
  <xsl:when test="$vAuthenticationFile != ''">
   <xsl:variable name="vUserID"
select="$vAuthenticationFile/aaa-data/authenticate[string(user)=$aUser][string(password)=$aPassw
ord][contains(string(client-ip),$aClientIP)]/user/@id"/>
    <!-- ***
      If credentials match then we want to output something to let the AAA framework know that
authentication succeeded.
      If credentials do not match, we output nothing and that means 'authentication failure' to
the AAA framework.
      *** -->
```

```
    <!-- Debug value of vUserID-->
    <xsl:call-template name="debug">
     <xsl:with-param name="aLabel">userid</xsl:with-param>
     <xsl:with-param name="aMessage"><xsl:value-of select="$vUserID"/></xsl:with-param>
    </xsl:call-template>

    <xsl:choose>
     <xsl:when test="string($vUserID) != ''">
      <userid>
       <xsl:value-of select="$vUserID"/>
      </userid>
     </xsl:when>
     <!-- Add error handling in xsl:otherwise condition-->
     <xsl:otherwise/>
    </xsl:choose>
   </xsl:when>
   <!-- Add error handling in xsl:otherwise condition-->
   <xsl:otherwise/>
  </xsl:choose>

 </xsl:template>

</xsl:stylesheet>
```

If a user ID is found in the AAA-repository XML file, then the user is authenticated. In this case, we create an XML element, `<userid>...</userid>`, in which we insert the value of the current user identifier. This XML content is then transmitted to the authorization step.

Figure 6-4 shows the XML content of the output of our successful authentication step.

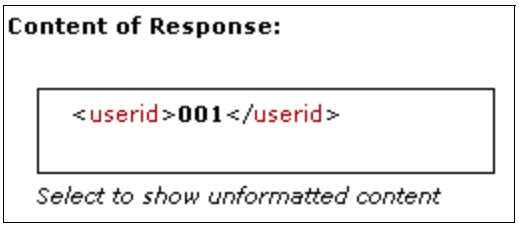


Figure 6-4 Output of the authentication step

This template does not manage any specific error handling. We discuss this point in 6.8, “Example 6: On-error handling using custom templates” on page 160.

Authorization

Figure 6-5 shows the XML content of the input of the authorization step.

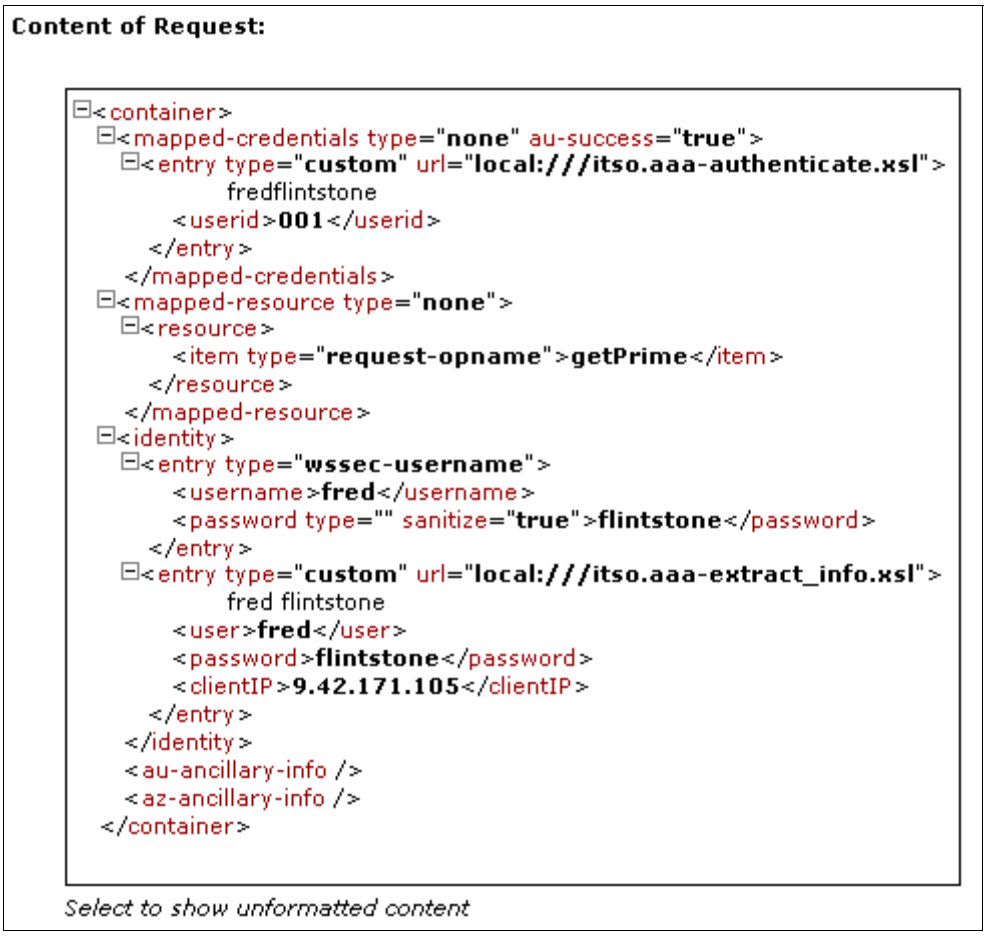


Figure 6-5 Input of the authorization step

In the `itso.aaa-authorize.xsl` template, we use input data from the authentication and resource extraction to verify if the current user is allowed to access the requested resource. The input of the authentication step is embedded in the XML content that contains both the mapped credentials and resource.

Moreover, we add a control based on a start date. We allow a user to request a resource from a specific date, expressed in milliseconds. The user is not allowed to request a resource before the request's start date.

If authorization is approved, the output consists of an `<approved>` element to let the DataPower AAA framework know that authorization step is successful. We use another element, such as `<declined>`, to indicate an authorization failure, as implemented in Example 6-13 on page 138.

Example 6-13 itso.aaa-authorize.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<!--
+ |*****************************************************************************
|*** Author: ITSO
|*** file: itso.aaa-authorize.xsl
|*** Description: This XSL is responsible for AAA authorization. Data used are those transmitted
by the
|*** previous authentication step. Here is an example of the incoming XML content:
|*** <container>
|***  <mapped-credentials>
|***   <entry>
|***    <userid>...</userid>
|***   </entry>
|***  </mapped-credentials>
|*** </container>
|*** Revision : 1.0 : initial version
|*****************************************************************************
+ -->

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
    extension-element-prefixes="dp"
    exclude-result-prefixes="dp">

 <!-- Import XSL stylesheet which provides debugging enhancements-->
 <xsl:import href="local:///itso-common-debug.xsl"/>

 <!--+
  |*********************
  |*** Matching Template
  |*** Element: ROOT
  |*********************
  + -->
 <xsl:template match="/">
  <xsl:apply-templates select="container"/>
 </xsl:template>

 <!--+
  |***********************
  |*** Matching Template
  |*** Element: container
  |***********************
  + -->
 <xsl:template match="container">
  <xsl:apply-templates select="mapped-credentials"/>
 </xsl:template>

 <!--+
  |******************************
  |*** Matching Template
  |*** Element: mapped-credentials
  |******************************
  + -->
```

```
 <xsl:template match="mapped-credentials">
  <xsl:apply-templates select="entry"/>
 </xsl:template>

 <!--+
  |*********************
  |*** Matching Template
  |*** Element: entry
  |*********************
  + -->
 <xsl:template match="entry">

  <!-- Get userid value -->
  <xsl:variable name="vUserId" select="userid"/>

  <!-- Get local name of the request element, that is the extracted resource -->
  <xsl:variable name="vRequestName"
select="dp:variable('var://context/WSM/resource/extracted-resource')"/>

  <!-- Perform authorization -->
  <xsl:call-template name="authorize">
   <xsl:with-param name="aUserId">
    <xsl:value-of select="$vUserId"/>
   </xsl:with-param>
   <xsl:with-param name="aRequestName">
    <xsl:value-of select="$vRequestName"/>
   </xsl:with-param>
   <xsl:with-param name="aDate">
    <xsl:value-of select="dp:time-value()"/>
   </xsl:with-param>
  </xsl:call-template>

 </xsl:template>

 <!--+
  |*********************
  |*** Named Template
  |*** name: authorize
  |*********************
  + -->
 <xsl:template name="authorize">
  <xsl:param name="aUserId"/>
  <xsl:param name="aRequestName"/>
  <xsl:param name="aDate"/>

  <!-- Define a variable for AAA repository XML file -->
  <xsl:variable name="vAuthorizationFile" select="document('local:///AAA-repository.xml')"/>

  <!-- Debug value of aRequestName parameter -->
  <xsl:call-template name="debug">
   <xsl:with-param name="aLabel">request name</xsl:with-param>
   <xsl:with-param name="aMessage"><xsl:value-of select="$aRequestName"/></xsl:with-param>
  </xsl:call-template>

  <!-- Debug value of user id-->
```

```
  <xsl:call-template name="debug">
   <xsl:with-param name="aLabel">user id</xsl:with-param>
   <xsl:with-param name="aMessage"><xsl:value-of select="$aUserId"/></xsl:with-param>
  </xsl:call-template>

  <xsl:choose>
   <xsl:when test="$vAuthorizationFile != ''">
    <!-- *** Step1 : get the start date from authorize/userid, if it exists ! *** -->
    <xsl:variable name="vStartDate"
select="$vAuthorizationFile/aaa-data/authorize/request[@name=$aRequestName]/user[@id=string($aUs
erId)]/@start-date"/>

    <!-- Debug value of vStartDate-->
    <xsl:call-template name="debug">
     <xsl:with-param name="aLabel">start-date</xsl:with-param>
     <xsl:with-param name="aMessage"><xsl:value-of select="$vStartDate"/></xsl:with-param>
    </xsl:call-template>

    <xsl:choose>
     <!-- *** Step2 : verify that current date is ok (above the service start date) *** -->
     <xsl:when test="number($aDate) &gt; number($vStartDate)">
      <!-- ***
        If everything matches then we want to output <approved/>
        to let the DataPower AAA framework know that authorization succeeded.
       *** -->
      <approved/>
     </xsl:when>
     <!-- Add specific error handling in xsl:otherwise condition -->
     <xsl:otherwise>
      <declined/>
     </xsl:otherwise>
    </xsl:choose>
   </xsl:when>
   <!-- Add specific error handling in xsl:otherwise condition -->
   <xsl:otherwise>
    <declined/>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:template>
</xsl:stylesheet>
```

Note: The value of the extracted resource (local name of the request element) is accessible by the DataPower context variable `var://context/WSM/resource/extracted-resource`.

Figure 6-6 shows the XML content of the output of our successful authorization step.

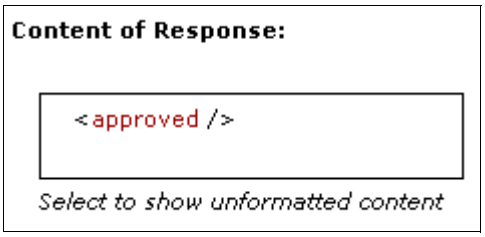


Figure 6-6 Output of authorization step

This template does not manage a specific error handling. We discuss this point in the example in 6.8, “Example 6: On-error handling using custom templates” on page 160.

6.4 Example 2: Dynamic routing based on custom templates

In this example, we show how to implement dynamic routing based on custom templates.

6.4.1 Objectives and presentation

The goal of the second example is to answer the following questions:

- How do we store and get the route information?
- How do we dynamically route an incoming request?
- What is the necessary DataPower device configuration to perform dynamic routing?

The following exclusive XSL stylesheets are used in this example:

- `itso.dynamic-routing-xml.xsl` is for dynamic routing based on an XML file.
- `itso.dynamic-routing-db2.xsl` is for dynamic routing using DB2®.

A first option, using `itso.dynamic-routing-xml.xsl`, is to define the route in an XML file, routes.xml, which contains the target destination linked to the local name of the request. Example 6-14 shows the contents of the routes.xml file.

Example 6-14 routes.xml

```
<?xml version="1.0" encoding="utf-8"?>
<routes>
<route request-name="getPrime"
service-url="http://itsolab1:9080/PrimesWebService/services/Primes"/>
<route request-name="getOdd"
service-url="http://itsolab1:9080/PrimesWebService/services/Odd"/>
<route request-name="getEven"
service-url="http://itsolab1:9080/PrimesWebService/services/Even"/>
</routes>
```

A second option, which uses `itso.dynamic-routing-db2.xsl`, is to use a database to link the local name of a request with the target destination of the requested service. We then create a DB2 database, called SAMPLE, in which we add a table ROUTE that contains two columns: REQUEST_NAME and SERVICE_URL.

Table 6-2 shows an entry in the ROUTE table that associates a request name with a target service URL.

