WebSphere. software

IBM

Integrating Back-end Systems with WebSphere Application Server on z/OS through Web Services

Service-oriented scenarios

ESB scenarios

Integration with CICS, IMS, and DB2

Alex Louwe Kooijmans
G Michael Connolly
Sergio Straessli Pinto
Karthik Shanmugam
Ronald Townsend
Mario Visoni

Redbooks

ibm.com/redbooks

International Technical Support Organization

Integrating Back-end Systems with WebSphere Application Server on z/OS through Web Services

September 2008

SG24-7548-00

Note: Before using this information and the product it supports, read the information in "Notices" on page VII.

First Edition (September 2008)

This edition applies to WebSphere Application Server Version 6.1.0.10.

Contents

Notices

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

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

CICS®
Cloudscape®
DB2®
Domino®
IBM®
IMS™
Informix®
iSeries®
Language Environment®
Lotus®
MVS™
OS/390®
RACF®
Rational®
Redbooks®
Redbooks (logo) ®
REXX™
System z™
WebSphere®
z/OS®
zSeries®

The following terms are trademarks of other companies:

Adobe, and Portable Document Format (PDF) are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

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

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

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

Intel, Pentium, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Preface

Service Oriented Architecture (SOA) requires a new way of looking at integration between subsystems. The primary subsystems of interest on z/OS® are CICS®, IMS™, and DB2®. There are many ways to integrate WebSphere® Application Server on z/OS with those subsystems.

In this IBM® Redbooks® publication, we focus on Web Services integration scenarios and scenarios using an Enterprise Service Bus (ESB) solution. In the IBM Redbooks publication *WebSphere for z/OS V6 Connectivity Handbook*, SG24-7064, we elaborate on both Web Services and J2C scenarios.

The scenarios in this book are described from a development perspective. In the Additional material, a workspace is included for most scenarios.

The team that wrote this book

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

Alex Louwe Kooijmans is a Project Leader with the International Technical Support Organization (ITSO) in Poughkeepsie, NY. He specializes in WebSphere, Java™ and SOA on System z™ with a focus on integration, security, high availability and application development. He previously worked as a Client IT Architect in the Financial Services sector with IBM in The Netherlands, advising financial services companies on IT issues such as software and hardware strategy and on demand. Alex has also worked at the Technical Marketing Competence Center for zSeries® and Linux® in Boeblingen, Germany, providing support to customers starting up with Java and WebSphere on zSeries. From 1997 to 2002, Alex completed a previous assignment with the ITSO, managing various IBM Redbooks projects and delivering workshops around the world from Poughkeepsie.

G Michael Connolly is an IT consultant at the International Technical Support Organization, Poughkeepsie Center. He has more than 30 years of IBM software development experience in both distributed systems and the mainframe zSeries. He holds a BA in Humanities from Villanova University. Michael's areas of expertise include TCP/IP Communications, UNIX® System Services, EWLM, and WebSphere for z/OS.

Sergio Straessli Pinto is an IT Specialist with IBM Brazil in Integrated Technology Delivery, Server Systems Operations. He has worked at IBM for 11 years. He has four years of experience as a Support Analyst in WebSphere MQ and WebSphere Message Broker on distributed environments and the WebSphere Application Server on z/OS environment. He has also worked on developing software using the COBOL/CICS command level, and Visual Basic® using an Oracle® database. Sergio holds a Bachelor's degree in Business Administration and Accounting from Instituto Catolico de Minas Gerais, Brazil.

Karthik Shanmugam is a Software Engineer with Wachovia Bank, USA. He has more than 10 years of experience in developing software and is a member of the Technology Services Division supporting Web Services-based distributed access to transactional mainframe resources (IMS, CICS, DB2) on the z/OS platform, called the Integration Hub. His areas of expertise include design; support components, applications, and products on the J2EE™ platform; and specialization in the WebSphere on z/OS technology stack. Karthik holds a Master's degree in Computer Science from the University of New Mexico, Albuquerque, USA.

Ronald Townsend is a Staff Software Engineer in the Systems and Technology Lab Services Group and has been with IBM for 10 years. He holds a Bachelor's degree in Computer Science from Purdue University. His area of expertise is utilizing SOA and Integration in transforming traditional host applications on System z.

Mario Visoni is a Senior IT Specialist with IBM Brazil. He has five years of experience in WebSphere with the Software Support Center and in project delivery. Most recently he has supported WebSphere on z/OS at Banco do Brasil. Mario holds a degree in Computer Science from Mackenzie University and a Post Graduate degree in Business Administration from Universidade Paulista. His areas of expertise include WebSphere for distributed systems and on zSeries.

The Redbook team, from let to right: Karthik Shanmugam, Sergio Straessli Pinto, Mario Visoni, Alex Louwe Kooijmans, G. Michael Connolly, Ronald Townsend

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

Reginaldo Barosa
IBM Software Group, TechWorks

Richard Conway
International Technical Support Organization, Poughkeepsie Center

John Diamond
IBM Systems and Technology Group, Lab Services

Michael Schenker
IBM Software Group, Information Management, Santa Theresa lab

Carl Farkas
IBM Software Group, France, WebSphere Business Integration for System z

Mitch Johnson
IBM Software Group, USA, Application and Integration Middleware Software

Subhajit Maitra
IBM Advanced Technical Support (ATS), Americas

Peggy Rader
IBM Software Group, Information Management

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 books to be as helpful as possible. Send us your comments about this book 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

This book is about integrating WebSphere on z/OS with backend systems. We focus on SOA integration styles using either direct Web Services or an Enterprise Service Bus (ESB).

We have chosen to use a limited number of scenarios that can be easily repeated instead of providing a description of functions and features of all theoretically possible solutions. All scenarios described in this book have been tested on the ITSO systems.

To help you in making architecture decisions on integration, we refer you to *SOA Transition Scenarios for the IBM z/OS Platform*, SG24-7331.

In the following section we summarize the scenarios covered in this book. We also remind you that most of the information in *WebSphere for z/OS V6 Connectivity Handbook*, SG24-7064 is still applicable and can be used for additional integration technologies and scenarios.

1.1 Scenarios discussed

The following scenarios are discussed in this book:

- Direct integration between WebSphere and CICS on z/OS using Web Services over HTTP.
  We included both an inbound and an outbound scenario.
- Direct integration between WebSphere and CICS on z/OS using Web Services over JMS.
- Direct integration between WebSphere and DB2 on z/OS using Web Services.
  We used the new DB2 V9 Data Web Services Toolkit for this purpose.
- Direct integration between WebSphere and IMS on z/OS using Web Services. This scenario is making use of the new IMS SOAP Gateway on z/OS.
- Integration between WebSphere on z/OS and backend systems using WebSphere Message Broker.
- Integration between WebSphere on z/OS and backend systems using WebSphere ESB.

1.2 Tooling used

For each scenario we have used the recommended IBM tooling with the level available at the time of writing. At the time of publication or thereafter, newer versions of tools may become available. The scenarios can generally be executed with newer versions of the tools, but there may be differences in the screens, the options chosen, and the buttons clicked.

1.3 Additional material

In the additional material we have included source files, workspaces, and so on of the components we developed. Refer to Appendix B, “Additional material” on page 397 for details.

2

Accessing CICS from WebSphere using Web Services over HTTP

We discuss our experience using Rational Developer for z Version 7.1 (WDz) to develop and deploy a solution to access CICS Transaction Server from WebSphere Application Server using Web Services over HTTP. We describe how to generate the artifacts and deploy them to CICS Transaction Server and WebSphere Application Server.

We use an existing CICS/COBOL application called CUSTINQ, which is part of the CICS “Catalog Order” application. This application is a traditional CICS application with several 3270 screens.

The chapter is organized as follows:

- In “Overview” on page 5 we discuss the architecture of the scenario and the infrastructure to develop, deploy and run the solution.
- In “Components used” on page 7 we provide more details on the technology used in this scenario.
- In “Development approach” on page 15 we talk about how to develop a CICS Web Services solution.

- In “The steps in detail” we go through deployment of the CICS-related artifacts and in 2.5.8, “Deployment to WebSphere Application Server” on page 78 we describe deployment of the WebSphere Application Server Web Service consumer application.

2.1 Overview

Our scenario is based on the use of Web Services technology, and we basically built a *point-to-point* connection between WebSphere Application Server on z/OS and CICS Transaction Server.

We built our environment with CICS being the Web Service *provider* and WebSphere Application Server being the Web Service *consumer*. The COBOL application running in CICS is invoked by the Web Service consumer application running in WebSphere Application Server. Both WebSphere Application Server and CICS Transaction Server support SOAP messages to be sent and received over either HTTP or JMS. In this scenario we chose to use HTTP as the *transport* protocol.

Rational Developer for z Version 7.1 (WDz) is used to build the artifacts that make the interoperation between these components possible. WDz uses the XML Services for the Enterprise (XSE) feature to generate the Web Services interfaces for the existing COBOL program. After a WSDL file is generated, you can use this file to create a Web Services consumer application, which is then deployed to WebSphere Application Server.

Figure 2-1 shows a diagram of the solution.

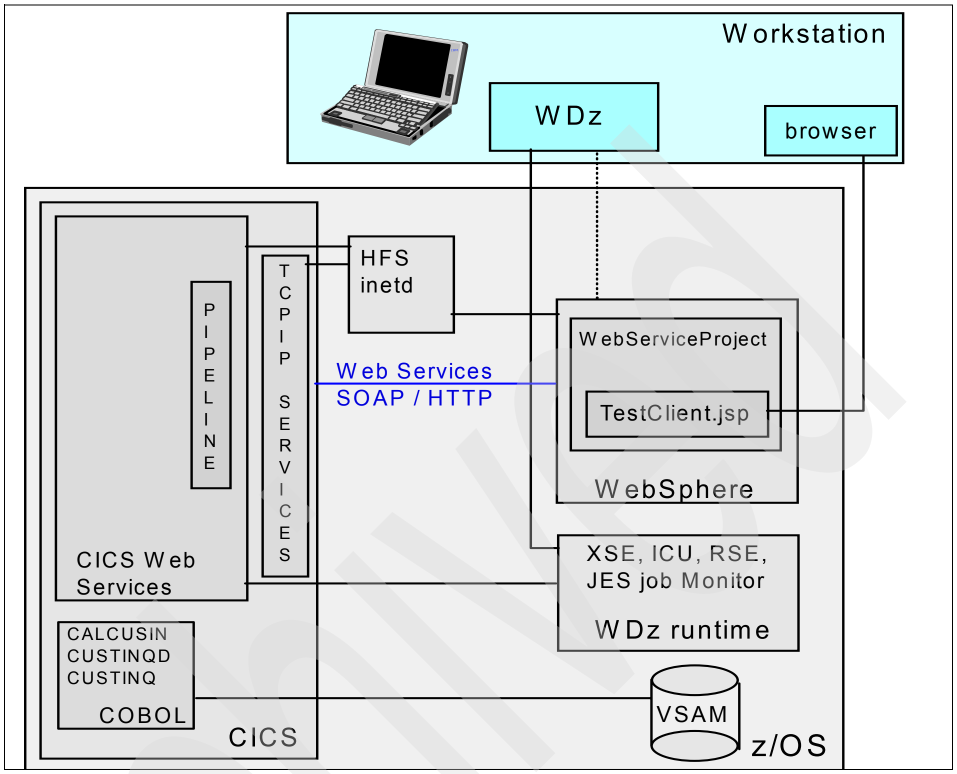


Figure 2-1 Overview of CICS Web Services over HTTP

Note: Although our description in this book is based on WebSphere Developer System z Version 7, the same scenario can also be developed using Rational® Application Developer for System z version 7.1.1. There are only some minor differences in the wizards and panes.

2.1.1 Steps to implement the scenario

The steps to implement this scenario are:

1. Collect the CICS program sources and copybooks.
2. Generate the Web Service artifacts based on the existing COBOL code.
3. Deploy the generated artifacts to CICS on z/OS.
4. Test the Web Service in CICS using the Web Services Explorer in RDz.

5. Generate a client interface to the Web Service based on the generated WSDL.
6. Build a client (Web Service consumer) application.
7. Deploy the client application to WebSphere Application Server on z/OS.
8. Test the solution by using a browser pointing to the URL of the client application in WebSphere Application Server.

2.2 Components used

The objective of this book is to demonstrate the connectivity of WebSphere and not the installation and configuration of the products, but we list the required components of the infrastructure for this scenario in the following sections.

2.2.1 Websphere Developer for System z

WebSphere Developer for System z version 7 is the product used to develop the artifacts used to make the existing COBOL CICS application available through Web Services and generate the Web Service client to be deployed to WebSphere Application Server.

WDz consists of a workbench installed on a Windows® or Linux workstation and a runtime that is installed on z/OS in our case.

- WDz workbench on the workstation

  The WDz workbench is a full Eclipse-based IDE used to:

  – Develop the application

  – Generate artifacts

  – Connect to the host to perform remote z/OS operations, such as compiling and link-editing and testing

  We used the workbench on a workstation with a 2.33 Ghz processor and 1.5 GB RAM running Windows XP Professional Service Pack 2 under a VMWARE image. This would be the bare minimum configuration to use the workbench in a VMWare image with a reasonable performance.

- WDz runtime on z/OS

  The z/OS runtime responsible for allowing the workbench on the workstation to connect to z/OS and request z/OS operations.

In “Prerequisites for WDz” on page 8 we provide the prerequisites for WDz.

Prerequisites for WDz

We cover the requirements for Rational Developer for z Version 7.1 in this section, but we do not explain how to install or configure the product. Refer to the appropriate documentation for additional information.

You can find the complete list of requirements for WDz at:

`http://www-306.ibm.com/software/awdtools/wdzseries/reqs`

Or, if you are using Rational Developer for System z, you can find them at:

`http://www-306.ibm.com/software/awdtools/rdz/sysreqs`

You can also check the product publication *IBM WebSphere Developer for System z Prerequisites*, SC31-6352 or *IBM WebSphere Developer for System z Prerequisites,* SC31-6352, if you are planning to use Rational Developer for System z.

Workstation hardware prerequisites

Table 2-1 shows the minimum hardware requirements for the workstation, but not using VMWare. Running WDz under VMWare has higher hardware prerequisites.

Table 2-1 Hardware requirements for RDz

Hardware	**Requirements**
Processor	800 MHz Pentium®(R) III or higher
Memory	Minimum: 768 MB RAM
Disk space	Minimum 2 GB of disk space is required to install the product. An additional 650 MB of disk space is required in the TEMP directory.
Display	1024 x 768 display minimum using 256 colors (higher is recommended)
Other	Microsoft® mouse or compatible pointing device

However, to reach a good level of productivity, we recommend at least double the processor speed and RAM.

Workstation software prerequisites

The following operating systems are supported:

- Windows XP Professional with Service Pack 2
- Windows 2000 Professional with Service Pack 4
- Windows 2000 Server with Service Pack 4
- Windows 2000 Advanced Server with Service Pack 4

- Windows Server® 2003 Standard Edition with Service Pack 1
- Windows Server 2003 Enterprise Edition with Service Pack 1

Note: These operating systems support all of the national languages that are supported by WebSphere Developer for System z.

Important: Rational Developer for z Version 7.1 was developed for use with an Eclipse IDE of version 3.2.1 or greater that uses at least version 1.5 of the IBM Java Development Kit (JDK™). You may only extend an existing Eclipse IDE that meets these requirements.

Additional software requirements

Use one of the following Web browsers to view the readme files and the installation guide:

- Microsoft Internet Explorer® 6.0 with Service Pack 1
- Mozilla 1.6 or 1.7
- Firefox 1.0.x or 1.5

To properly view multimedia user assistance (such as Tours, Tutorials, and Show me viewlets, you must install Adobe® Flash Player Version 6.0, 65 or later.

For information about supported database servers, Web application servers, and other software products, refer to the online help.

User privileges requirements

You must have a user ID that meets the following requirements before you install RDz:

- Your user ID must not contain double-byte characters.
- You must have a user ID belonging to the Administrators group.

WDz host prerequisites

Use of Rational Developer for z Version 7.1 requires that you have z/OS v 1.6 or higher, including batch functions with the appropriate prerequisites.

The z/OS operating system needs to have the following components installed, configured, and operational:

- Binder
- High Level Assembler
- Interactive System Productivity Facility (ISPF)

- IBM Language Environment®
- Unix System Services
- HFS
- RACF® or equivalent
- APPC
- IP Services component of IBM Communications Server
- One of the following levels of REXX™ must be installed on the host:
  - 5695-014 - REXX/370 Library V1.3
  - 5695-014 - REXX/370 Alternate Library (FMID HWJ9143)

Note: A free version of the REXX/370 Alternate Library is available from the product Web site at:

`http://www.ibm.com/software/awdtools/rexx/rexxzseries/`

Using this free version will affect performance.

You must install IBM Language Environment fixes based on your operating system level and the language version in use as described in Table 2-2.

Table 2-2 LE PTFs

Operating System Level	Language Version	APAR	PTF
z/OS v 1.6	English	PK01298	UK01099
	Japanese	PK01298	UK01100

To install the software on the operating system, System Modification Program/Extended (SMP/E) Version 3 Release 1 or higher is required. The corresponding program number is 5655-G44.

WDz host corequisites

The following products and other stated software are required to support specific features of WebSphere Developer for System z version 7. The WebSphere Developer for System z version 7 workstation client can be successfully installed without these requisites. However, a stated requisite must be installed and operational at runtime for the corresponding feature to work as designed.

- IBM SDK for z/OS Java 2 Technology Edition One of the levels shown in Table 2-3 on page 11 must be installed on the host to support applications using the RSE server.

Table 2-3 Java PTFs

Program Number	Product Name	PTFs or Service Levels Required
5655–I56	IBM SDK for z/OS Java 2 Technology Edition, Version 1.4	No PTF or Service Level Required
5655–I56	IBM SDK for z/OS Java 2 Technology Edition, Version 1.5	No PTF or Service Level Required

Note: The 64-bit version is not supported.

- COBOL Compiler

  One of the products listed in Table 2-4 must be installed on the host to compile COBOL programs developed or edited in WebSphere Developer for System z version 7.

Table 2-4 COBOL PTFs

Program Number	Product Name	PTFs or Service Levels Required
5655–G53	IBM Enterprise COBOL for z/OS V3.1 (minimum required level)	UQ75902
5635–G53	IBM Enterprise COBOL for z/OS V3.2	No PTF or Service Level Required
5635–G53	IBM Enterprise COBOL for z/OS V3.3	No PTF or Service Level Required

Note: IBM Enterprise COBOL for z/OS v 3.1 is the minimum level of the compiler required to enable integrated translator support for CICS and integrated coprocessor support for DB2.

- IBM Enterprise COBOL v 3.2 or later is a full function offering for remote debug support. Debug Tool for z/OS and OS/390® is shipped as part of this product.
- Population of the Remote Problems List with compile errors when COBOL programs without any CICS and DB2 statements are supported for all levels of COBOL compilers, beginning with the minimum level.

  The related product Web site is:

  `http://www.ibm.com/software/awdtools/cobol/zos/`

- CICS Transaction Server

  One of the CICS levels mentioned in Table 2-5 must be installed on the host to support applications with embedded CICS statements.

Table 2-5 CICS PTFs

Program Number	Product Name	PTFs or Service Levels Required
5697–E93	CICS Transaction Server for z/OS v 2.2	No PTF or Service Level Required
5697–E93	CICS Transaction Server for z/OS v 2.3	No PTF or Service Level Required
5697–E93	CICS Transaction Server for z/OS v 3.1	UK15767, UK15764, UK11782, UK11294, UK12233, UK12521, UK15261, UK15271

  Additional Notes:

  – CICS Transaction Server for z/OS v 2.2 has the integrated CICS Translator required to send errors encountered during WebSphere Developer build, JCL generation, and submission to the task list under the following conditions:

    • COBOL or PL/I programs with embedded CICS are built.

    • The minimum compiler level is used: IBM Enterprise COBOL for z/OS v 3.1 or IBM Enterprise PL/I for z/OS and OS/390 v 3.1.

    • The CICS Transaction Server requires additional configuration to work with the Debug tool. The related product Web sites are:
    `www.ibm.com/software/htp/cics/platforms/cicsts/`
    `www-306.ibm.com/software/htp/cics/tserver/v31/`
    `www-306.ibm.com/software/htp/cics/tserver/v23/`

- COBOL Runtime Support for z/OS

  IBM Enterprise Developer Server for z/OS V5.0 provides the runtime libraries for programs that execute on z/OS. These are programs that were developed with either WebSphere Developer for System z or WebSphere Studio Enterprise Developer.

  The runtime libraries provided by IBM Enterprise Developer Server for z/OS V5.0 provide common runtime subroutines that are shared by all Enterprise Generation Language (EGL) programs created with WebSphere Developer or WebSphere Studio Enterprise Developer (for example, data conversion and error management).

Table 2-6 lists what must be installed on the host to support the COBOL Runtime Support for z/OS.

Table 2-6 Enterprise Developer Server PTFs

Program Number	Product Name	PTFs or Service Levels Required
5655–I57	IBM Enterprise Developer Server for z/OS V5.0	UQ84056, UK00137

2.2.2 WebSphere Application Server for z/OS

The WebSphere Application Server is the J2EE and Web Services application server that runs the client application used to test the solution. We used WebSphere Application Server for z/OS Version 6.1.0.10 on a z/OS V1.9 system.

WebSphere Application Server prerequisites

We used WebSphere Application Server for z/OS Version 6.1.0.10; this version requires z/OS v 1.6 or higher.

The WebSphere Application Server uses its own JRE™ instance; it is the J2RE 1.5.0 IBM J9 2.3 level.

To check the system requirements for WebSphere Application Server for z/OS V6.1, refer to the Web page at:

http://www-1.ibm.com/support/docview.wss?rs=404&uid=swg27007657

2.2.3 CICS Transaction Server V3.1

CICS TS is the transaction server where the older application resides. We used CICS TS 3.1 on a z/OS 1.9 system.

CICS Transaction Server prerequisites

We used CICS Transaction Server V3.1, which requires z/OS v 1.4 or higher.

It is beyond the scope of this book to discuss the prerequisites of CICS TS and we assume that you already have CICS installed. In any case, you can find more details at:

http://www-1.ibm.com/support/docview.wss?uid=swg27006364

2.2.4 VSAM

The data store that the CICS application is using is a VSAM data set.

2.3 Host configuration

Some components need to be configured on z/OS in order to execute this scenario. Our focus, however, is to show how to use WDz to make a backend CICS/COBOL application available to WebSphere Application Server through the use of Web Services.