Table 6-2 ROUTE table

REQUEST_NAME	SERVICE_URL
getPrime	`http://itsolab1:9080/PrimesWebService/services/Primes`

These custom templates and routes.xml file are loaded in the local directory of the DataPower device.

6.4.2 DataPower configuration

Figure 6-7 shows the request processing rule, on which we add dynamic routing.

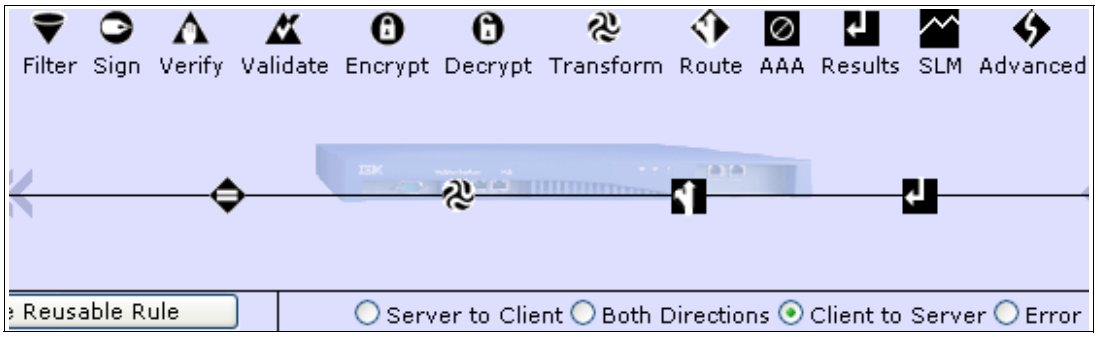


Figure 6-7 Request rule with dynamic routing

The transformation action that we add on the request processing rule can implement either of the following different XSL stylesheets:

- `itso.dynamic-routing-xml.xsl`
- `itso.dynamic-routing-db2.xsl`

It is possible to use the `itso.dynamic-routing-xml.xsl` stylesheet to retrieve the target destination from an XML file. Alternatively, we can use the `itso.dynamic-routing-db2.xsl` stylesheet to recover the destination from a DB2 database. We might switch from one to another custom template without any other modification on the request rule.

The route action uses a custom global variable that determines the target destination. This variable is set in the transformation action by using either `itso.dynamic-routing-xml.xsl` or `itso.dynamic-routing-db2.xsl`. Its name is `var://context/txn-info/dynamic-route`.

General configuration

To perform dynamic routing, on the Web Service Proxy configuration page (Figure 6-8), for Type, select the **dynamic-backend** option.

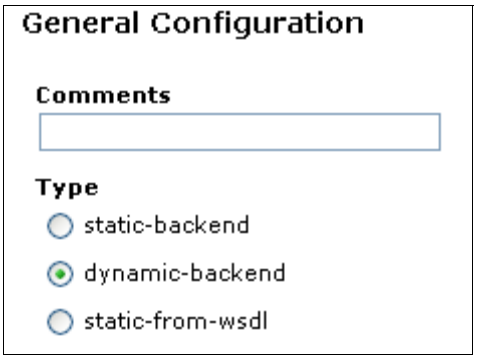


Figure 6-8 general configuration for dynamic routing

SQL Data Source object

For the purpose of the custom template implementing a DB2 connection, we must define a DataPower SQL Data Source object. The name of this object is ITSODS. To access the SQL Data Source objects, select **NETWORK** → **SQL Data Source**.

Figure 6-9 shows the details of the ITSODS object, by using DB2 and the database SAMPLE, located on the server `itsolab1` (a host alias for the actual IP address).

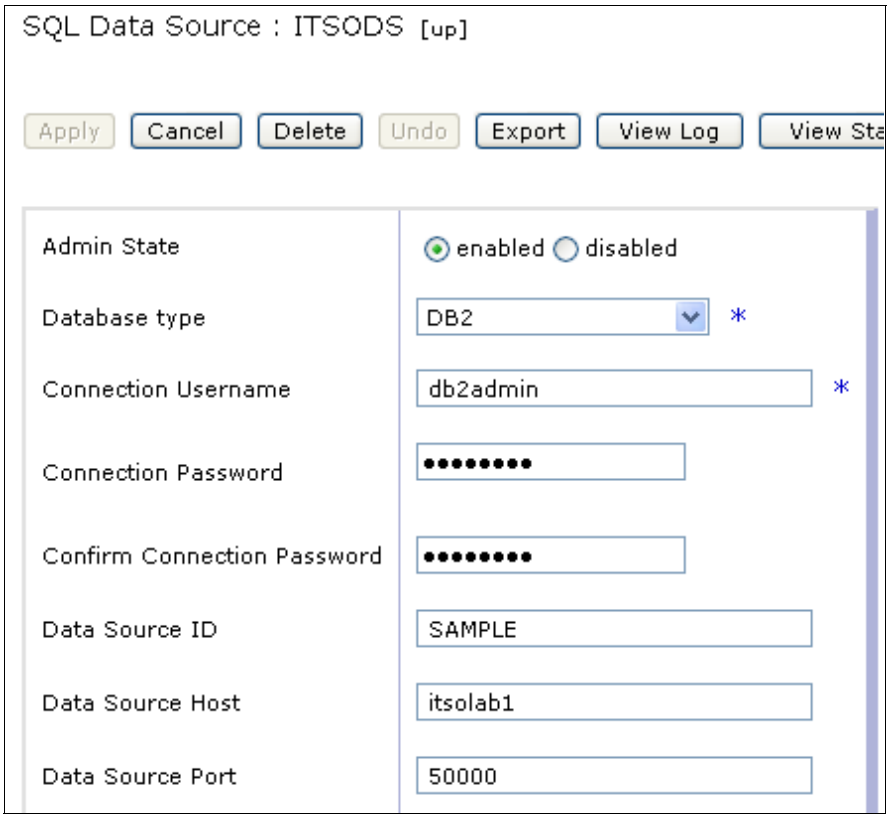


Figure 6-9 SQL Data Source object: ITSODS

SQL Data Source objects contain all the required information that is needed to access a remote database (DB2, Oracle®, Sybase, or Microsoft® SQL Server®).

6.4.3 Incoming SOAP message

Example 6-15 shows that the incoming request is a SOAP message.

Example 6-15 getPrime_EX2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:q0="http://com.itso"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Header/>
<soapenv:Body>
<q0:getPrime>
<numDigits>24</numDigits>
</q0:getPrime>
</soapenv:Body>
</soapenv:Envelope>
```

6.4.4 XSL stylesheet details

In this section, we use the following XSL stylesheets to complete the recovery of the target destination and to set the multi-step variable `var://context/txn-info/dynamic-route`, which is used by the route action to perform dynamic routing:

- `itso.dynamic-routing-xml.xsl` (if the target destination is recovered from the routes.xml file)
- `itso.dynamic-routing-db2.xsl` (if the target destination is recovered from the DB2 database)

Dynamic routing based on an XML file

Example 6-16 shows the custom XSL stylesheet that we used to dynamically get and set the target destination of a service.

Example 6-16 itso.dynamic-routing-xml.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<!--
+
|*************************************************************************
|*** Author: ITSO
|*** file: itso.dynamic-routing-xml.xsl
|*** Description: This XSL Stylesheet performs dynamic routing. URL is recovered
|*** from an XML file and stored in a variable that is used by a DataPower route action
|*** Revision : 1.0 : initial version
|*************************************************************************
+ -->
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:dp="http://www.datapower.com/extensions"
 extension-element-prefixes="dp"
 exclude-result-prefixes="dp">
```

```
<!-- +
  |************************************
  |*** Matching Template
  |*** Element: First Element of the Body
  |************************************
  + -->
 <xsl:template match="/*[local-name()='Envelope']/*[local-name()='Body']/*[1]">

  <!-- We get the service Url linked to our request name -->
  <xsl:variable name="vServiceUrl">
   <xsl:call-template name="getUrl">
    <xsl:with-param name="aRequestName" select="local-name()"/>
   </xsl:call-template>
  </xsl:variable>

  <!-- Set variable 'var:///context/txn-info/dynamic-route' that is used for dynamic routing -->
  <dp:set-variable name="'var://context/txn-info/dynamic-route'" value="string($vServiceUrl)"/>

 </xsl:template>

<!-- +
  |******************
  |*** Named Template
  |*** Name: getUrl
  |******************
  + -->
 <xsl:template name="getUrl">
  <xsl:param name="aRequestName"/>

  <!-- We create a variable using the xml file that contains routing info -->
  <xsl:variable name="vFile" select="document('local:///routes.xml')"/>

  <!-- We get the URL associated to the local name of the incoming request -->
  <xsl:variable name="vServiceUrl"
select="$vFile/routes/route[@request-name=$aRequestName]/@service-url"/>

  <!-- Return the URL value as a string -->
  <xsl:value-of select="$vServiceUrl"/>

 </xsl:template>

</xsl:stylesheet>
```

Extension element: We use the extension element `<dp:set-variable/>` to set the global variable `var://context/txn-info/dynamic-route` that is used by the route action for dynamic routing.

Dynamic routing based on a database

Example 6-17 shows the custom XSL stylesheet that we used to dynamically get and set the target destination of a service. The service URL is recovered from a DB2 database.

Example 6-17 itso.dynamic-routing-db2.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<!--
+
|*************************************************************************
|*** Author: ITSO
|*** file: itso.dynamic-routing-db2.xsl
|*** Description: This XSL Stylesheet performs dynamic routing. URL is recovered
|*** from DB2 and stored in a variable that is used by a DataPower route action
|*** Revision : 1.0 : initial version
|*************************************************************************
+
-->
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:dp="http://www.datapower.com/extensions"
 extension-element-prefixes="dp      "
 exclude-result-prefixes="dp">

<!-- +
  |************************************
  |*** Matching Template
  |*** Element: First Element of the Body
  |************************************
  + -->
 <xsl:template match="/*[local-name()='Envelope']/*[local-name()='Body']/*[1]">

  <!-- We get the service Url linked to our request name -->
  <xsl:variable name="vServiceUrl">
   <xsl:call-template name="getUrl">
    <xsl:with-param name="aRequestName" select="local-name()"/>
   </xsl:call-template>
  </xsl:variable>

  <!-- Set variable 'var:///context/txn-info/dynamic-route' that is used for dynamic routing -->
  <dp:set-variable name="'var://context/txn-info/dynamic-route'" value="string($vServiceUrl)"/>

 </xsl:template>

<!-- +
  |******************
  |*** Named Template
  |*** Name: getUrl
  |******************
  + -->
 <xsl:template name="getUrl">
  <xsl:param name="aRequestName"/>

  <xsl:variable name="vSqlQuery">
   SELECT SERVICE_URL FROM DPADMIN.ROUTE WHERE REQUEST_NAME='<xsl:value-of
select="$aRequestName"/>'
```

```
  </xsl:variable>

  <!-- Perform the SQL query to recover the service URL -->
  <xsl:variable name="vDB2Return" select="dp:sql-execute('ITSODS',$vSqlQuery)"/>

  <!-- Return the URL value as a string -->
  <xsl:value-of select="$vDB2Return/sql/row/column/value"/>

 </xsl:template>

</xsl:stylesheet>
```

First, we recover the local name of the first body's element of `getPrime`. Then we execute the following SQL statement:

```
SELECT SERVICE_URL FROM DPADMIN.ROUTE WHERE REQUEST_NAME='getPrime'
```

Extension function: An SQL statement is executed by using the extension function `dp:sql-execute(SQLDataSource, SQLStatement)`. The first parameter identifies the SQL Data Source object, which must be created on the DataPower device. The second parameter is the SQL statement itself.

Figure 6-10 shows the results of the SQL statement that are included in the XML content.

Figure 6-10 Results of the dp:sql-execute function

This content must be parsed to obtain the value of the target destination. After the value is retrieved, you can set the multistep variable `var://context/txn-info/dynamic-route`.

6.5 Example 3: GET request transformed into a SOAP message

In this example, we explain how to transform an incoming HTTP GET request into an HTTP SOAP posted message. The example is based on the loopback proxy XML firewall `ITSO_FWL_XSL`.

6.5.1 Objectives and presentation

The goal of this third example is to answer the following questions:

- How do we recover request parameters from a URL?
- How do we transform these parameters into a SOAP message?
- How do we transform an incoming HTTP GET into an HTTP POST that is required to send a SOAP message to a Web service?

The XSL stylesheet `itso.get2soap-dynamic_routing.xsl` is used to perform this action. This custom template is loaded in the local directory of the DataPower device. The `itso.get2soap-dynamic_routing.xsl` XSL stylesheet is in charge of the following actions:

- Recovering request parameters from the incoming HTTP GET request
- Creating a SOAP message, based on the previous parameter values, which is used to complete the POST request
- Retrieving the URL of the target destination of the Web service
- Posting the SOAP request to the appropriate Web service and providing the result

The destination of the Web service is dynamically recovered by using a DB2 database, as demonstrated in 6.4, “Example 2: Dynamic routing based on custom templates” on page 141.

6.5.2 DataPower configuration

In this example, we use an XML firewall (ITSO_FWL_XSL) as a loopback proxy. On this type of server, a response rule is useless, because request and response processing are managed in the same request processing rule. Figure 6-11 shows the processing rule of the loopback proxy that requires a single transformation.

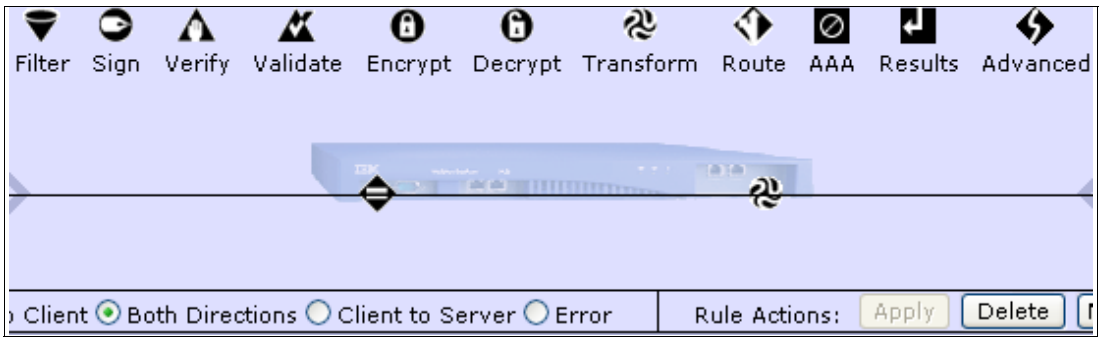


Figure 6-11 Request rule for the HTTP GET to HTTP SOAP(POST) transformation

You can find more configuration details about this XML firewall in Appendix A, “XSL programming issues” on page 179.

Both directions rule: You can use a “both directions” type rule, because requests and responses are treated in the same rule, while using an XML firewall as a loopback proxy.

6.5.3 Incoming HTTP GET request

The incoming request is an HTTP GET, whose URL contains two required parameters:

- The *op* parameter contains the name of the operation. Only getPrime is allowed.
- The *value* parameter contains the number of digits for the requested prime number.

Figure 6-12 shows an example of a GET request, by using the **curl** command.

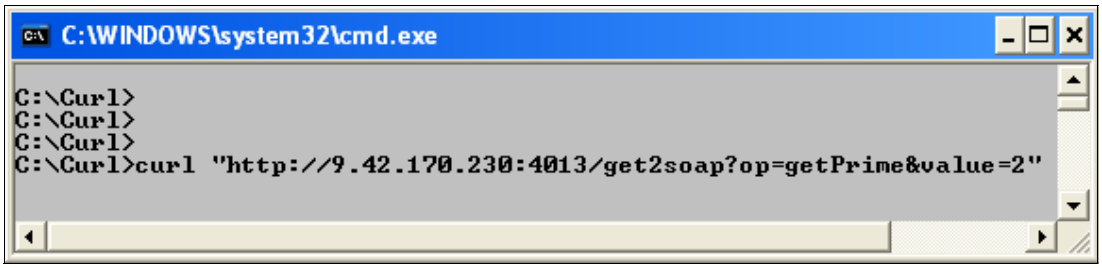


Figure 6-12 The curl command for the GET request

/get2soap: Use the pattern `/get2soap` in the URL in order to use the appropriate rule of the XML firewall.

6.5.4 XSL stylesheet details

In this section, we introduce the custom template `itso.get2soap-dynamic_routing.xsl` that we use to complete the HTTP GET to HTTP SOAP transformation and to perform the call to the appropriate Web service. With this XSL stylesheet, we can perform the following actions:

- Transcode incoming parameters into a SOAP message
- Recover the destination of the Web service via a DB2 database
- Call a Web service by using the extension element `<dp:url-open>`, as shown in Example 6-18

Example 6-18 itso.get2soap-dynamic_routing.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<!--
+
|*****************************************************************************
|*** Author: ITSO
|*** file: itso.get2soap-dynamic_routing.xsl
|*** Description: This XSL performs GET to SOAP (POST) transformation using dp:url-open element
|*** Web-Service url is dynamically recovered from DB2 database
|*** Revision : 1.0 : initial version
|*****************************************************************************
+
-->
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:dp="http://www.datapower.com/extensions"
   xmlns:dpquery="http://www.datapower.com/param/query"
   xmlns:itso="http://com.itso"
   extension-element-prefixes="dp dpquery"
   exclude-result-prefixes="dp dpquery">
```

```
 <!-- Import XSL stylesheet which provides debugging enhancements-->
 <xsl:import href="local:///itso-common-debug.xsl"/>

 <!-- Output definition -->
 <xsl:output method="xml" indent="yes" encoding="utf-8" />

 <!-- Get both 'op' and 'value' parameters value -->
    <xsl:param name="dpquery:op" />
    <xsl:param name="dpquery:value" />

<!--+
  |*********************
  |*** Matching Template
  |*** Element: ROOT
  |*********************
  + -->
<xsl:template match="/">
 <!-- Create the output SOAP message based on parameter values -->
      <xsl:variable name="vSoapMessage">
         <xsl:call-template name="createSoapMessage">
            <xsl:with-param name="aOperation" select="$dpquery:op" />
            <xsl:with-param name="aValue" select="$dpquery:value" />
         </xsl:call-template>
      </xsl:variable>

 <!-- Debug value of vStartDate-->
 <xsl:call-template name="debug">
  <xsl:with-param name="aLabel">soap message</xsl:with-param>
  <xsl:with-param name="aMessage"><xsl:value-of select="$vSoapMessage"/></xsl:with-param>
 </xsl:call-template>

 <!-- We get the service Url linked to our request name -->
 <xsl:variable name="vServiceUrl">
  <xsl:call-template name="getUrl">
   <xsl:with-param name="aRequestName" select="$dpquery:op"/>
  </xsl:call-template>
 </xsl:variable>

<!-- create HTTP header to be used for the POST request-->
 <xsl:variable name="vHeaders">
  <header name="SOAPAction">
   <xsl:value-of select="''"/>
  </header>
 </xsl:variable>

 <dp:url-open http-headers="$vHeaders"
target="http://itsolab1:9080/PrimesWebService/services/Primes" response="xml">
       <xsl:copy-of select="$vSoapMessage" />
 </dp:url-open>

   </xsl:template>

 <!--+
        |*************************
```

```
        |*** Named Template
        |*** Name: createSoapMessage
        |**************************
        +
    -->
   <xsl:template name="createSoapMessage">
      <xsl:param name="aOperation" />
      <xsl:param name="aValue" />

 <SOAP-ENV:Envelope>
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
   <xsl:element name="{concat('itso:',$aOperation)}">
    <numDigits><xsl:value-of select="$aValue" /></numDigits>
   </xsl:element>
  </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>

   </xsl:template>

<!-- +
 |******************
 |*** Named Template
 |*** Name: getUrl
 |******************
 + -->
 <xsl:template name="getUrl">
  <xsl:param name="aRequestName"/>

  <xsl:variable name="vSqlQuery">
   SELECT SERVICE_URL FROM DPADMIN.ROUTE WHERE REQUEST_NAME='<xsl:value-of
select="$aRequestName"/>'
  </xsl:variable>

  <!-- Perform the SQL query to recover the service URL -->
  <xsl:variable name="vDB2Return" select="dp:sql-execute('ITSODS',$vSqlQuery)"/>

  <!-- Return the URL value as a string -->
  <xsl:value-of select="$vDB2Return/sql/row/column/value"/>

 </xsl:template>

</xsl:stylesheet>
```

dpquery: We use the namespace *dpquery*, so that we can directly obtain the value of the parameters `op` and `value` by using `<xsl:param name="dpquery:op"/>` and `<xsl:param name="dpquery:value"/>`.

The SOAP message is created by using values of both `op` and `value` parameters. The destination URL of the Web service is recovered by a DB2 database query, using the `ITSODS` SQL Data Source object.

A POST request is completed by using the extension element `<dp:url-open>`. If content is defined in `<dp:url-open>`, then an HTTP POST is performed. Otherwise it is a GET.

The SOAPAction HTTP header is added by using the http-headers attribute of the <dp:url-open> element. As we work with a loopback proxy, the result is directly sent to the client. Figure 6-13 shows an example of this result.

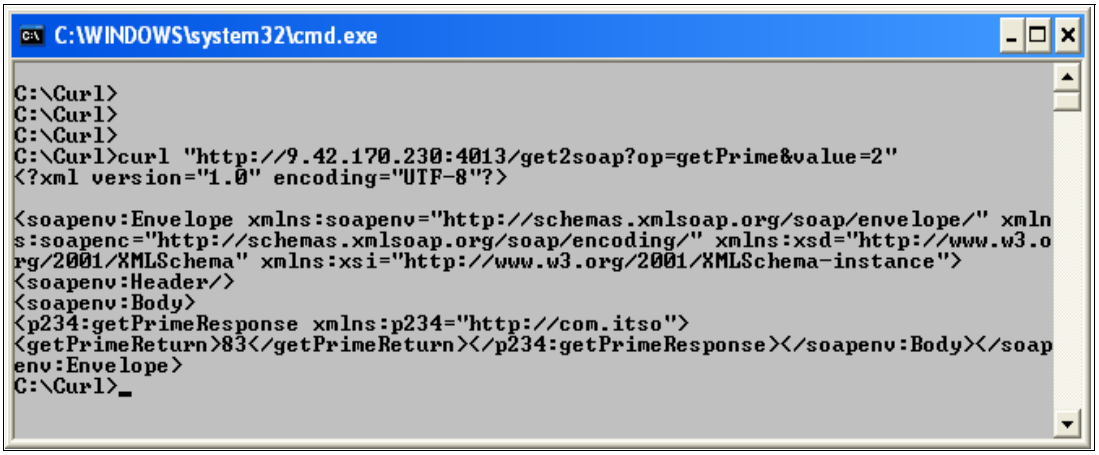


Figure 6-13 Resulting SOAP message of a GET request

More information: You can find more details about the <dp:url-open> extension element in the latest edition of the *DataPower Extension Functions Catalog* on the Web at the following address:

https://www14.software.ibm.com/webapp/iwm/web/reg/download.do?source=swg-datapower&S_PKG=xi50_9003_s2&dlmethod=http

6.6 Example 4: Debugging into the DataPower XSL stylesheet

In this example, we demonstrate how to debug into the DataPower custom templates. We use the first example to present this enhancement.

6.6.1 Objectives and presentation

The goal of this fourth example is to answer the question: How do we easily debug into DataPower custom templates?

We use the XSL stylesheet itso-common-debug.xsl in this example. This XSL stylesheet provides a named template, called debug, which creates a DataPower debug message with an error level that is visible in probes. For more details about the DataPower error level, see Chapter 5, “Logging capabilities the in DataPower appliance” on page 113.

Custom template: This custom template is loaded in the local directory of the DataPower device.

6.6.2 DataPower configuration

The default log level of the system must not be set higher than the error level, because our named template uses the error priority.

To see the debug messages, we must enable probes for the appropriate DataPower service. For example, we enable probes on the Web service proxy that we used in 6.3, “Example 1: AAA policy based on custom templates” on page 129. In addition, debug messages are also displayed on DataPower system logs.

6.6.3 Incoming SOAP message

The incoming SOAP message and request are the same as those used in 6.3.3, “Incoming SOAP message” on page 131.

6.6.4 XSL stylesheet details

In this section, we introduce the `itso-common-debug.xsl` template that we use to display debug messages into probes or system logs. This XSL stylesheet provides a `debug` named template (Example 6-19) that may be used in other XSL stylesheets.

Example 6-19 itso-common-debug.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<!--
+
|*****************************************************************************
|*** Author: ITSO
|*** file: itso-common-debug.xsl
|*** Description: This XSL creates a DataPower debug message with error level, visible in probes
or system logs
|*** Import this stylesheet to use the 'debug' named template in your custom template
|*** Revision : 1.0 : initial version
|*****************************************************************************
+ -->
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
    extension-element-prefixes="dp       "
    exclude-result-prefixes="dp">

 <!-- +
   |******************
   |*** Named Template
   |*** Name: debug
   |******************
   + -->
 <xsl:template name="debug">
  <xsl:param name="aLabel"/>
  <xsl:param name="aMessage"/>
  <xsl:message dp:priority="error">
   <xsl:if test="string($aLabel)!=''">
<xsl:value-of select="concat('*** ITSO-debug | ',$aLabel,':',$aMessage,'***')"/>
   </xsl:if>
  </xsl:message>
 </xsl:template>
</xsl:stylesheet>
```

DataPower logging capabilities: We use the DataPower logging capabilities to debug XSL stylesheets. We use the <xsl:message> element with the attribute dp:priority="error" to set the priority level to error.

The debug named template requires the following two parameters:

- aLabel, which is the label of the message
- aMessage, which is the debugging message itself

To use this named template, we must import the itso-common-debug.xsl template from the XSL stylesheet that requires debug information. The import is performed by using the <xsl:import> element as shown in Example 6-20.

Example 6-20 itso.aaa-authorize.xsl

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
    xmlns:itso="http://com.itso"
    extension-element-prefixes="dp"
    exclude-result-prefixes="dp itso">

 <!-- Import XSL stylesheet which provides debugging enhancements-->
 <xsl:import href="local:///itso-common-debug.xsl"/>

<!-- *** HERE IS THE CODE OF THE XSL STYLESHEET *** -->

</xsl:stylesheet>
```

<xsl:import>: If used, the <xsl:import> element must be the first child node of <xsl:stylesheet>.

After <xsl:import href="local:///itso-common-debug.xsl"/> is added to our stylesheet, we use the debug template as shown in Example 6-21, which is an extract of the itso.aaa-authorize.xsl stylesheet.

Example 6-21 Using the debug named template, itso.aaa-authorize.xsl

```
...
<!--+
  |*********************
  |*** Named Template
  |*** name: authorize
  |*********************
  + -->
 <xsl:template name="authorize">
  <xsl:param name="aUserId"/>
  <xsl:param name="aRequestName"/>
  <xsl:param name="aDate"/>

  <!-- Define a variable for AAA repository XML file -->
  <xsl:variable name="vAuthorizationFile"
select="document('local:///AAA-repository.xml')"/>

  <!-- Debug value of aRequest parameter -->
```

```
<xsl:call-template name="debug">
<xsl:with-param name="aLabel">request name</xsl:with-param>
   <xsl:with-param name="aMessage"><xsl:value-of
select="$aRequestName"/></xsl:with-param>
  </xsl:call-template>
...
```

In the named template `authorize`, we use the `debug` named template to debug the value of the `aRequestName` parameter. Figure 6-14 shows the contents of the Web service proxy probes.

Figure 6-14 Debug message in the Web service proxy probes

Debug message: The debug message is displayed with the category `xsltmsg` and level `error` as defined in the `debug` named template.

6.7 Example 5: Logging from custom templates

In this example, we explain how to send a log message from a DataPower XSL stylesheet. This method requires a specific DataPower log configuration. For more details about DataPower logging capabilities, see Chapter 5, “Logging capabilities the in DataPower appliance” on page 113. We use the template shown in Example 6-5 on page 127.

6.7.1 Objectives and presentation

The goal of this fifth example is to answer the following questions:

- How do we send a log message from DataPower custom templates?
- What is the required DataPower log configuration?

In this example, we use the XSL stylesheet `itso.get2soap-dynamic_routing_log.xsl`. This custom template is based on `itso.get2soap-dynamic_routing.xsl` that we explained in 6.5, “Example 3: GET request transformed into a SOAP message” on page 147, on which we implement a modification to send a specific log message.

The XSL stylesheet `itso.get2soap-dynamic_routing_log.xsl` sends a log message in the `logtemp:///logFile.log` filestore inside the DataPower device. As discussed in Chapter 5, “Logging capabilities the in DataPower appliance” on page 113, a better practice is to send logs outside of the DataPower device.

The log message that is sent contains a value of the `op` request parameter. The custom template `itso.get2soap-dynamic_routing_log.xsl` is loaded in the local directory of the DataPower device.

6.7.2 DataPower configuration

In this section, we discuss the log configuration that is required to send a log message from the custom template `itso.get2soap-dynamic_routing_log.xs` into the `logtemp:///logFile.log` filestore.

To complete this task, we must create the following items in the order shown:

1. A log category that is used in our custom template to activate our logging process.
2. A log target by using our category as a subscribed event. The target is the `logtemp:///logFile.log` filestore.

We must also define the priority level of the log message. In this example, the error level is used.

Log category

The name of our log category is `itso_log`, which we define with the Log Category parameters (Figure 6-15). To define the log category parameters, we first click **ADMINISTRATION** → **Configure Log Categories**.

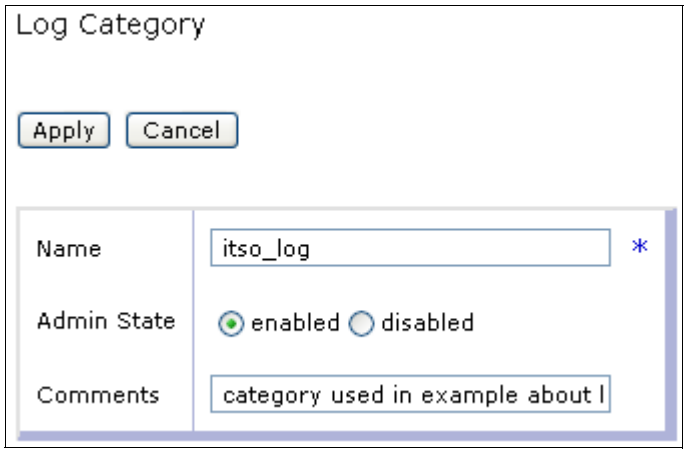


Figure 6-15 itso_log category parameters

Figure 6-16 shows the log category that we created.

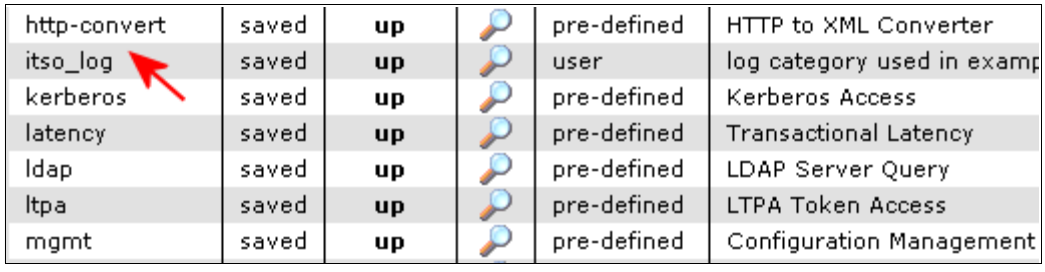


Figure 6-16 itso_log category

Log target

The name of our log target is `itso_log_target`, which we define in the Log target parameters (Figure 6-17). We create this log target by first selecting **ADMINISTRATION** → **Manage Log Targets**.

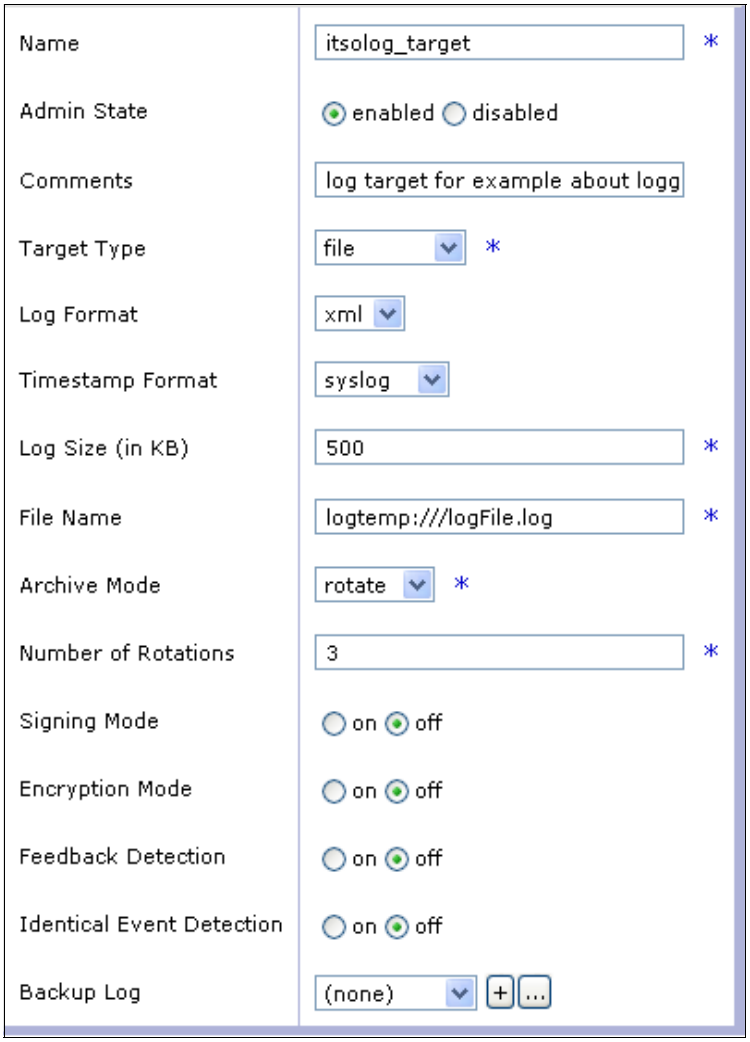


Figure 6-17 Log target parameters

After we create the log target, we click the **Event Subscriptions** tab (Figure 6-18), and link it with the `itso_log` category and the error priority level.

Figure 6-18 Event subscription

6.7.3 Incoming request

The incoming request is an HTTP GET, whose URL contains two required parameters:

- The *op* parameter contains the name of the operation. Only getPrime is allowed. The value of the op parameter is logged in the logtemp:///logFile.log filestore, inside the DataPower appliance.
- The *value* parameter contains the number of digits of the requested prime number.

Figure 6-19 shows an example of a GET request by using the **`curl`** command.

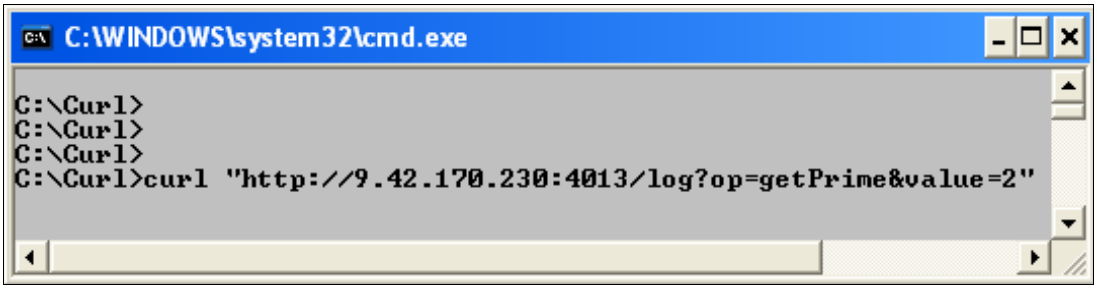


Figure 6-19 The curl command for logging

/log: Use the pattern `/log` in the URL to use the appropriate rule of the XML firewall.

6.7.4 XSL stylesheet details

In this section, we introduce the custom template `itso.get2soap-dynamic_routing_log.xsl` to send a log message into the `logtemp:///logFile.log` filestore. To send a log from a custom template, we use the `<xsl:message>` element with some specific attributes, as shown in Example 6-22.