WDz host components configuration

Besides the workstation configuration, WDz has some components to be installed on z/OS. There is also some configuration to be done. Refer to the following online documentation:

- *IBM WebSphere Developer for System z Host Planning Guide,* SC31-6599
- *IBM WebSphere Developer for System z Installation Guide,* SC31-6316
- *IBM WebSphere Developer for System z Host Configuration Guide,* SC31-6930

You can also download a copy from the product library Web page at:

http://www-306.ibm.com/software/awdtools/wdzseries/library/

Network configuration

The success of using WDz depends on the proper configuration and reliability of the network between the developers' workstations and the z/OS host system. WDz uses a variety of protocols to communicate with the z/OS host in both directions. You should make sure that firewalls are open for the TCP/IP addresses and port numbers being used.

You can also refer to the manual *IBM WebSphere Developer for System z Host Configuration Guide,* SC31-6930, where you can find some topics about the network configuration related to our needs.

CICS Transaction Server

We provide some basic configuration information later in this chapter regarding our environment. If you need more details on setting up CICS for Web Services, refer to *Implementing CICS Web Services,* SG24-7206 or the product Web pages at:

http://www-306.ibm.com/software/htp/cics/tserver/support/

http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp?topic=/com.ibm.cics.ts31.doc/prod/home.html

WebSphere Application Server

In our scenario, WebSphere Application Server on z/OS runs the WebServiceProject application. This application serves a Web page taking input. In our solution the input is straightforward and only consists of a customer number. The customer number is the key field used by the CICS/COBOL program to access the VSAM file.

In our environment we did not enable security in WebSphere Application Server. In a normal situation this would be the case.

Again, it is beyond the scope of this book to describe the configuration of WebSphere Application Server on z/OS; we refer to the online documentation at the Information Center:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp

2.4 Development approach

This chapter shows the steps we used in our environment to enable an existing CICS TS COBOL program to be accessed by Web Services using the WDz Enterprise Service Tools (EST) feature, create and deploy the Web Services client application, and invoke the Web Service from WDz and from the Web browser.

We go through the following steps:

1. Creating a *driver/converter* programs and a *WSBind* file that can be deployed to a CICS Transaction Server V3.1 region.
2. Creating Web Services Description Language (WSDL) and XML Schema Definition (XSD) files that describe the CICS-based Web Service.
3. Creating a Web Service client application.
4. Deploying the client to WebSphere Application Server.
5. Testing the client application.

Additionally, this section provides insight into the CICS resources needed on z/OS, including their definition. We also demonstrate the use of the Web Services Explorer Tool to test the new CICS-based Web Service.

2.4.1 WDz Enterprise Service Tools (EST)

The WDz Enterprise Service Tools (EST) is a set of features that aid you in the transformation of COBOL- and PL/I-based business applications, exposing them as Web Services.

Enterprise Service Tools (EST) perspective

The EST perspective in WDz provides views and editors that assist you in developing service flow projects and single-service projects (including Web Services for CICS projects, SOAP for CICS projects, IBM SOAP Gateway projects, and Batch, TSO and USS projects).

XML Services for the Enterprise (XSE)

The XML capability of WDz includes tools that let you easily adapt existing COBOL-based applications to both consume and produce XML messages. This adoption is accomplished without modifying the existing COBOL application and more importantly without modifying the other applications that call the existing COBOL application. A COBOL application must be able to process XML messages to be a Web Service, since XML is the format used to describe the Web Service request and the parameters that are passed to the application.

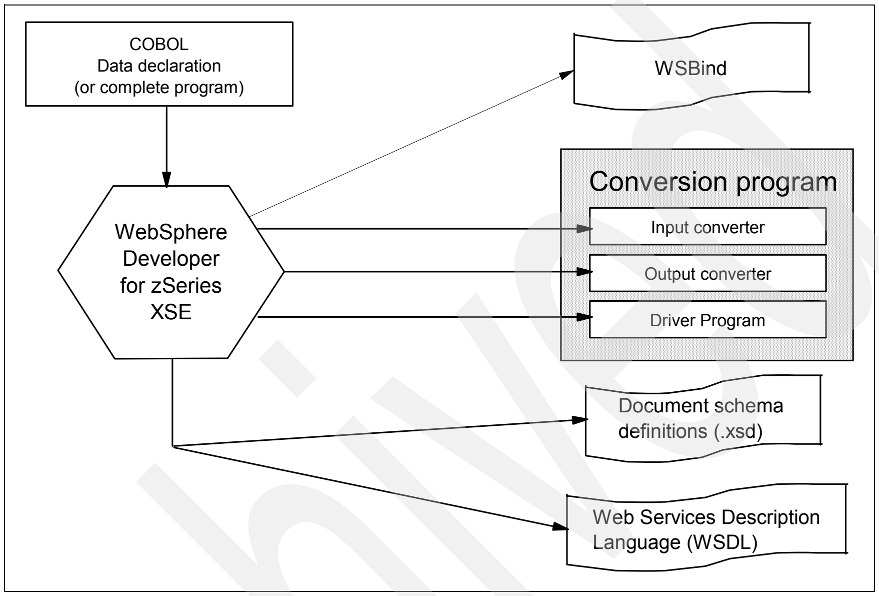
XML Services for the Enterprise (XSE) provide the following wizards:

- The Create New Service Interface (bottom-up) wizard generates a new Web Service interface for an existing COBOL program. Typically, this is called a “bottom-up” approach since the existing COBOL application is at the “bottom” of the new Web Service creation process.

  The scenario described in this chapter uses this capacity.
- The Create New Service Interface (top-down) wizard generates a new Web Service implementation template for a CICS Web Service runtime. Typically, this is called a “top-down” approach since the COBOL application is created “top-down” starting from the existing Web Service definition (typically, being a WSDL file).
- The XML to COBOL mapping wizard and tools help map existing Web Service interfaces or XML data to an existing COBOL program.
- The Batch processor allows you to generate the Web Service interface in unattended “batch” mode. The Batch processor currently supports the “bottom-up” Web Services creation with compiled XML message conversion technology at runtime. The functionality provided by the Batch processor is equivalent to that of the “Create new service interface” (bottom-up) wizard for compiled XML conversions.

The bottom-up approach

This chapter focuses on the bottom-up approach. In this scenario we generate a Web Service description and runtime-specific XML message processing artifacts from a high-level language data structure. This enables you to expose an application program as a service provider. See Figure 2-2.



COBOL Data declaration (or complete program)
WebSphere Developer for zSeries XSE
WSBind
Conversion program
Input converter
Output converter
Driver Program
Document schema definitions (.xsd)
Web Services Description Language (WSDL)

Figure 2-2 Create Web Services in steps

Figure 2-2 depicts the following components:

- WSbind file

  Web service binding file used by CICS Transaction Server v3.1 to perform the mapping between application data structures and XML-based SOAP messages. This file is generated when the Web Services for CICS converter type is specified.

- Web Services Description Language (WSDL) file

  XML-based document describing a Web Service and how to access it.

- Web Services for CICS Conversion “Driver” Program[1]

  COBOL program that performs the input and output XML conversion using the converters. The conversion is needed both before the CICS Web

[1] Note that this asset is generated only if Compiled XML Conversion is chosen as the conversion type versus Interpretative XML Conversion. Same comment for the Inbound/Outbound XML converters and the Inbound/Outbound XSD files. If interpretative is chosen, CICS handles the conversion but there are limitations that are discussed in the next section.

Services runtime calls the unchanged COBOL application, and afterwards to convert the response.

- CICS SOAP Inbound XML converter[1]

  COBOL program that takes an incoming XML document and maps it to the corresponding COBOL data structure that the COBOL application expects.

- CICS SOAP Outbound XML converter [1]

  COBOL program that takes the COBOL data returned from the COBOL application and maps it to an outbound XML document.

- Inbound XML schema definition [1](XSD)

  XML schema that describes the incoming XML document for processing by the input converter.

- Outbound XML schema definition [1] (XSD)

  XML schema that describes the outgoing XML document for processing by the output converter.

2.4.2 CICS Web Services

Previous support for CICS Transaction Server for z/OS V2.2 and V2.3 was provided through the SOAP for CICS feature. This support has now been fully integrated into CICS Transaction Server for z/OS V3.1. With this new release of CICS, applications can now act in the role of both service provider and service requester, where the services are defined using Web Services Description Language (WSDL). The infrastructure provided as part of CICS Transaction Server V3.1 includes a distributed transaction coordination capability compatible with the WS-Atomic Transaction specification.

The support for Web Services includes CICS Web Services Assistant (WSA) integration, a batch utility that can help you to:

- Transform an existing CICS application into a Web Service.
- Enable a CICS application to use a Web Service provided by an external provider.

The CICS Web Services Assistant can create a WSDL document from a simple language structure, or a language structure from an existing WSDL document. It supports COBOL, C/C++, and PL/I. It also generates information used to enable automatic runtime conversion of the SOAP messages to containers and COMMAREAs, and vice versa.

The XML Services for the Enterprise capability of Websphere Developer for System z extends and complements this by providing conversion for COBOL types and constructs which are not covered natively by CICS. For example, OCCURS DEPENDING ON and REDEFINES used in data specification entries are not supported by the CICS Web Services Assistant.

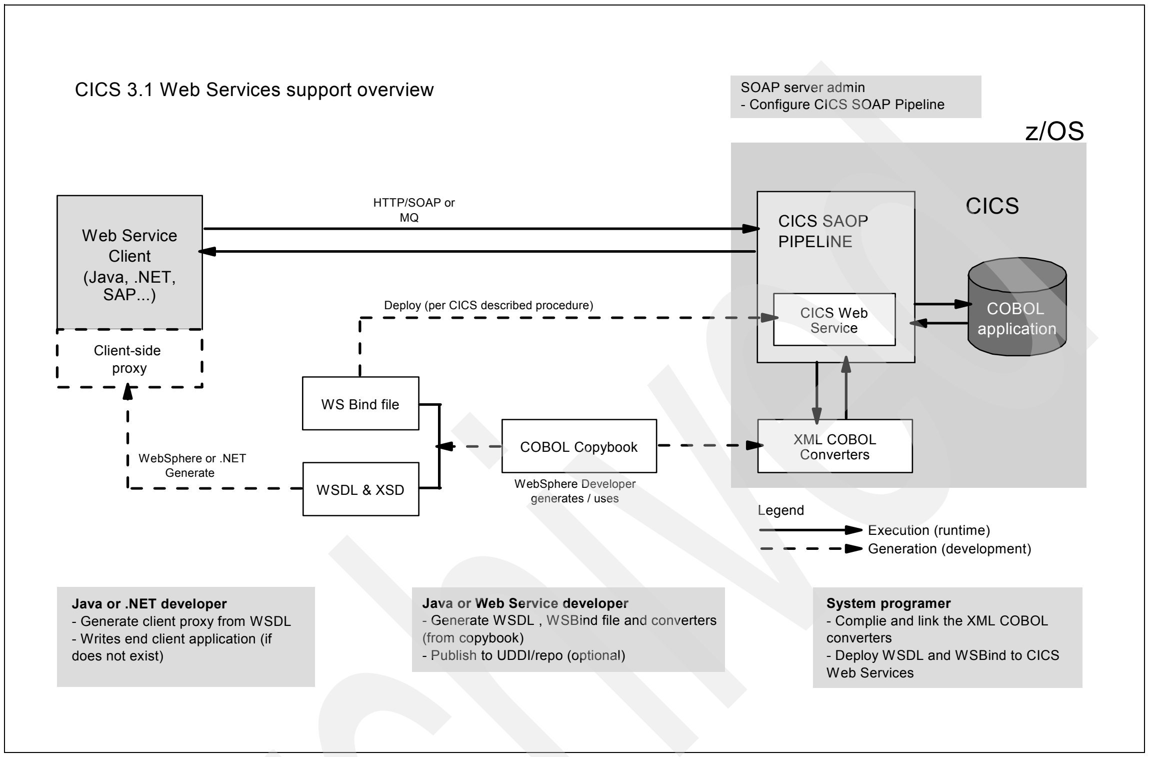

CICS 3.1 Web Services support overview
SOAP server admin
- Configure CICS SOAP Pipeline
z/OS
CICS
Web Service Client (Java, .NET, SAP...)
Client-side proxy
HTTP/SOAP or MQ
CICS SAOP PIPELINE
CICS Web Service
COBOL application
Deploy (per CICS described procedure)
WS Bind file
WSDL & XSD
COBOL Copybook
XML COBOL Converters
WebSphere or .NET Generate
WebSphere Developer generates / uses
Legend
Execution (runtime)
Generation (development)
Java or .NET developer
- Generate client proxy from WSDL
- Writes end client application (if does not exist)
Java or Web Service developer
- Generate WSDL , WSBind file and converters (from copybook)
- Publish to UDDI/repo (optional)
System programer
- Complie and link the XML COBOL converters
- Deploy WSDL and WSBind to CICS Web Services

Figure 2-3 CICS Web Services support overview

2.4.3 Overview of steps to create a CICS Web Service

We are about to create a CICS Web Service from an existing COBOL/CICS program (CUSTINQ) that is invoked by another COBOL program (CALCUSIN).