Example 6-22 itso.get2soap-dynamic_routing_log.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<!--
+ |*****************************************************************************
|*** Author: ITSO
|*** file: itso.get2soap-dynamic_routing_log.xsl
|*** Description: This XSL performs GET to SOAP (POST) and log the name of the 'op' parameter
value
|*** Revision : 1.0 : initial version
|*****************************************************************************
+ -->

<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:dp="http://www.datapower.com/extensions"
   xmlns:dpquery="http://www.datapower.com/param/query"
   xmlns:itso="http://com.itso"
   extension-element-prefixes="dp dpquery"
   exclude-result-prefixes="dp dpquery">

 <!-- Import XSL stylesheet which provides debugging enhancements-->
 <xsl:import href="local:///itso-common-debug.xsl"/>

 <!-- Output definition -->
 <xsl:output method="xml" indent="yes" encoding="utf-8" />

 <!-- Get both 'op' and 'value' parameters value -->
    <xsl:param name="dpquery:op" />
    <xsl:param name="dpquery:value" />

 <!--+
        |*********************
        |*** Matching Template
        |*** Element: ROOT
        |*********************
        +
    -->
   <xsl:template match="/">

 <!-- Create the output SOAP message based on parameter values -->
      <xsl:variable name="vSoapMessage">
         <xsl:call-template name="createSoapMessage">
            <xsl:with-param name="aOperation" select="$dpquery:op" />
            <xsl:with-param name="aValue" select="$dpquery:value" />
         </xsl:call-template>
      </xsl:variable>
```

```
 <!-- Send a log message containing the name of the 'op' parameter value-->
 <!-- *** use dp:type to define the 'itso_log' category *** -->
 <!-- *** use dp:priority to define the 'error' log level *** -->
 <xsl:message dp:type="itso_log" dp:priority="error">
  <xsl:value-of select="concat('operation : ',$dpquery:op)"/>
 </xsl:message>
...
   </xsl:template>
...
</xsl:stylesheet>
```

Here we only mention how to send a log message, because the other part of this XSL stylesheet is exactly the same as `itso.get2soap-dynamic_routing.xsl`. As described, sending a log message is completed by using the `<xsl:message>` element with following attributes:

- `dp:type`, which defines the category that we want to use
- `dp:priority`, which defines the priority level

It is possible to see the logged information in the `logtemp:///logFile.log` file. Example 6-23 shows an extract of this file.

Example 6-23 Extract of the logtemp:///Logfile.log file

```
<log-entry serial='1' domain='XSLTProgramming'>
<date>Wed Jun 13 2007</date>
<time utc='1181747208714'>11:06:48</time>
<date-time>2007-06-13T11:06:48</date-time>
<type>itso_log</type>
<class>xmlfirewall</class>
<object>ITSO_FWL_XSL</object>
<level num='3'>error</level>
<transaction-type>request</transaction-type>
<transaction>48895</transaction>
<client>9.42.171.105</client>
<code>0x80000001</code>
<file>local:///itso.get2soap-dynamic_routing_log.xsl</file>
<message>operation : getPrime</message>
</log-entry>
```

We can see log details in the `<message>` element. The following information is also displayed:

- The current date and time
- The category and priority level
- The transaction identifier
- The custom template which initiates the log

6.8 Example 6: On-error handling using custom templates

In this example, we explain how to handle errors in the DataPower custom templates. For more details about DataPower on-error handling, see Chapter 5, “Logging capabilities the in DataPower appliance” on page 113. We use the first example scenario described in 6.3, “Example 1: AAA policy based on custom templates” on page 129, based on a Web service proxy, to the present purpose for on-error handling.

6.8.1 Objectives and presentation

The goal of this sixth example is to answer the following questions:

- How do we configure an on-error processing rule on a processing policy?
- How do we use an on-error processing rule from a custom template, when an error occurs?

We use the following XSL stylesheets in this example:

- `itso.on-error_handling.xsl`

  This custom template creates a SOAP fault message by using the following dynamic information:

  - A fault code
  - A fault string
  - An error code

- `itso.aaa-authenticate_error-handling.xsl`

  Because our example is based on 6.3, “Example 1: AAA policy based on custom templates” on page 129, we add specific error handling on authentication step of the `ITSO_XSL_PROG` AAA policy. Therefore, in this example, we use `itso.aaa-authenticate_error-handling.xsl` instead of `itso.aaa-authenticate.xsl` in the AAA policy.

The XSL stylesheet `itso.aaa-authenticate_error-handling.xsl` handles authentication errors. If a user is not authenticated, we generate a SOAP fault message indicating the authentication failure.

Both `itso.on-error_handling.xsl` and `itso.aaa-authenticate_error-handling.xsl` are loaded in the local directory of the DataPower device.

6.8.2 DataPower configuration

In this section, we describe on-error processing rule creation, on the Web service proxy `HelloWSProxy`.

On-error processing rule

We add a processing rule at the Web Services Description Language (WSDL) level of our Web service proxy. The type of this rule is `Error`, indicating that it is an on-error processing rule. We then add a transformation action, which uses the `itso.on-error_handling.xsl` custom template, as shown in Figure 6-20.

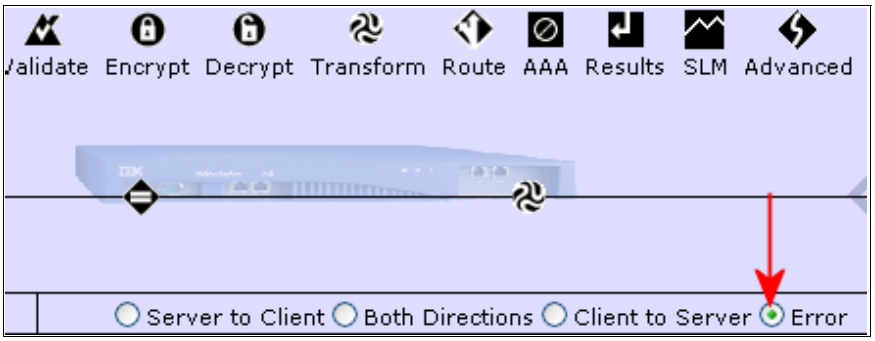


Figure 6-20 On-error processing rule

Request processing rule

The only change we make on the request processing rule concerns the AAA action. Indeed, the AAA policy must use the custom template `itso.aaa-authenticate_error-handling.xsl`, which handles authentication server errors.

6.8.3 Incoming request

The incoming request is the same as the one that we used in 6.3.3, “Incoming SOAP message” on page 131. If we want a SOAP fault message that indicates an authentication failure, we provide a wrong user name or password in the WS-Security header tokens that define the current user.

6.8.4 XSL stylesheet details

In this section, we describe the XSL stylesheets that we used to demonstrate on-error handling.

Error handling in authentication custom template

The authentication custom template works with identity extracted information to complete authentication. If the current user is not authenticated, we set a global variable of `var://context/itso/error-set`, which contains error details regarding the fault code, fault string, and error code.

Multi-step global variables: Minimize the creation of multi-step global variables. In our case, we create a single global variable that contains all error details as a node set.

In order to execute the on-error rule, we use the `<dp:reject>` extension element, as shown in Example 6-24.

Example 6-24 itso.aaa-authenticate_error-handling.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<!--
+ |****************************************************************************
|*** Author: ITSO
|*** file: itso.aaa-authenticate_error-handling.xsl
|*** Description: This XSL is responsible for authentication
|*** In case of authentication failure, an error is returned, using on-error handling
|*** Revision : 1.0 : initial version
|****************************************************************************
+ -->

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
    xmlns:itso="http://com.itso"
    extension-element-prefixes="dp"
    exclude-result-prefixes="dp itso">

...

 <!--+
  |************************
```

```
  |*** Named Template
  |*** Name: authenticate
  |************************
  + -->
 <!-- Verify authentication for a specific {user,password,clientIP} triplet -->
 <xsl:template name="authenticate">
  <xsl:param name="aUser"/>
  <xsl:param name="aPassword"/>
  <xsl:param name="aClientIP"/>

  <!-- define a variable for AAA repository XML file -->
  <xsl:variable name="vAuthenticationFile" select="document('local:///AAA-repository.xml')"/>

  <!-- Here we perform authentication -->
  <xsl:choose>
   <xsl:when test="$vAuthenticationFile != ''">
    <xsl:variable name="vUserID"
select="$vAuthenticationFile/aaa-data/authenticate[string(user)=$aUser][string(password)=$aPassw
ord][contains(string(client-ip),$aClientIP)]/user/@id"/>
    <!-- ***
      If credentials match then we want to output something to let the AAA framework know that
authentication succeeded.
      If credentials do not match, we output nothing and that means 'authentication failure' to
the AAA framework.
      *** -->


    <!-- Debug value of vUserID-->
    <xsl:call-template name="debug">
     <xsl:with-param name="aLabel">userid</xsl:with-param>
     <xsl:with-param name="aMessage"><xsl:value-of select="$vUserID"/></xsl:with-param>
    </xsl:call-template>

    <xsl:choose>
     <xsl:when test="string($vUserID) != ''">
      <userid>
       <xsl:value-of select="$vUserID"/>
      </userid>
     </xsl:when>
     <!-- on-error handling -->
     <xsl:otherwise>
      <xsl:variable name="errorSet">
       <error>
        <faultcode>Client error</faultcode>
        <faultstring>invalid user</faultstring>
        <errorcode>authentication failure</errorcode>
       </error>
      </xsl:variable>
      <dp:set-variable name="'var://context/itso/error-set'" value="$errorSet"/>
      <dp:reject/>
     </xsl:otherwise>
    </xsl:choose>
```

```
   </xsl:when>
      <xsl:otherwise/>
  </xsl:choose>
 </xsl:template>
</xsl:stylesheet>
```

We only mention the new `authenticate` named template, because the other part of this XSL stylesheet is exactly the same as `itso.aaa-authenticate.xsl` that we present in 6.3.4, “XSL stylesheet details” on page 131.

<dp:reject>: Use the `<dp:reject>` extension element to execute the on-error processing rule.

On-error XSL stylesheet

The `itso.on-error_handling.xsl` template is used in the error processing rule. It gets a value of a global variable `var://context/itso/error-set` that is used to set error details. In addition, it creates a SOAP fault message with this recovered value, as shown in Example 6-25.

Example 6-25 itso.on-error_handling.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<!--
+
|*****************************************************************************
|*** Author: ITSO
|*** file: itso.on-error_handling.xsl
|*** Description: This XSL is responsible for creating a SOAP fault message based on value of
global     |*** transaction variables:
|*** - var://context/itso/fault-code : faultcode
|*** - var://context/itso/fault-string : faultstring
|*** - var://context/itso/error-code : errorcode
|*** Created SOAP fault message is the following:
|*** <SOAP-ENV:Envelope>
|***  <SOAP-ENV:Header/>
|***  <SOAP-ENV:Body>
|***   <SOAP-ENV:Fault>
|***    <faultcode>...</faultcode>
|***    <faultstring>...</faultstring>
|***    <detail>
|***     <errorcode>...</errorcode>
|***    </detail>
|***   </SOAP-ENV:Fault>
|***  </SOAP-ENV:Body>
|*** </SOAP-ENV:Envelope>
|*** Revision : 1.0 : initial version
|*****************************************************************************
+ -->

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/       "
    xmlns:dp="http://www.datapower.com/extensions"
    extension-element-prefixes="dp"
    exclude-result-prefixes="dp">
```

```
<xsl:output method="xml" encoding="utf-8" indent="yes"/>

<!--+
 |*********************
 |*** Matching Template
 |*** Element: ROOT
 |*********************
 + -->
<xsl:template match="/">

 <!-- ***
  Get the different data used to create a SOAP fault message.
  The variable is set before calling dp:reject extended element
 *** -->
 <!-- error set contains: fault-code, fault-string and error-code -->
 <xsl:variable name="vErrorSet" select="dp:variable('var://context/itso/error-set')"/>

 <!-- Call template that creates the SOAP fault message -->
 <xsl:call-template name="createSoapFault">
  <xsl:with-param name="aFaultCode" select="$vErrorSet/error/faultcode"/>
  <xsl:with-param name="aFaultString" select="$vErrorSet/error/faultstring"/>
  <xsl:with-param name="aErrorCode" select="$vErrorSet/error/errorcode"/>
 </xsl:call-template>

</xsl:template>

<!--+
 |*************************
 |*** Named Template
 |*** Name: createSoapFault
 |*************************
 + -->
<xsl:template name="createSoapFault">
 <xsl:param name="aFaultCode"/>
 <xsl:param name="aFaultString"/>
 <xsl:param name="aErrorCode"/>

 <SOAP-ENV:Envelope>
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
   <SOAP-ENV:Fault>
    <faultcode><xsl:value-of select="$aFaultCode"/></faultcode>
    <faultstring><xsl:value-of select="$aFaultString"/></faultstring>
   <detail>
    <errorcode><xsl:value-of select="$aErrorCode"/></errorcode>
   </detail>
   </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>

</xsl:template>

</xsl:stylesheet>
```

In case we do not use an authenticated user, authentication failure is handled. Figure 6-21 shows the SOAP fault message.

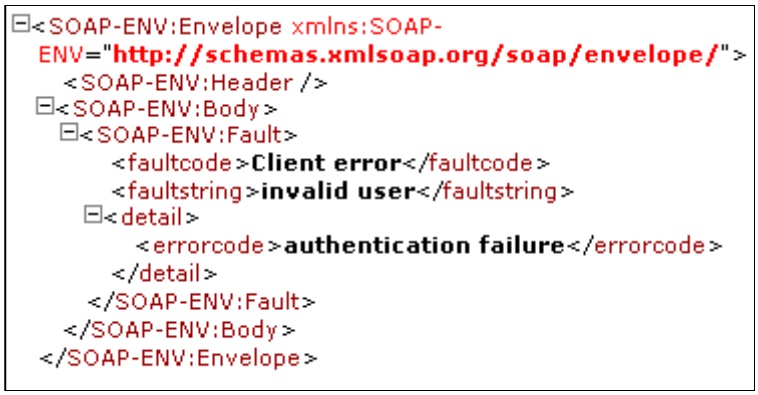


Figure 6-21 SOAP fault message

6.9 Summary

In this chapter, we provided implementation details about custom templates that can be used to perform specific actions. We also presented important extension functions and elements that are provided by the DataPower and EXSLT modules.

Before using a custom template, check whether an existing DataPower object can complete the required tasks.

7

Web 2.0 support

In this chapter, we discuss support for Web 2.0 technologies by using the DataPower appliances. The features that are offered by Web 2.0 applications are becoming more widely adopted by individuals and enterprises alike. In this chapter, we attempt to demonstrate how you can use existing Web services to provide content for Web 2.0 applications. We begin this chapter with an overview of Web 2.0 technology. Then we present an example of Web 2.0 integration and the configuration of the DataPower appliance to support this type of integration.

7.1 Overview of Web 2.0

The phrase Web 2.0 has come to represent any Internet-based application that goes beyond merely serving dynamic content to clients. With Web 2.0 applications or Web sites, end users can upload content to share with other Internet users. This ability sets them apart from other more traditional Web sites that are essentially read-only.

While the underlying technology that supports Web 2.0-enabled sites has not changed, the way in which the Internet is used has. The Internet is no longer a one-way street where content is delivered to the user's browser for consumption. With Web 2.0 technology, users can easily submit their own content for others to see or use a Web-based application as though it were part of their own desktop.

7.1.1 Web 2.0 technologies

Examples of Web 2.0 technologies include social networking, Web logs (or blogs), wikis, and podcasting. In this chapter, we focus on content distribution by using the Atom Syndication Format. Atom is similar to Really Simple Syndication (RSS) and was developed to resolve compatibility issues with the RSS format. Information that is distributed by using the Atom or RSS format is referred to as a *feed*.

Atom defines an XML-based format for providing summarized informational updates for a particular subject. The maintainer of the subject matter's feed can add new information or updates by modifying the Atom file for the feed. End users use feed readers or aggregators to collect and filter the new or updated information. An example use case is a news feed where breaking news headlines can be added to the Atom file by the feed's maintainer. A subscriber to the feed (via their reader) can then view the updated information when they refresh the feed.

7.1.2 Web 2.0 and DataPower appliances

While support for Web 2.0 technology in DataPower appliances is not yet widely acknowledged, the devices can be used to support Web 2.0-based applications. DataPower appliances can be used to mediate Atom feeds because they are XML based.

7.2 Example of Web 2.0 integration

To demonstrate the ability of the DataPower appliance to interoperate with and support Web 2.0-based technologies, we use the DataPower appliance to provide content from a SOAP-based Web service in the Atom Syndication Format.

7.2.1 SOAP Web service

The example SOAP Web service used in this discussion provides a list of customers. Requests to the service define the number of customers that should be returned. Example 7-1 shows a request to the Web service.

Example 7-1 Web service sample request

```
<env:Envelope xmlns:query-string="http://www.datapower.com/param/query"
xmlns:q0="http://sample10.datapower.ibm.com/crafted/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:dp="http://www.datapower.com/schemas/management">
<env:Body>
<q0:CustomerListRequest>
<length>1</length></q0:CustomerListRequest>
</env:Body>
</env:Envelope>
```

The Web service responds with a list of customers, including their names and IDs. Example 7-2 shows a sample response.

Example 7-2 Sample Web service response

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:dp="http://www.datapower.com/schemas/management">
<soapenv:Body>
<CustomerList xmlns="http://sample10.datapower.ibm.com/crafted/">
<Customer xmlns="">
<CustomerID>0</CustomerID>
<CustomerName>customer0</CustomerName>
</Customer>
</Customer>
</CustomerList>
</soapenv:Body>
</soapenv:Envelope>
```

7.2.2 Atom feed

As described in 7.1.1, “Web 2.0 technologies” on page 168, Atom feeds are XML based. Example 7-3 shows a sample Atom document.

Example 7-3 Sample Atom document

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
<id>johndoe@my.test.feed.org</id>
<title>This is an example feed</title>
<link href="http://my.test.feed.org/feed/" rel="self"/>
   <link href="http://my.test.feed.org/"/>
<updated>2007-10-05T18:30:02Z</updated>
<author>
```