The CALCUSIN program sends a 3270 BMS map used to request a customer number. If the customer number is valid, CALCUSIN then invokes another COBOL program named CUSTINQ using an EXEC CICS LINK via the CICS COMMAREA.

CUSTINQ receives the customer number, reads a KSDS VSAM data set and sends the data retrieved back to the COMMAREA (name, last name, and address). When CALCUSIN receives control it displays the data on the 3270 screen.

Since the objective is to show how to enable an existing COBOL/CICS application as a Web Service, we chose to use VSAM instead of DB2 to simplify the development of the generated artifacts.

The objective is to create a WSDL and use the Web Services Explorer Tool to invoke the CUSTINQ COBOL program.

We will perform the following tasks:

1. Become familiar with the sample programs and invoke them using the 3270 terminal emulation.
2. Import the CUSTINQ COBOL program into WDz Web Services for the CICS Project.
3. Generate the converter driver, input and output converters, XML schemas, WSDL, and the CICS required file (WSBind).
4. Display the CICS Web Services on the z/OS configuration.
5. Deploy the generated COBOL to CICS Transaction Server for z/OS.
6. Use the Web Services Explorer to test the CICS-based Web Service.
7. Create the Web Services client from the WSDL.

2.4.4 WebSphere Application Server

The WebSphere Application Server will run the Web Service client application to provide a Web page to inquire the information from VSAM through the Web Services.

The WebServiceProject is the name of the Web Service client application generated with Websphere Developer for System z to provide a way for you to see the Web Services working. We execute TestClient.jsp to input the CustNo and retrieve the customer information at the screen. This jsp runs in WebSphere Application Server and the customer information is stored in the VSAM file accessed by COBOL.

2.5 The steps in detail

In the following sections we describe in more detail the steps to get the existing COBOL/CICS program exposed as Web Services. Note that our description is based on our environment; you may have to use other naming conventions in your environment.

2.5.1 Executing the sample application

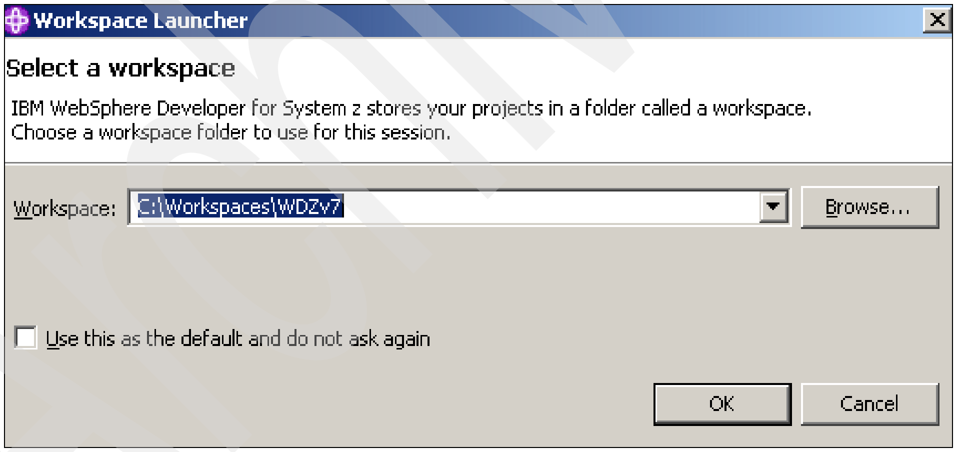
First, it is a good idea to start a 3270 terminal emulation and execute the sample application, which we will then enable as a Web Service.

We will also create an Enterprise Service Tools (EST) project which will be used to hold a logon macro.

Note: Refer to Appendix B, "Additional material" on page 397 for details on obtaining the content of this sample.

Starting WDz and opening a workspace

1. Start Websphere Developer for System z as follows: **Start** → **Programs** → **IBM Software Development** → **IBM Websphere Developer for System z** → **IBM Websphere Developer for System z**, or at a Quick Launch or shortcut.
2. In the Workspace Launcher dialog, specify the name of the workspace you wish to use. In our scenario we use C:\Workspaces\WDZv7 as the workspace. Click **OK.**



IBM WebSphere Developer for System z stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Workspaces\WDZv7

Figure 2-4 Opening a workspace in WDz

Defining a z/OS connection in WDz

Define a new connection to z/OS as follows:

1. Open the z/OS Projects perspective by selecting **Window** → **Open Perspective** → **z/OS Projects**, as shown in Figure 2-5.

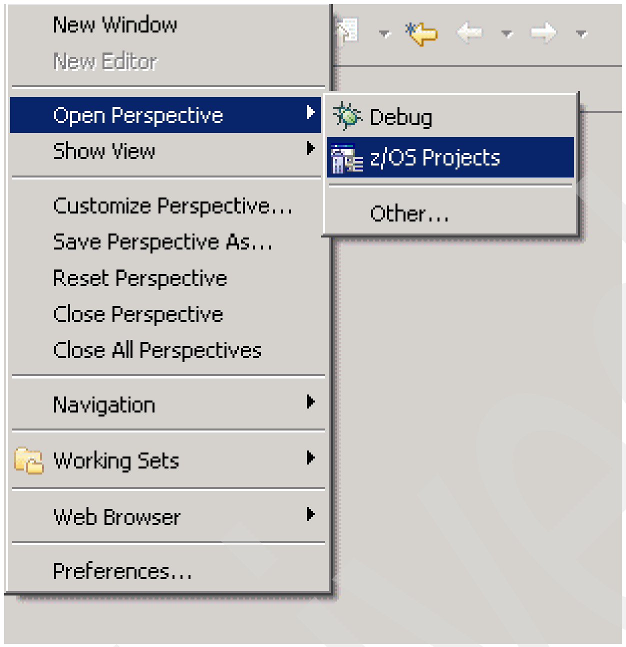


Figure 2-5 Opening the z/OS Projects perspective

2. Create a new z/OS connection. Click the tab **Remote Systems**. In the Remote Systems view, expand the **New Connection** node if needed.
3. From the New Connection tree, right-click **z/OS...** and select **New Connection** to open the pop-up menu, as shown in Figure 2-6.

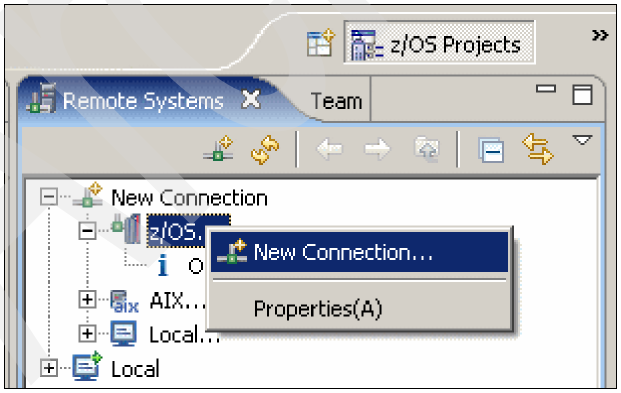
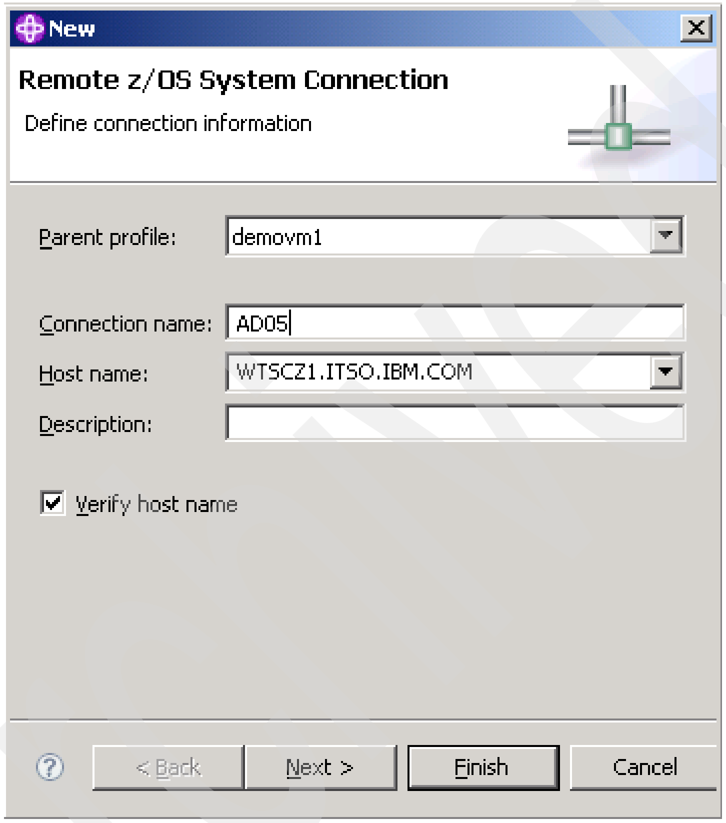


Figure 2-6 z/OS connection - Selection menu

4. Type `demovm1` or another name as the personal profile (the first time that you attempt to create a connection to a remote system you are prompted to create a profile before you can create a new connection).
5. Click **Next**.

6. In the Connection name field, type `AD05` or another name. In the Host name field, type the hostname of the z/OS system you need to work on. In our example we are using wtscz1.itso.ibm.com as hostname. To verify that the hostname or IP address in the Host name field is valid, select the **Verify host name** check box. See Figure 2-7.



Parent profile: demovm1

Connection name: AD05

Host name: WTSCZ1.ITSO.IBM.COM

Description:

[✓] Verify host name

Figure 2-7 z/OS connection - New connection

7. Click **Next** to proceed to the JES subsystem properties page.
8. In the JES job Monitor port field insure that the port field is set to the value defined by your systems programmer. In our scenario we use the value 7715**.** Click **Next**.
9. In the MVS™ Files window ensure that the Daemon Port field is set to the value defined by your systems programmer. In our scenario we use the value 4037**.** Click **Finish.**

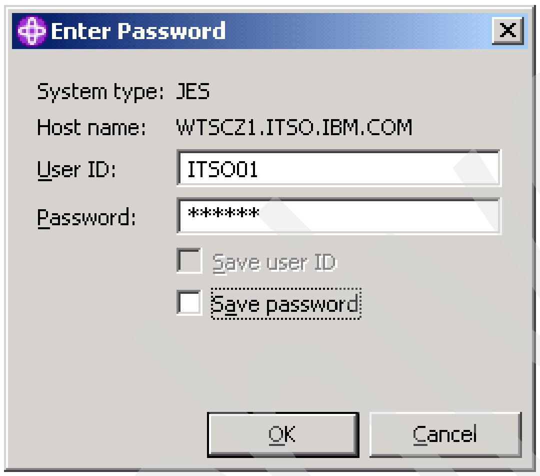
Recording a WDz Macro to connect to the host

You will use the user ID and password assigned to you to log on to the z/OS system where you will deploy the CICS Web Service.

In our scenario we are using userid ITSO01 and password ITSO01.

To connect to the z/OS system, right-click the **AD05** connection you created earlier and select **Connect**.

Type the user ID and password and click **OK**, as shown in Figure 2-8.



System type: JES
Host name: WTSCZ1.ITSO.IBM.COM
User ID: ITSO01
Password: ******

Figure 2-8 z/OS login screen

Since your connection is not secured, you may receive an SSL message. In case you do, just click **YES** to continue.

Emulating a 3270 terminal with WDz

Using the Remote Systems view, in the z/OS perspective, right-click **AD05** and select **Host Connection Emulation Support**, as shown in Figure 2-9.

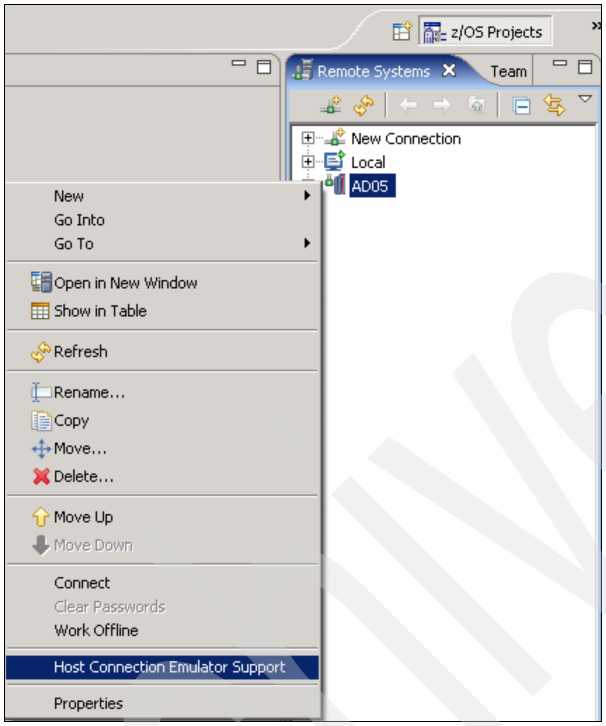
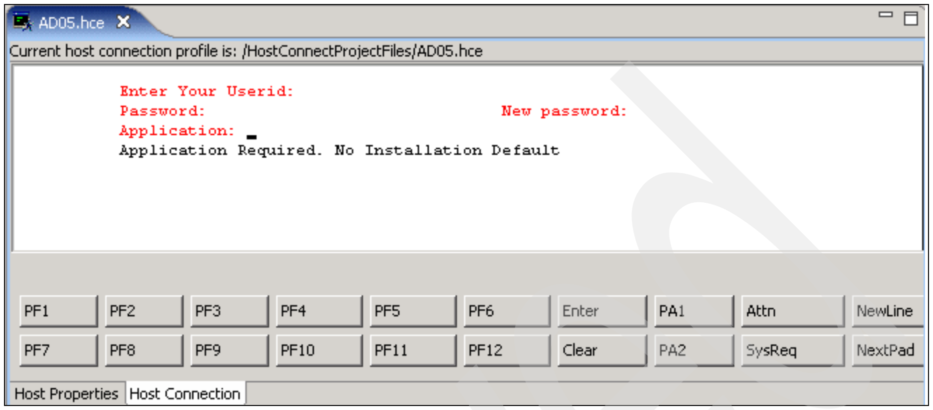


Figure 2-9 Selecting the Host emulator

You should now be presented with a 3270 emulation window, as shown in Figure 2-10. The exact content of the 3270 screen depends on your specific environment.



Current host connection profile is: /HostConnectProjectFiles/AD05.hce

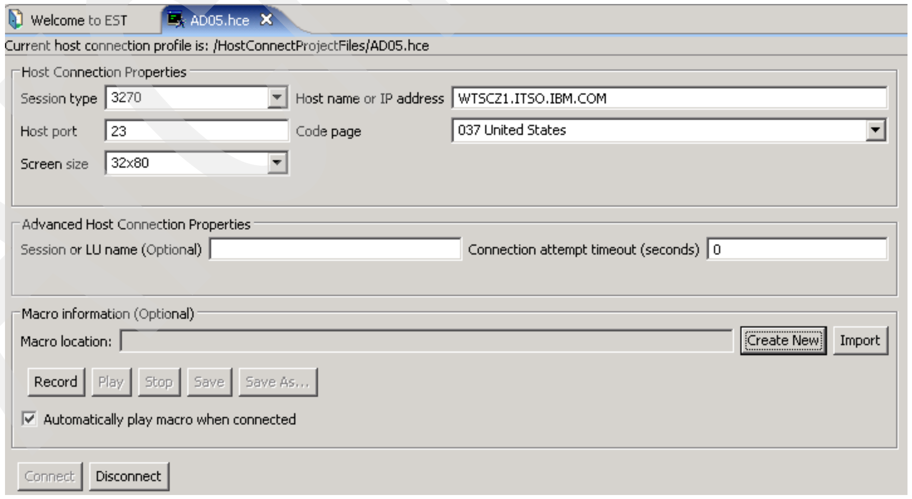
```
Enter Your Userid:
Password:                                    New password:
Application: _
Application Required. No Installation Default
```

Figure 2-10 ITSO System Welcome panel

To better view the screen, double-click the AD05.hce title. Later, to return the screen to its original size, double-click the AD05.hce title again.

Now we explain how to create a macro to automate the login procedure.

1. Click the **Host Properties** tab (at the bottom of the 3270 emulation screen) and click **Create New**.



Current host connection profile is: /HostConnectProjectFiles/AD05.hce

Host Connection Properties

Session type: 3270
Host name or IP address: WTSCZ1.ITSO.IBM.COM
Host port: 23
Code page: 037 United States
Screen size: 32x80

Advanced Host Connection Properties

Session or LU name (Optional):
Connection attempt timeout (seconds): 0

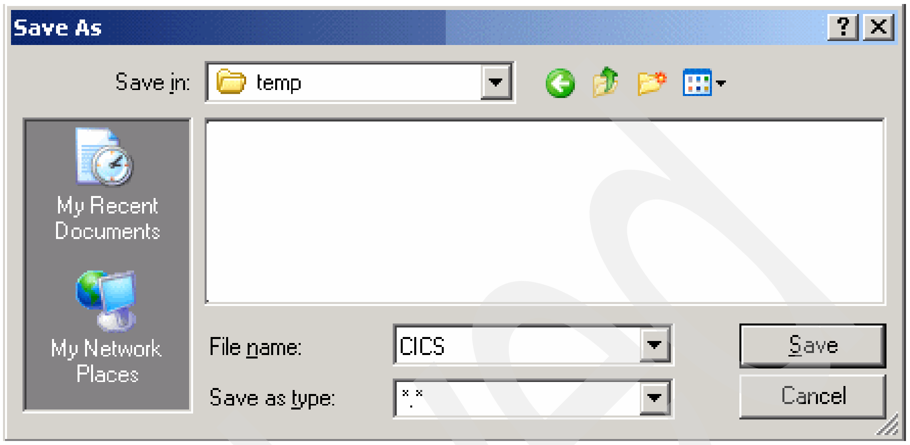
Macro information (Optional)

Macro location:

[✓] Automatically play macro when connected

Figure 2-11 Create a new macro

2. Now open the file browse screen to select a file to save the new macro to. Navigate to the c:\temp\ or other directory of choice and type the file name CICS or another name of your choice and click **Save**.



File name: CICS

Save as type: *.*

Figure 2-12 Save the new macro

3. Now return to the Host Properties screen. Click **Record** and confirm the option **Automatically play macro** while being connected to the host.

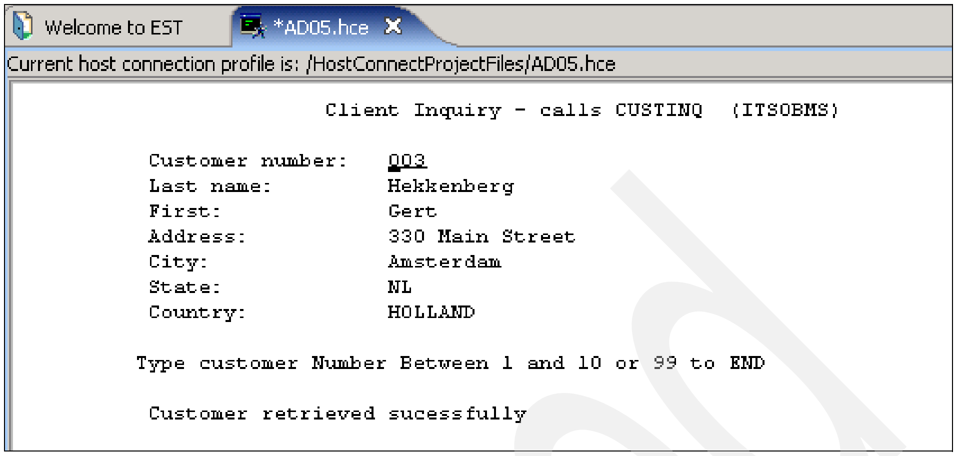
   From now on the sequence of activities in your macro will depend on how your CICS logon is set up. Continue with your CICS login until you reach the message `DFHCE3549 Sign-on is complete (language ENU).`

4. You must now stop the record sequence. Click the tab **Host Properties** (bottom right) again and click **Stop**. After that, click **Save** to save your signon sequence in the file c:/temp/CICS “macro”.
5. Type `CTRL + s` to save the emulation settings.

Now connect and sign on to CICS using the macro you just created.

Running the CICS transaction using WDz host emulation

Using the host emulation session opened earlier, click the tab **Host Connection**. Type the CICS transaction IINQ and press Enter. Type customer number 003 and press Enter again.



Current host connection profile is: /HostConnectProjectFiles/AD05.hce

```
                      Client Inquiry - calls CUSTINQ  (ITSOBMS)

           Customer number:   003
           Last name:         Hekkenberg
           First:             Gert
           Address:           330 Main Street
           City:              Amsterdam
           State:             NL
           Country:           HOLLAND

          Type customer Number Between 1 and 10 or 99 to END

           Customer retrieved sucessfully
```

Figure 2-13 IINQ CICS Transaction results

The CALCUSIN program sends the BMS map and accepts the Customer number as input. If a valid customer is entered, it executes a EXEC CICS LINK to the CUSTINQ program, which then reads a VSAM data set and sends the results back via the COMMAREA.

The CALCUSIN program receives control, retrieves the results from COMMAREA and displays them using the BMS map.

Later we will enable the CUSTINQ program to make it accessible as a CICS Web Service.

To end IINQ transaction type 009 and type Enter.

2.5.2 Creating a CICS Web Services project

Now that we have explained how to work with the CICS sample application using the WDz host emulation features, it is a good time to start the development work of the new Web Service. To perform the tasks efficiently, you need to use the Enterprise Service Tools perspective.

Creating a Web Services project in WDz

Development work in WDz always starts with the creation of a “place holder” of the components you will be creating. We call this a *Project*.

1. Select **Window** → **Open Perspective** → **Other...** → **Enterprise Service Tools**.

   The Welcome to EST page opens.

2. To create a new project, use the sequence: **File** → **New** → **Project** → **Enterprise Service Tools** and select **CICS Web Services Project**, as shown in Figure 2-14.

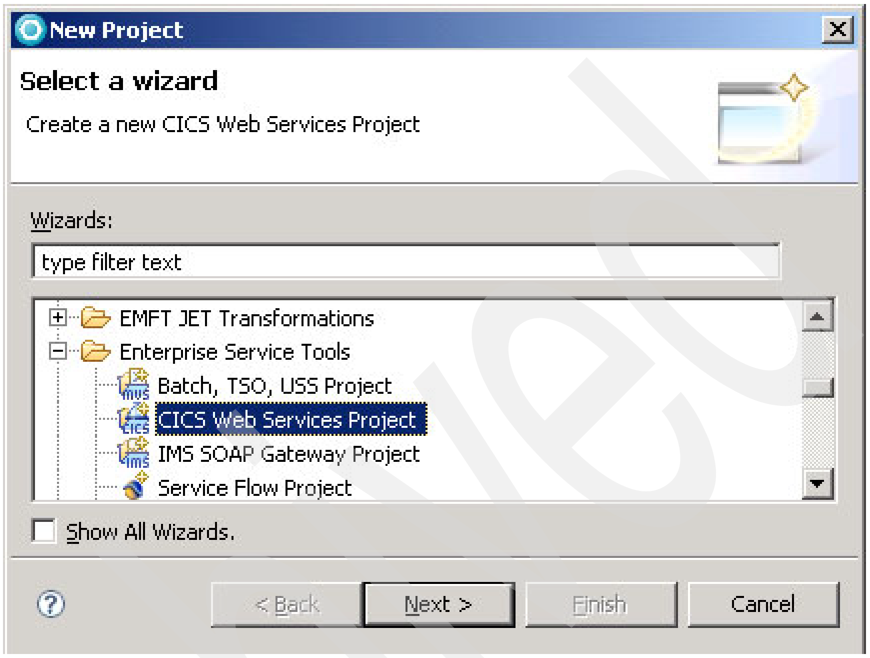


Figure 2-14 EST Project list

3. Click **Next**. On the next screen type the name of the project. We used ITSO_AD05_CICS_Service as the project name. Click **Finish**.

An empty project is created. Now you can import the COBOL source, called CUSTINQ, and its required copybooks needed to create the Web Service.

Importing the copybook and COBOL sources

Once a project is created, you can start importing the required sources of the COBOL program.

1. Click the **EST Project Explorer** tab to switch to the Project Explorer view.

2. Using the Enterprise Service Tools Perspective and the EST Project Explores View, right-click **ITSO_AD05_CICS_Service** and select **Import Source Files**, as shown in Figure 2-15.

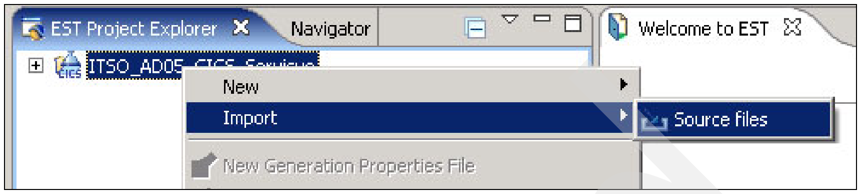


Figure 2-15 EST Project explorer - Import source files

3. Click **File System** and navigate to the directory with the source file CUSTINQ.cbl and its copybooks COMMAREA.cpy and ITSOVSAM.cpy, change filter to *.*, select the files, then click **Open.** The imported source appears as shown in Figure 2-16.

Note: The sources are included in the additional material for this book. Refer to Appendix B, "Additional material" on page 397 for details.

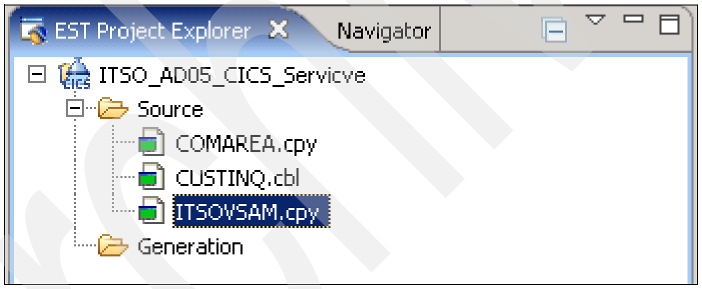


Figure 2-16 EST Project Explorer window: Source

The copybook of the COMMAREA is fairly simple and in reality COMMAREAS will be more complex.

Example 2-1, Example 2-2 on page 32 and Example 2-3 on page 32 show the COMMAREA, the VSAM file record and the CUSTINQ COBOL source, respectively.

Example 2-1 COMMAREA copybook

```
02  CustNo       PIC S9(9) COMP-5.
02  LastName     PIC A(25).
02  FirstName    PIC A(15).
02  Address1     PIC X(20).
```

```
   02  City        PIC A(20).
   02  State       PIC A(5).
   02  Country     PIC X(15).
   02  RetCode     PIC S9.
```

Example 2-2   ITSOVSAM copybook

```
      *****************************************************************
      ***   DSN     = ITSOOC.ITSO.VSAM
      ***   FCT     = ITSOVSAM               DSORG   = VSAM KSDS   ****
      *****************************************************************
       01  ITSOVSAM-RECORD-REC.
           03 CUST-NO           PIC 999.
           03 CUST-LN           PIC X(25).
           03 CUST-FN           PIC X(15).
           03 CUST-ADDR1        PIC X(20).
           03 CUST-CITY         PIC X(20).
           03 CUST-ST           PIC X(5).
       03 CUST-CTRY         PIC X(15).
```

Example 2-3   CUSTINQ program source

```
IDENTIFICATION DIVISION.
       PROGRAM-ID.    CUSTINQ.
       AUTHOR.        Reginaldo Barosa.
       INSTALLATION.  IBM Dallas.
       DATE-WRITTEN.  01/21/07
      *****************************************************************
      *  DESCRIPTION : CICS SUBROUTINE TO CUSTOMER DATA               *
      *  Receive customer #, read VSAM,  move data  to COMMAREA       *
      *  ENVIRONMENT : CICS, COBOL II                                 *
      *  CICS TRANSACTION NAME : NONE                                 *
      *  SUBROUTINE: NONE                                             *
      *  VSAM FILES:  DNET045.WDZV7.POT                               *
      *****************************************************************
      ***   MODIFICATIONS                                             **
      *** WHO   DATE    CHANGE                                        **
      *** --- -------- ----------------------------------------------**
      *** KMV 10/05/06  ORIGINAL                                      **
      *** RB  01/21/07  ADAPTED TO WDZ V7 POT                         **
      *****************************************************************
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       EJECT
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  WS-PROGRAM                PIC X(08)  VALUE 'CUSTINQ'.
       01  WS-LITERAL-WS             PIC X(48)  VALUE
           '        WORKING STORAGE STARTS HERE'.
```

```
 01  WORK-VARIABLES.
     05  WS-RESP                PIC S9(8)    VALUE 0  COMP.
* Below is the VSAM record area
 COPY ITSOVSAM.
 01  WS-LITERAL                  PIC X(48)  VALUE
      '        LINKAGE SECTION STARTS HERE'.
* -------------------------------------------------------------- *
 LINKAGE SECTION.
* -------------------------------------------------------------- *
 01  DFHCOMMAREA.
      COPY COMAREA.
 PROCEDURE DIVISION.
*****************************************************************
 0100-MAIN-ROUTINE.
     IF  CustNo > 10 OR  CustNo < 1 THEN
         MOVE CustNo  TO CUST-NO
         MOVE  SPACES TO DFHCOMMAREA
         MOVE CUST-NO TO CustNo
         MOVE 'Should be > 0 and < 11' TO LastName
         MOVE  -1 to RetCode
         EXEC CICS RETURN
         END-EXEC.
*****************************************************************
* READ VSAM FILE - FCT is ITSOVSAM                               *
*****************************************************************
 0300-ITSOVSAM-READ.
      MOVE CustNo  TO CUST-NO .
      EXEC CICS READ
               FILE   ('ITSOVSAM')
               INTO   ( ITSOVSAM-RECORD-REC )
               LENGTH (LENGTH OF ITSOVSAM-RECORD-REC )
               RIDFLD ( CUST-NO )
               EQUAL
               RESP   (WS-RESP)
     END-EXEC.
 0400-CHECK-IFOK.
     IF  WS-RESP NOT = DFHRESP(NORMAL)
          MOVE SPACES                    TO DFHCOMMAREA
          MOVE 'INVALID READ ITSOVSAM ' TO LastName
         EXEC CICS RETURN
         END-EXEC.
     IF WS-RESP = DFHRESP(NOTFND)
         MOVE SPACES                    TO DFHCOMMAREA
         MOVE 'ITSOVSAM  NOT FOUND ' TO LastName
         EXEC CICS RETURN
         END-EXEC.
 0500-DATA-RETURNED-OK.
     MOVE  CUST-LN    TO LastName.
     MOVE  CUST-FN    TO FirstName.
     MOVE  CUST-ADDR1 TO Address1 .
     MOVE  CUST-CITY  TO City .
     MOVE  CUST-ST    TO State .
     MOVE  CUST-CTRY  TO Country .
     MOVE ZEROES TO RetCode .
     EXEC CICS RETURN
     END-EXEC.
```

2.5.3 Generating the CICS Web Services resources

In the next steps we use the WDz wizards that will generate the COBOL driver and converter programs, the WSDL file, the CICS WSBind file, and the XML schemes.

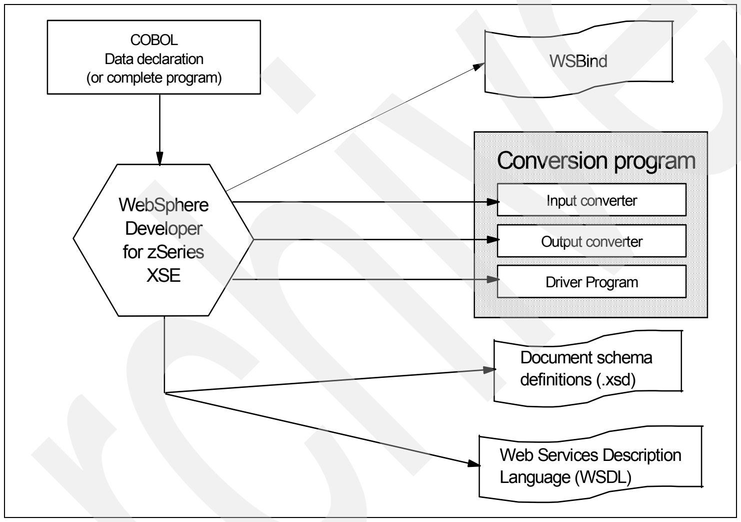
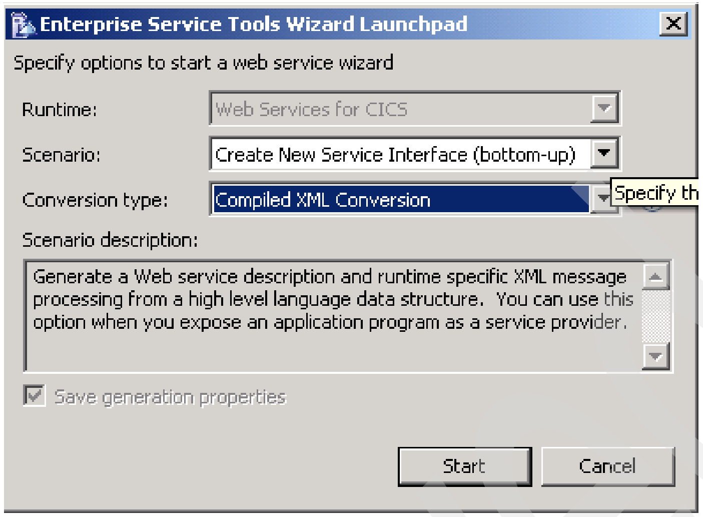


Figure 2-17 Creating a CICS Web Service - components

1. Under the ITSO_AD05_CICS_Service project, right-click the **CUSTINQ.cbl** program and select **Generate Web Services for CICS resources**.

   This will launch the Web Service Runtime and Scenario Selection dialog allowing you to specify the necessary options to start the Web Service wizard.

   We start from the COBOL CUSTINQ program to create the Web Services, using bottom-up operations. Also, since we want to select a single field as the input message (customer number) we need to specify a Conversation type of Compiled XML Conversation. See Figure 2-18.


Enterprise Service Tools Wizard Launchpad

Specify options to start a web service wizard

Runtime: Web Services for CICS

Scenario: Create New Service Interface (bottom-up)

Conversion type: Compiled XML Conversion

Scenario description:

Generate a Web service description and runtime specific XML message processing from a high level language data structure. You can use this option when you expose an application program as a service provider.

Save generation properties

Figure 2-18 Web Service Runtime and Scenario Selection

2. Accept all defaults and press **Start** to begin the Web Service wizard.

Informational: Compiled versus Interpretive XML conversion.

There are two conversion type choices available for this scenario and runtime combination. Either Compiled XML Conversion or Interpretive XML Conversion may be selected from the Web Service Runtime and Scenario Selection dialog.

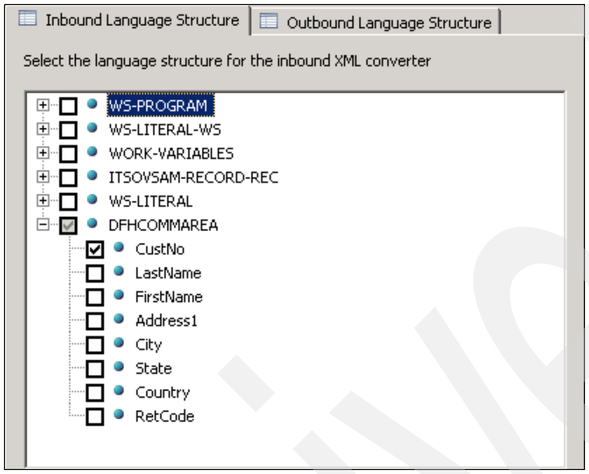
Compiled XML Conversion

The CICS runtime interacts with a driver program that either includes or invokes conversion logic to provide conversion of XML to and from language structures. The compiled XML conversion type provides more extensive support for language structure data types and constructors than the interpretive XML conversion type. However, compilation of XML Converters is required and there are more artifacts to manage.

Interpretive XML Conversion

XML conversion is accomplished using generated metadata and an XML conversion component built into the runtime. While the Interpretive XML Conversion type has limited support for language structure data types and constructors, it requires fewer artifacts and does not involve compilation of XML converters

3. On the Web Services for CICS - Create New Service Interface (bottom-up) wizard page, make sure the **Inbound Language Structure** tab is selected. Expand **DFHCOMMAREA** by clicking the + sign preceeding it and select **CustNo** as the only element that will comprise the data structure for the inbound XML converter. See Figure 2-19.



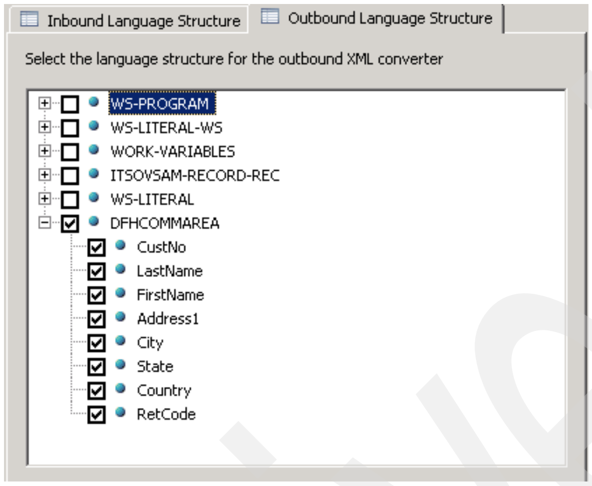
Select the language structure for the inbound XML converter

- [ ] WS-PROGRAM
- [ ] WS-LITERAL-WS
- [ ] WORK-VARIABLES
- [ ] ITSOVSAM-RECORD-REC
- [ ] WS-LITERAL
- [✓] DFHCOMMAREA
  - [✓] CustNo
  - [ ] LastName
  - [ ] FirstName
  - [ ] Address1
  - [ ] City
  - [ ] State
  - [ ] Country
  - [ ] RetCode

Figure 2-19 Web Services for CICS Wizard Windows

Note: Only the Conversion type of Compiled XML Conversion allows selecting a specific field as in the example above.

4. Select the **Outbound Language Structure** tab. Click the check box next to DFHCOMMAREA to select all the data items as the data structure for the outbound XML converter. See Figure 2-20.

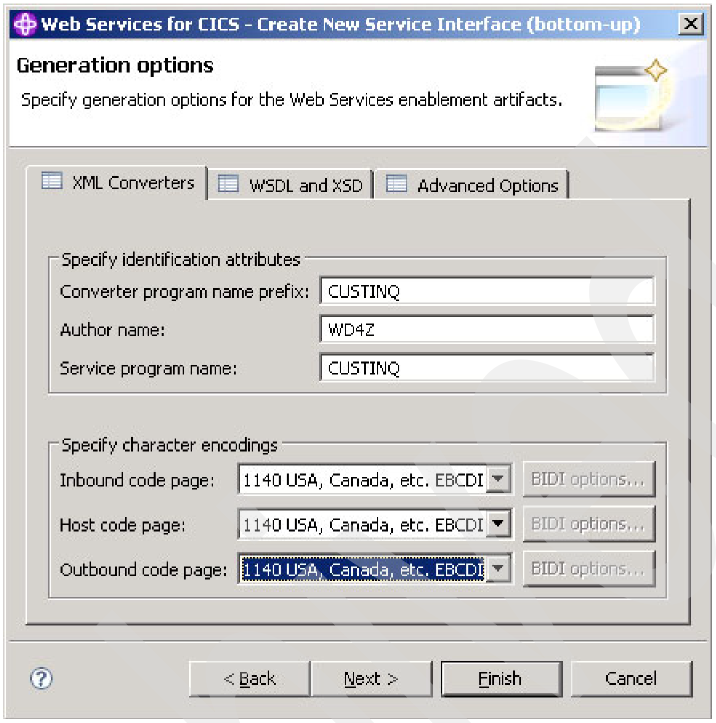


Select the language structure for the outbound XML converter

- [ ] WS-PROGRAM
- [ ] WS-LITERAL-WS
- [ ] WORK-VARIABLES
- [ ] ITSOVSAM-RECORD-REC
- [ ] WS-LITERAL
- [✓] DFHCOMMAREA
  - [✓] CustNo
  - [✓] LastName
  - [✓] FirstName
  - [✓] Address1
  - [✓] City
  - [✓] State
  - [✓] Country
  - [✓] RetCode

Figure 2-20 Web Services for CICS Wizard Windows

5. Click **Next**. You will see the panel shown in Figure 2-21.



Web Services for CICS - Create New Service Interface (bottom-up)

Generation options

Specify generation options for the Web Services enablement artifacts.

Specify identification attributes

Converter program name prefix: CUSTINQ

Author name: WD4Z

Service program name: CUSTINQ

Specify character encodings

Inbound code page: 1140 USA, Canada, etc. EBCDI

Host code page: 1140 USA, Canada, etc. EBCDI

Outbound code page: 1140 USA, Canada, etc. EBCDI

Figure 2-21 Web Services for CICS Wizard Windows

Suffixes will be added to the program name prefix to form the program IDs of the driver and converter programs. For example, with a program name prefix of CUSTINQ, the program ID of the driver program is CUSTINQD.

The CICS Web Services runtime looks into the WSBind file for the name of the conversion program to be invoked. WDz stores the program ID of the driver program in the WSBind file; therefore, the conversion program (driver/converters) must be built into a load module with the name CUSTINQD.

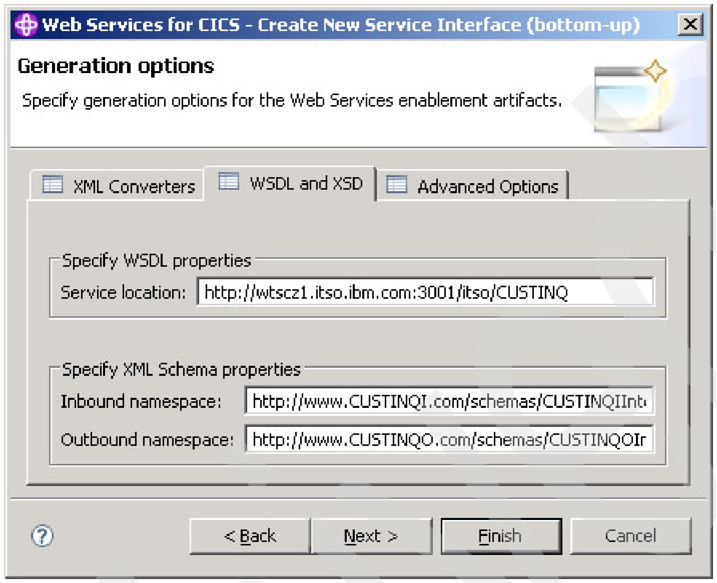
Select the **WSDL and XSD** tab and specify the Service location. In our example, we use:

```
http://wtscz1.itso.ibm.com:3001/itso/CUSTINQ
```

This will be the SOAP address that is used to reach the target CICS system where the conversion program (CUSTINQ) is deployed. The transport mechanism used is HTTP.

In our example, the host where CICS is running is wtscz1.itso.ibm.com, the port number used for CICS Web Services is 3001, and the local URI portion is

/itso/CUSTINQ, which will be used to create the URIMAP used by CICS Web Services facility to matching URI requests.


Web Services for CICS - Create New Service Interface (bottom-up)

Generation options

Specify generation options for the Web Services enablement artifacts.

Specify WSDL properties

Service location: http://wtsczl.itso.ibm.com:3001/itso/CUSTINQ

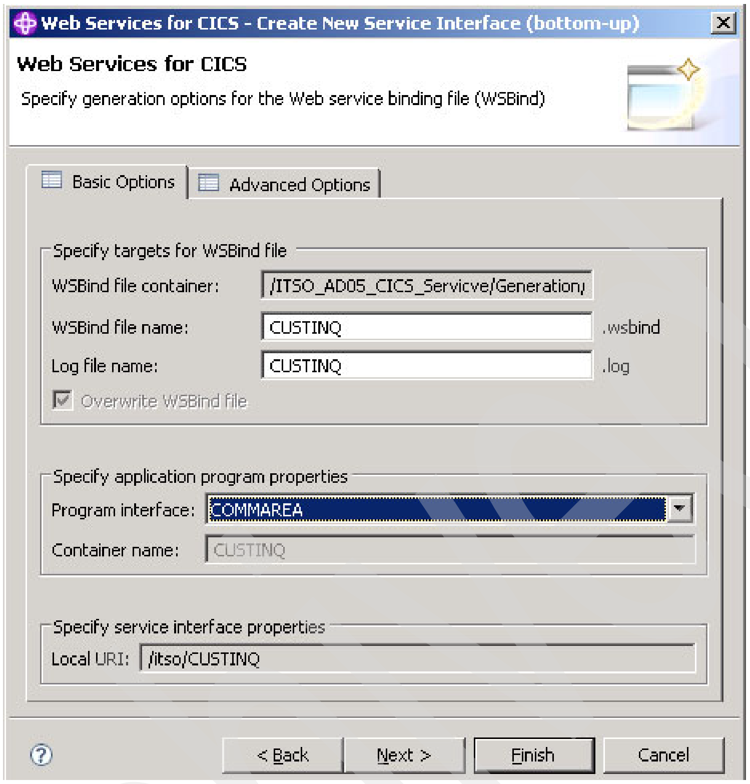
Specify XML Schema properties

Inbound namespace: http://www.CUSTINQI.com/schemas/CUSTINQIInt

Outbound namespace: http://www.CUSTINQO.com/schemas/CUSTINQOIr

Figure 2-22 Web Service for CICS Wizard window

6. Click **Next**. You will see the Web Services for CICS window. Using the Basic Options tab, accept the default WSBind file names of CUSTINQ and COMMAREA as the program interface and click **Next**.


Web Services for CICS - Create New Service Interface (bottom-up)

Web Services for CICS

Specify generation options for the Web service binding file (WSBind)

Specify targets for WSBind file

WSBind file container: /ITSO_AD05_CICS_Servicve/Generation

WSBind file name: CUSTINQ .wsbind

Log file name: CUSTINQ .log

Overwrite WSBind file ✓

Specify application program properties

Program interface: COMMAREA

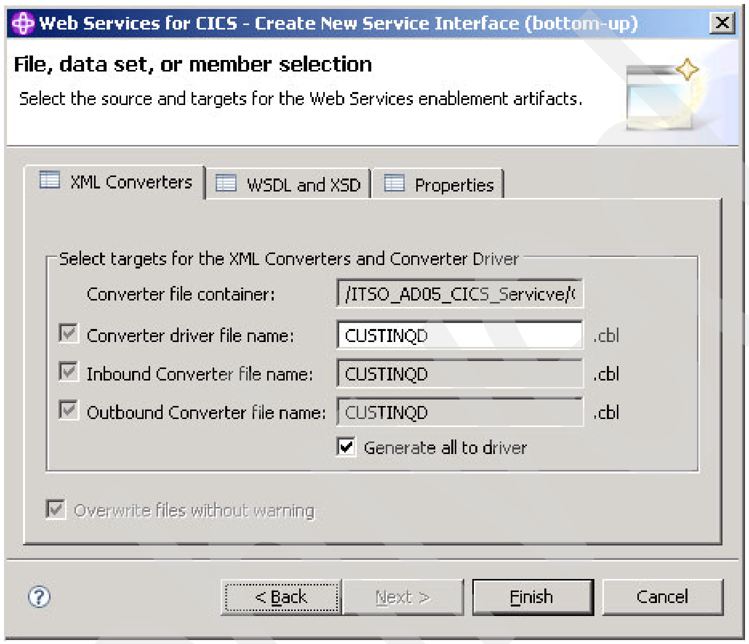
Container name: CUSTINQ

Specify service interface properties

Local URI: /itso/CUSTINQ

Figure 2-23 Web Services for CICS wizard window

7. On the *File, data set, or member selection* window, as shown in Figure 2-24, select the **XML Converters** tab and make sure that CUSTINQD is specified as the Converter driver file name. Also make sure the check box **Generate all to driver** is checked. This generates all the components to the same program source file.


Web Services for CICS - Create New Service Interface (bottom-up)
XML Converters
WSDL and XSD
Properties
< Back
Next >
Finish
Cancel

File, data set, or member selection

Select the source and targets for the Web Services enablement artifacts.

Select targets for the XML Converters and Converter Driver

Converter file container: /ITSO_AD05_CICS_Servicve/(

[✓] Converter driver file name: CUSTINQD .cbl

[✓] Inbound Converter file name: CUSTINQD .cbl

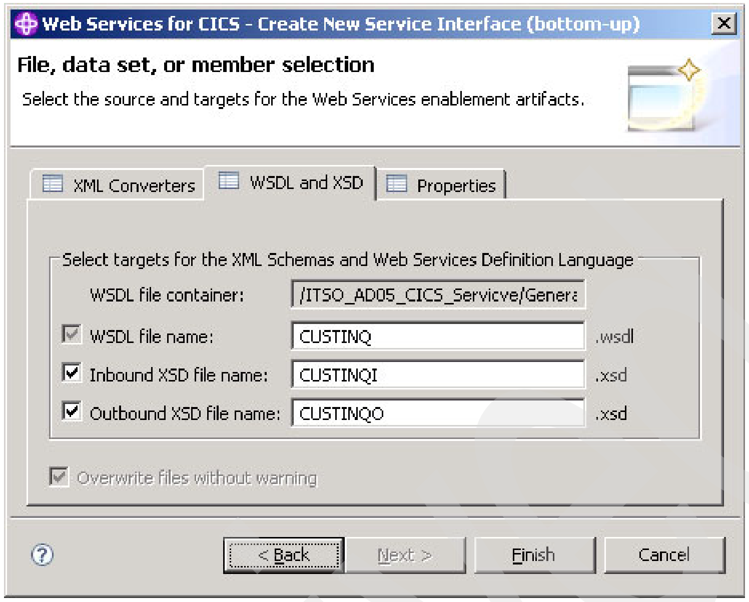
[✓] Outbound Converter file name: CUSTINQD .cbl

[✓] Generate all to driver

[✓] Overwrite files without warning

Figure 2-24 Web Services for CICS wizard window

8. Select the **WSDL and XSD** tab. You will see the file names that were generated for the WSDL, Inbound, and Outbound XSD files.



Web Services for CICS - Create New Service Interface (bottom-up)

File, data set, or member selection

Select the source and targets for the Web Services enablement artifacts.

Select targets for the XML Schemas and Web Services Definition Language

WSDL file container: /ITSO_AD05_CICS_Servicve/Genera

[✓] WSDL file name: CUSTINQ .wsdl

[✓] Inbound XSD file name: CUSTINQI .xsd

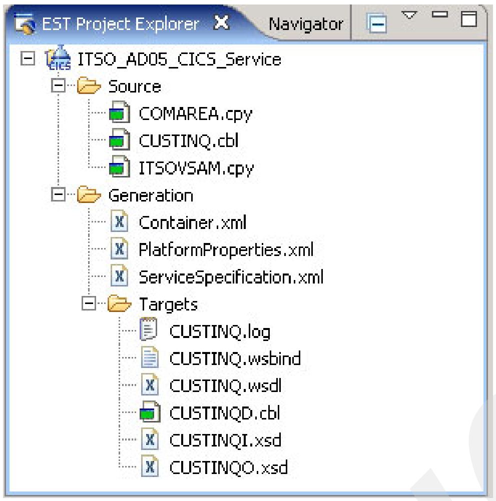
[✓] Outbound XSD file name: CUSTINQO .xsd

[✓] Overwrite files without warning

Figure 2-25 Web Services for CICS wizard window

9. Click **Finish** to complete the generation of the conversion program (converter driver, input and output converters), WSBind, XSD, and WSDL files.

   After the Web Services enablement wizard completes its generation, you should see several additional artifacts in the ITSO_AD05_CICS_Service project under the folder Generation, as shown in Figure 2-26.



```
ITSO_AD05_CICS_Service
  Source
    COMAREA.cpy
    CUSTINQ.cbl
    ITSOVSAM.cpy
  Generation
    Container.xml
    PlatformProperties.xml
    ServiceSpecification.xml
    Targets
      CUSTINQ.log
      CUSTINQ.wsbind
      CUSTINQ.wsdl
      CUSTINQD.cbl
      CUSTINQI.xsd
      CUSTINQO.xsd
```

Figure 2-26 EST Project Explorer window

The components are:

- The *.xml files are properties that will be used in the next wizard generation run.
- The CUSTINQ.log file is the generation log.
- The CUSTINQ.wsbind file tells CICS Web Services how conversion is to be performed and what the URI map looks like.
- The CUSTINQ.wsdl file describes the CICS-based Web Services.
- The CUSTINQD.cbl file is the conversion program that is made up of the converter driver and the input and output converters.
- The CUSTINQI.xsd and CUSTINQO.xsd files are XML Schema Definition files that describe the input and output XML documents, respectively.

Browsing the generated files

Double-click **CUSTINQD.cbl**. You will see the COBOL program source.

Note: CUSTINQD.cbl consists of multiple programs. The main program calls the various subroutines. All, however, will be deployed as a simple load module.

Because we specified the compiled conversion type, the XML parsing is done by the COBOL converter program logic contained in the driver program, instead of by CICS.

The converter logic is part of the driver program due to the “generate all to driver option” used at generation time.

Look for the XML PARSE string on the source program (use Crtl+f). You will see an example of the parsing being done by the converter.

The converter driver for CICS Web Services is different from the driver for SOAP for CICS version 2.2, specifically regarding its interaction with the original existing application program. The SOAP for CICS driver contains an EXEC CICS LINK to the original existing application program. The CICS Web Services driver does not contain a call to that program. That is because CICS Web Services is responsible for calling the application program.

2.5.4 Configuring CICS Web Services on z/OS

This section shows the CICS configuration for Web services that we used in our environment. We present an example of how a PIPELINE resource destination and a TCPIPService definition must be configured.

Browsing the HFS directories

To expose a CICS program as a Web Service over HTTP, TCPIPService and PIPELINE resources are required. The PIPELINE resource definition points to Hierarchical File System (HFS) directories as well as a pipeline configuration file.

The setup requires configuring files and directories in the HFS as described below.

The *shelf directory* (`/u/itso01/itso/shelf`) is used by CICS to store the Web Service Binding files that are associated with Web Services resources. Each Web Services resource is associated with a PIPELINE. The shelf directory is managed by CICS together with the PIPELINE resource. You should not modify the content of this directory. Several PIPELINEs can use the same shelf directory because CICS ensures a unique directory structure beneath the shelf directory for each PIPELINE. Since CICS writes to this directory, CICS requires appropriate security permissions.

The *pickup directory* (`/u/itso01/itso/wspickup/provider`) contains the Web Service Binding files that are to be associated with the PIPELINE. When a PIPELINE is installed, or in a response to a PERFORM PIPELINE SCAN command, CICS searches this directory for files ending in .wsbind. For each WSBind file, CICS dynamically creates a WEBSERVICE and URIMAP resource, associates them with the PIPELINE, and installs them ready for use.

You could use the WDz USS shell to create those HFS directories.

To do this, open the z/OS Project perspective by selecting **Windows** → **Open Perspective** → **Other...** and then selecting **z/OS project**.

Expand **AD05**, right-click **USS Files** and select **Launch Shell.** See Figure 2-27**.**