```
<name>John Doe</name>
<email>johndoe@my.test.feed.org</email>
</author>
<entry>
<title>DataPower Test Atom Feed</title>
<updated>2007-10-04T18:30:02Z</updated>
<summary>This is a test of the DataPower ATOM feed.</summary>
</entry>
</feed>
```

The feed element describes the title, ID, and update time stamp of the feed, as well as other metadata such as the author's name and relevant Internet links. The entry elements define the actual informational content of the feed. Every feed and entry element must contain a title, ID, and timestamp element.

To represent the Web service as an Atom feed, the list of customers that is returned by the Web service must be transformed into individual Atom entries.

7.2.3 XSL transformations

In the previous section, we briefly described the format of the Atom Syndication Format. To display the SOAP responses from the SampleRead Web service in this format, XSL transformations must be performed by the DataPower appliance:

- Converting the list of customers that is returned by the Web service to the Atom entry elements
- Adding metadata that describes the feed and its entries

These operations are performed by using two XML Stylesheet Language Transformation (XSLT) stylesheets. The first stylesheet generates a SOAP request and sends it to the SampleRead Web service. The response from the Web service is passed to the second stylesheet for processing. This stylesheet extracts the Customer elements of the CustomerList element and converts them to the Atom Syndication Format entry elements. It also inserts the required Atom metadata elements of ID, title, and time stamp for both the entry elements and the root feed element.

7.2.4 DataPower configuration

In this section, we explain the steps to configure the DataPower appliance to enable the SampleRead Web service as an Atom feed. We create two XML firewalls to support the SampleRead Web service as an Atom feed. Figure 7-1 shows the architecture.

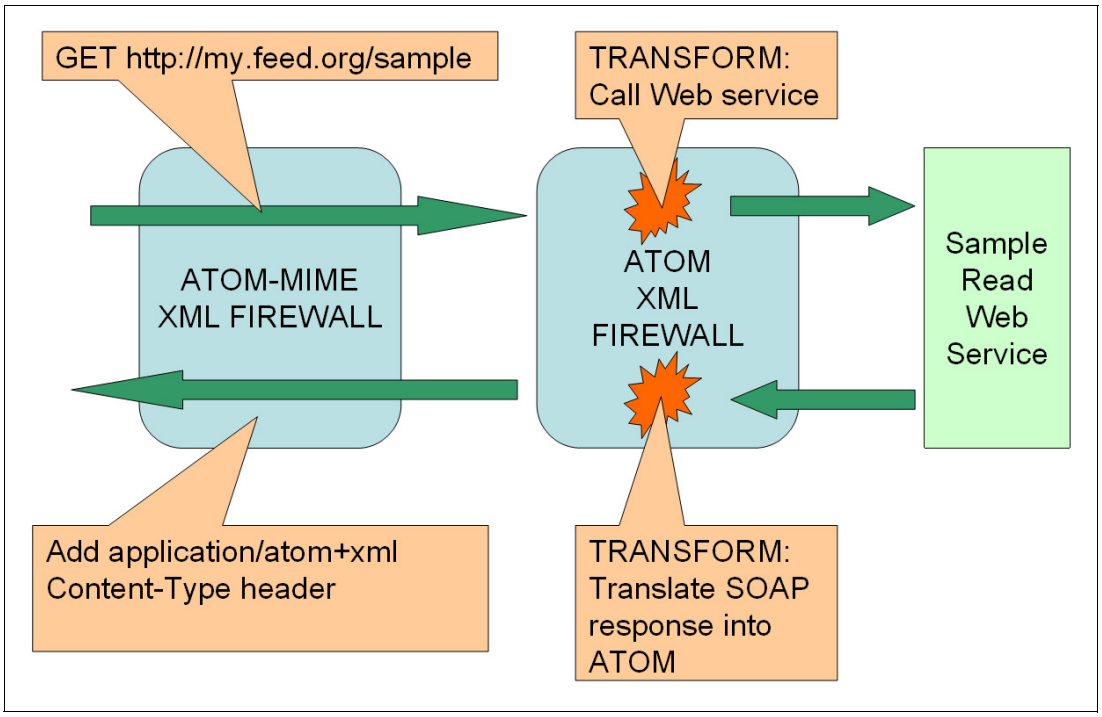


Figure 7-1 Logical architecture of the DataPower XML firewall

The first firewall, which receives the request from the feed reader, forwards the request unchanged to a second loopback firewall. The second (loopback) firewall has a processing policy that consists of two XSL transforms. The first of these transforms generates a SOAP request to the SampleRead Web service to obtain the list of customers. The second transform processes the Web service's response and translates it into the Atom format. The translated output then becomes the response input of the first firewall.

The first firewall's response processing policy adds the Atom content type. The content type header for Atom feeds should be `application/atom+xml`, where it is `application/text+xml` for Web services.

Creating the Web service facing the XML firewall

To configure the DataPower appliance, perform the following steps. We configure the second firewall first.

1. From the Control Panel, under Service, click **XML Firewall**.
2. In the Configure XML Firewall page, click **Add Wizard** to create a new XML firewall.
3. Click **Pass Thru** to configure the basic properties with the wizard. We customize the XML firewall later.

4. On the Create a Pass Thru XML Firewall Service page (Figure 7-2), complete these steps:
   a. For Firewall Name, type `atom`.
   b. For Firewall Type, select `loopback-proxy`.
   c. For Device Address, type `0.0.0.0` for the clients to connect to the XML firewall. This means that the firewall is listening on all active interfaces.
   d. For Device Port, type `3000`.
   e. Click **Commit** to save the firewall.

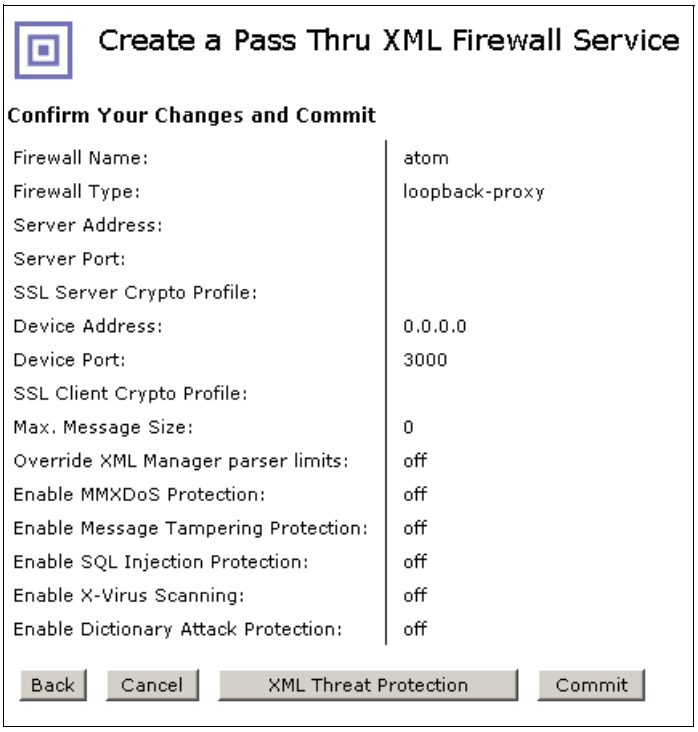


Figure 7-2 The atom XML Firewall Service

Figure 7-3 shows that the XML Firewall atom was successfully created.

Figure 7-3 Finishing the wizard

5. On the main XML Firewall page, click the newly created firewall.
6. For Request Type at the bottom right side of the page, select **Non-XML** so that the initial request from the feed reader is not in an XML format.
7. For the atom Firewall Policy, click the **...** button to customize the policy.

8. Drag a **Transform** action onto the processing rule diagram. Double-click the new **Transform** action icon to configure this action (Figure 7-4).

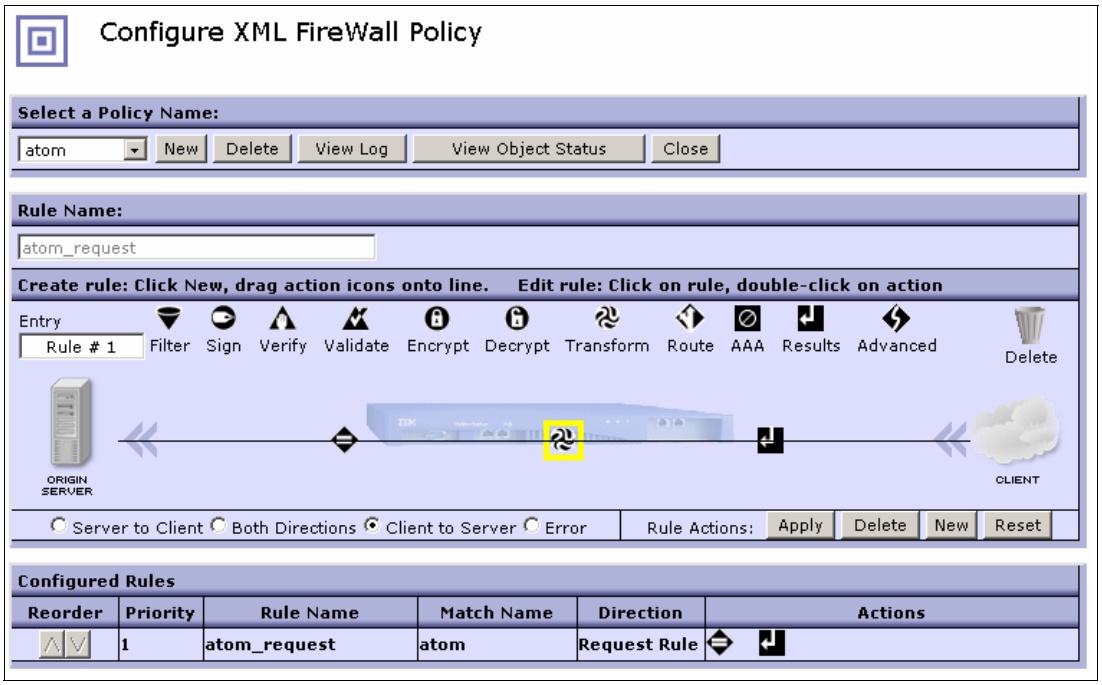


Figure 7-4 Configure the XML Firewall Policy page

9. On the Configure Transform Action page (Figure 7-5), upload the **genRequest.xsl** file that we want to use. This is the XSLT file that sends an out-of-band SOAP request to a Web service and receives a SOAP response with the requested list of items. Click **Done**.

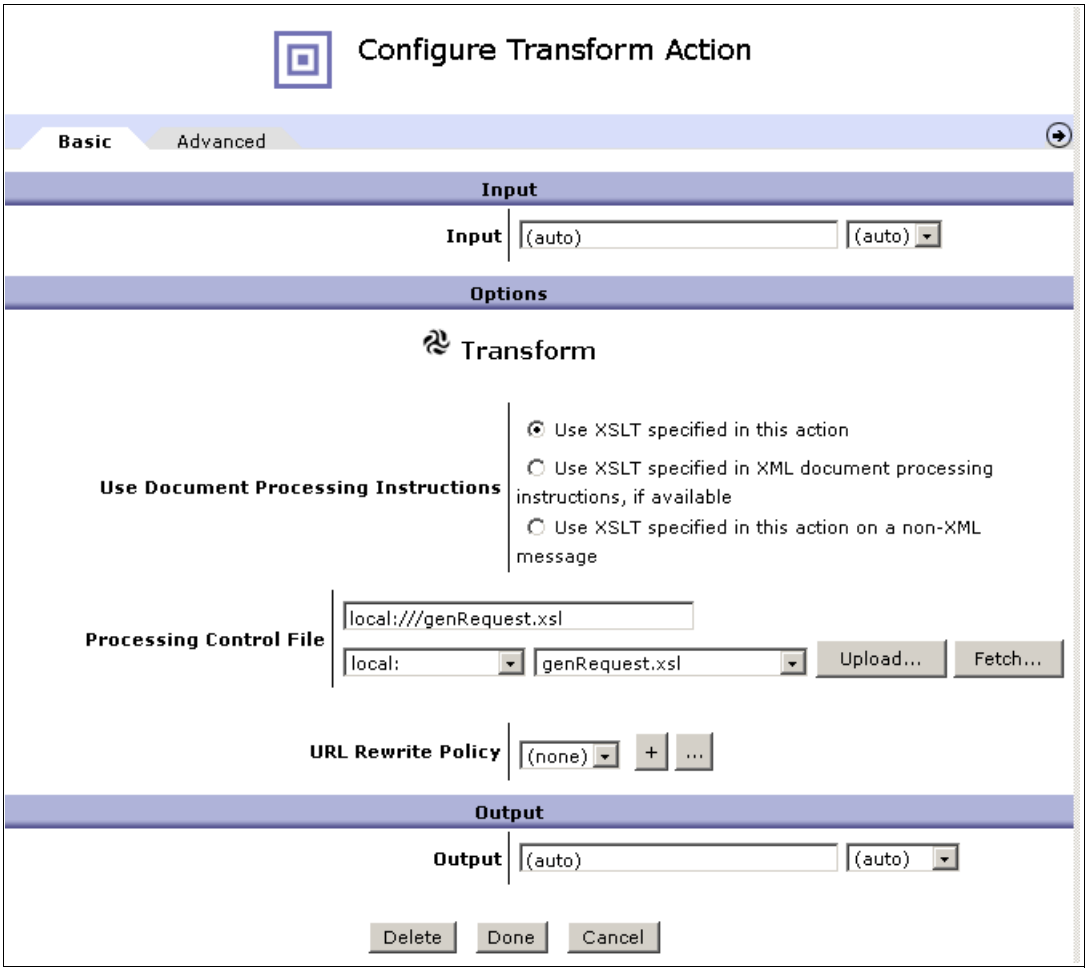


Figure 7-5 Configuring the Transform action

10. On the policy diagram, insert another **Transform** action after (to the right of) the existing Transform action. Double-click the newly created **Transform** action (Figure 7-6).

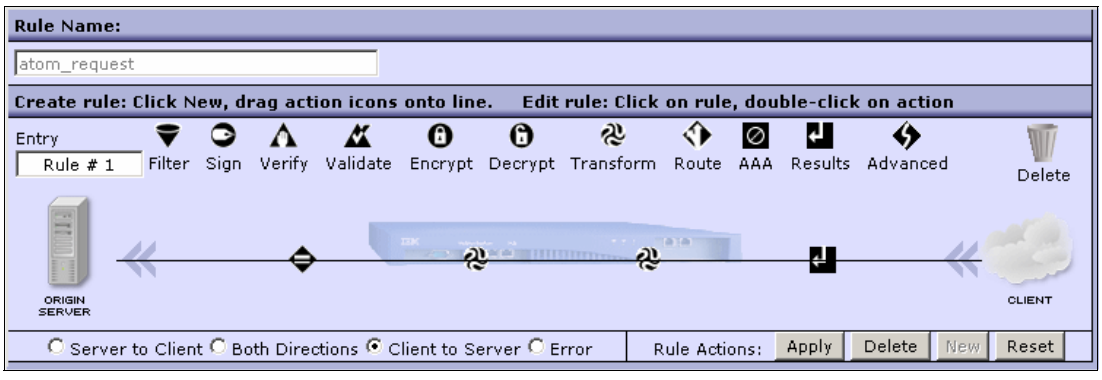


Figure 7-6 The final rule for the atom XML firewall

11. Configure the action to use the **atomResponseMapper.xsl** file. This stylesheet performs the transformation of the SOAP response from the previous step to the Atom format. Apply the changes to this rule and close the policy editing window.
12. Click **Apply** and **Save Config** to save the changes.

Creating the client facing XML firewall

Now we create the XML firewall that receives requests from a feed reader. We use the wizard as we did before. Name the firewall `atom-mime` and perform the following steps:

1. For the firewall type, select **static-backend**, so that we can define a front-end system and a back-end system.
2. In the back-end configuration (Figure 7-7), enter the address and port of the XML Firewall Service that we created before. For Server Address, type 127.0.0.1 to indicate a local service. For Server Port, type `3000`, which we defined previously. For Device Port, type `3001` for this firewall.

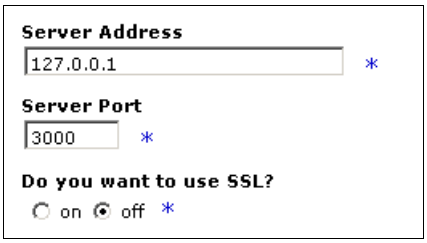


Figure 7-7 Back-end connection configuration

3. On the next page (Figure 7-8), click **Commit** to commit the configuration of this firewall.

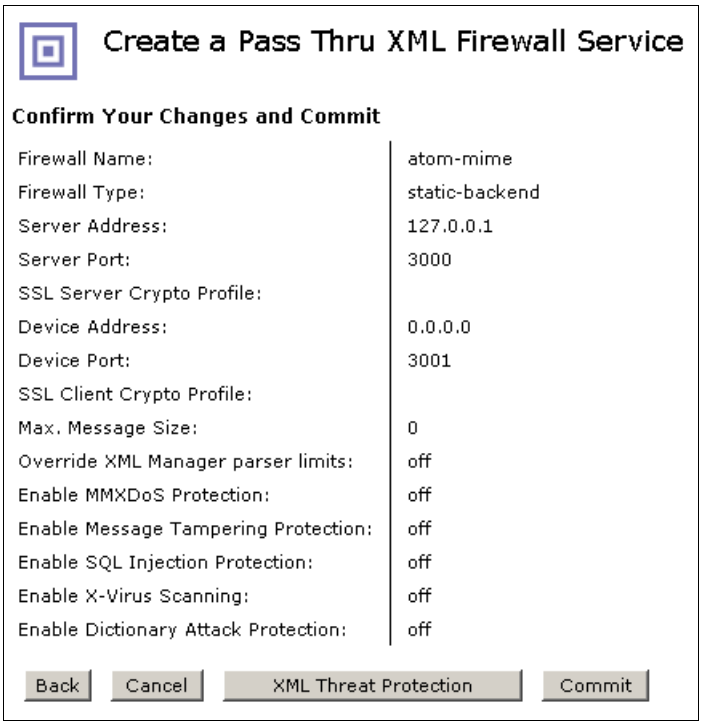


Figure 7-8 XML Firewall 'atom-mime'

4. On the Configure XML Firewall page (Figure 7-9), select the newly created **atom-mime** XML firewall.

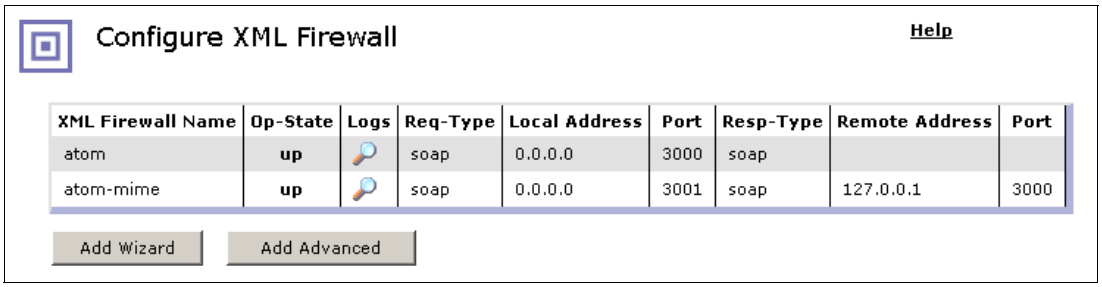


Figure 7-9 The two created XML firewalls

5. Set the Request Type and Response Type to **Non XML**.
6. Click the **...** button beside the atom-mime Firewall Policy to customize it.
7. Create a new Rule Action and make it a Server to Client (Response Rule).
8. Drag an **Advanced** action to the processing rule diagram. Double-click the **Advanced** action.
9. Select the **setvar** (Set Variable) option.
10. On the Set Variable page (Figure 7-10), for Variable Name, type `var://service/set-response-header /Content-Type`. For Variable Assignment, type `application/atom+xml`. Click **Done**.

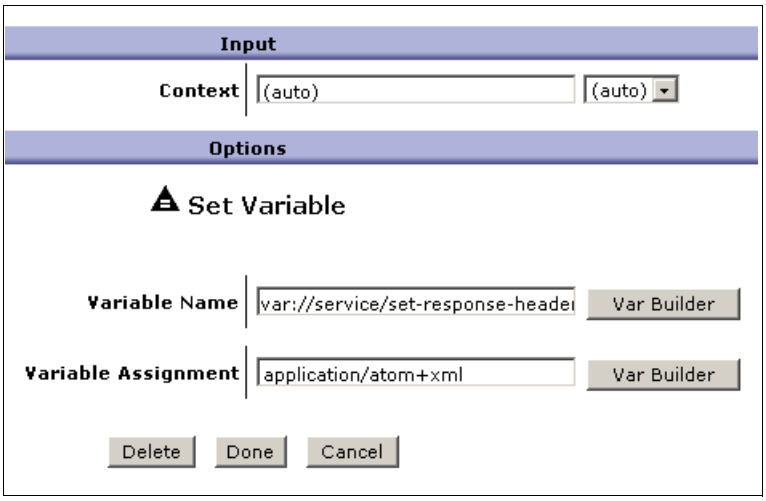


Figure 7-10 Setting a variable

11. On the processing policy window (Figure 7-11), double-click the matching rule icon (=) to open the configuration window for the action.

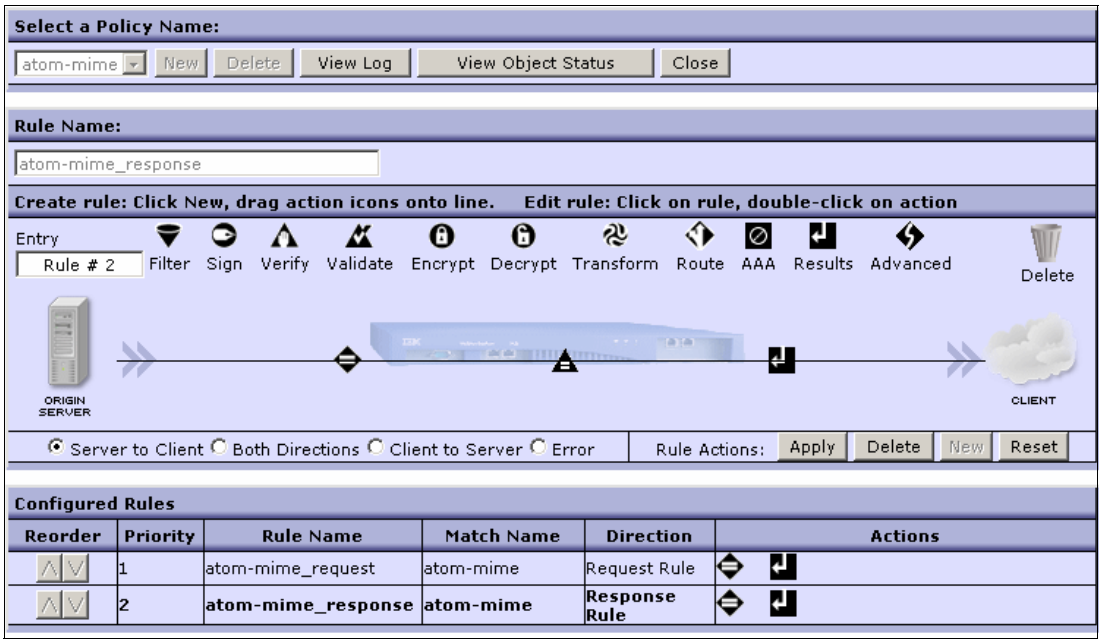


Figure 7-11 The atom-mime firewall policy

12. Click the **...** button next to the atom-mime matching rule.
13. On the **Rules** tab (Figure 7-12), change the URL Match field to `/feed*`. Click **Save**.

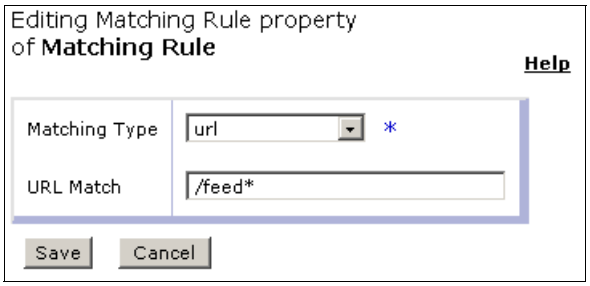


Figure 7-12 Creation of a new Matching Rule

14. Apply the changes to the processing policy and close the policy editing window.
15. On the main configuration page, click the **Advanced** tab and set Disallow GET (and HEAD) to **ON** so that we can *allow* GET requests for this firewall.
16. Click **Apply** and click **Save Config** to commit the changes to the atom-mime XML firewall.

7.3 Demonstration

By turning on the probe for the firewalls, we can view the data as it traverses the appliance. To test the configuration, we direct a feed reader to the client-facing XML firewall called atom-mime. This firewall has the port number 3001. Therefore, the feed reader should use the following URL:

```
http://<datapower host name>:3001/feed
```

Requests that are sent to this firewall are sent unchanged to the atom XML firewall. By viewing the probe for the atom firewall, we can see that the first transform action sends a SOAP request to the Web service. The SOAP response is then translated by the second transform into the ATOM format, as shown in Example 7-4.

Example 7-4 Translated SOAP response

```
<atom:feed xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:date="http://exslt.org/dates-and-times"
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:dp="http://www.datapower.com/schemas/management">
<atom:id>atom.sample.com</atom:id>
<atom:title>feed demo: List of Customers</atom:title>
<atom:updated>2007-11-14T22:56:58+11:00</atom:updated>
<atom:entry>
<atom:id>0</atom:id>
<atom:title>customer0</atom:title>
<atom:updated>2007-11-14T22:56:58+11:00</atom:updated>
<atom:content>
<Customer xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<CustomerID>0</CustomerID>
<CustomerName>customer0</CustomerName>
</Customer><
/atom:content>
</atom:entry>
</atom:feed>
```

The ATOM content is then returned to the atom-mime firewall, where the response processing rule is invoked. By viewing the response probe for the atom-mime firewall, we can see that the content-type header has been updated by the Set Variable action from `application/text+xml` to `application/atom+xml`.

7.4 Summary

In this chapter, we demonstrated how you can use the DataPower appliances to enable existing applications to support Web 2.0 technologies. Web 2.0 entities, such as blogs, wikis, and social networking are becoming increasingly popular as a means for both individuals and companies to interact with one another.

By using a simple XSL transformation, we have shown how SOAP-enabled Web services can provide content in the Atom Syndication Format. This format is widely used to provide syndicated news and other informational content to users. The XSL transformation performed by the DataPower appliance converted the Web service's SOAP-based response into the Atom format by adding the necessary Atom metadata around the Web service response.

A

XSL programming issues

In this appendix, we provide support information in regard to Chapter 6, “XSLT programming” on page 123. We begin with information about the **curl** commands that we used to complete the provided examples. Then we explain details about the XML firewall that we created in 6.5, “Example 3: GET request transformed into a SOAP message” on page 147.

DataPower domain: `XSLTProgramming` is the DataPower domain that we used to illustrate the examples.

The cURL commands

In this section, we explain the different **curl** commands that we used for each example in Chapter 6, “XSLT programming” on page 123.

Example 1

We used the following **curl** command for example 1:

```
curl --data-binary @getPrime_EX1.xml http://9.42.170.230:4012/hello
```

Example 2

We used the following **curl** command for example 2:

```
curl --data-binary @getPrime_EX2.xml http://9.42.170.230:4012/hello
```

Example 3

We used the following **curl** command for example 3:

```
curl "http://9.42.170.230:4013/get2soap?op=getPrime&value=2"
```

Example 4

We used the following **curl** command for example 4:

```
curl --data-binary @getPrime_EX4.xml http://9.42.170.230:4012/hello
```

Example 5

We used the following **curl** command for example 5:

```
curl "http://9.42.170.230:4013/log?op=getPrime&value=2"
```

Example 6

We used the following **curl** command for example 6:

```
curl --data-binary @getPrime_EX6.xml http://9.42.170.230:4012/hello
```

XML firewall configuration details

In this section, we show details about XML firewall that we used in 6.5, “Example 3: GET request transformed into a SOAP message” on page 147. Figure A-1 shows the main details of XML firewall.

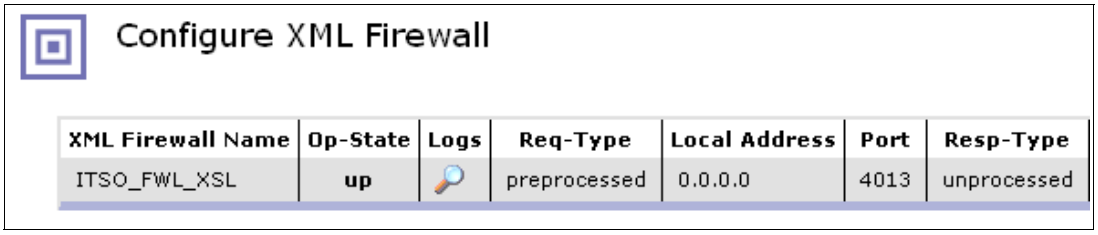


Figure A-1 XML firewall main details

General configuration

Figure A-2 and Figure A-3 show the general configuration of the XML firewall.

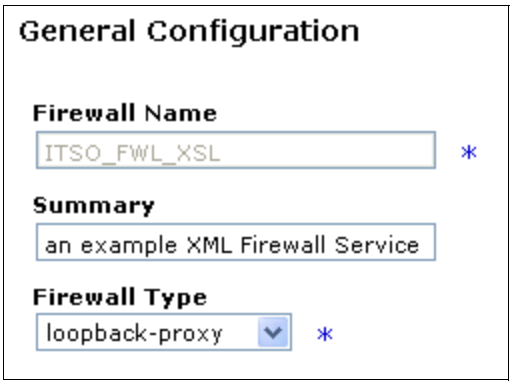


Figure A-2 General configuration (part 1 of 2)

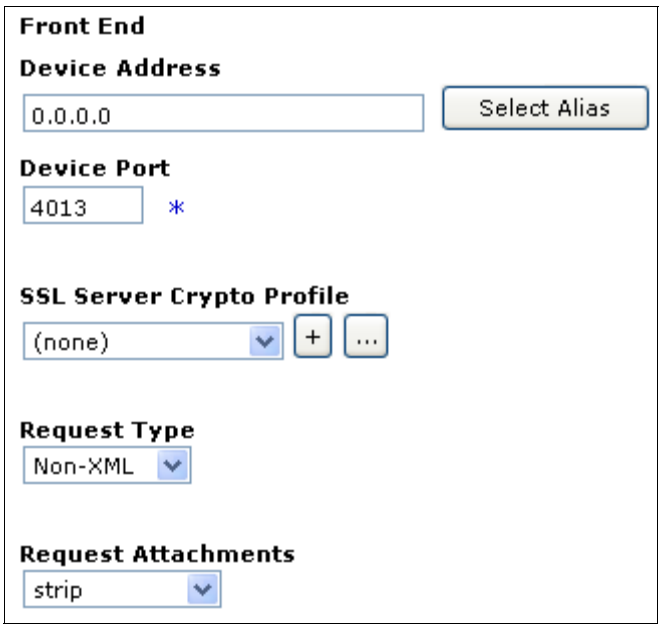


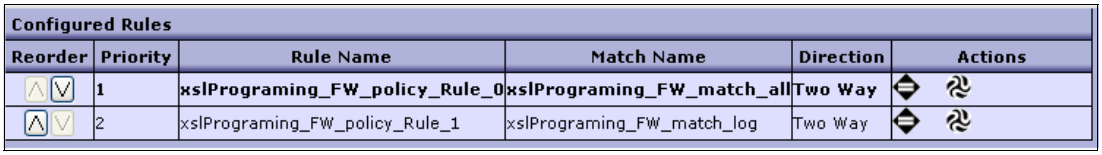
Figure A-3 General configuration (part 2 of 2)

Non-XML request type: We use the *Non-XML* request type to handle incoming GET requests.

Processing rules

Figure A-4 shows the two processing rules that are defined on the XML firewall. The XML firewall contains the following two rules, which respectively match the `/get2soap` and `/log` incoming URLs:

- `xslProgramming_FW_policy_Rule_0`
- `xslProgramming_FW_policy_Rule_1`



Configured Rules

Reorder	Priority	Rule Name	Match Name	Direction	Actions
	1	xslPrograming_FW_policy_Rule_0	xslPrograming_FW_match_all	Two Way	
	2	xslPrograming_FW_policy_Rule_1	xslPrograming_FW_match_log	Two Way	

Figure A-4 XML firewall procesing rules

B

Additional material

This paper refers to additional material that can be downloaded from the Internet as described below.

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/REDP4327

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redpaper form number, REDP4327.

Using the Web material

The additional Web material that accompanies this paper includes the following files:

File name	*Description*
redp4327.zip	Compressed code samples

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 40 MB minimum
Operating System: Microsoft Windows®
Processor: 2 GB or higher
Memory: 2 MB

How to use the Web material

Create a subdirectory (folder) on your workstation, and extract the contents of the Web material compressed 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

For information about ordering these publications, see “How to get Redbooks” on page 186. Note that some of the documents referenced here may be available in softcopy only.

- *Enabling SOA Using WebSphere Messaging*, SG24-7163
- *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
- *Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB*, SG24-7369
- *WebSphere Message Broker Basics*, SG24-7137

Other publications

The following publications are also relevant as further information sources. They are either available as part of the product or orderable for a fee. The common documentation is available on the Web at the following address:

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

- *IBM WebSphere DataPower Example Configurations Guide*
- *IBM WebSphere DataPower Common Installation Guide*
- *IBM WebSphere DataPower Integration Appliance XI50 Reference Kit*, part number 42C4212
- *IBM WebSphere DataPower WebGUI User's Guide*
- *IBM WebSphere DataPower XML Integration Appliance XI50 CLI Reference Guide Release 3.6.0*
- *IBM WebSphere DataPower XML Accelerator XA35 Reference Kit*, part number 42C4210
- *IBM WebSphere DataPower XML Security Gateway XS40 Reference Kit*, part number 42C4211

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 Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, 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 I: Overview and Getting Started

Understand and effectively deploy DataPower SOA appliances

Parse and transform binary, flat text, and XML messages

Learn how to extend your SOA infrastructure

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, secure, and accelerate XML and Web services deployments while extending the 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 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 identify the circumstances under which to deploy DataPower appliances. They include a sample implementation and architectural best practices for an SOA message-oriented architecture in an existing production ESB environment.

This part of the series provides a general overview of DataPower SOA appliances and a primer to using the appliances in common scenarios. The entire 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-4327-00

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks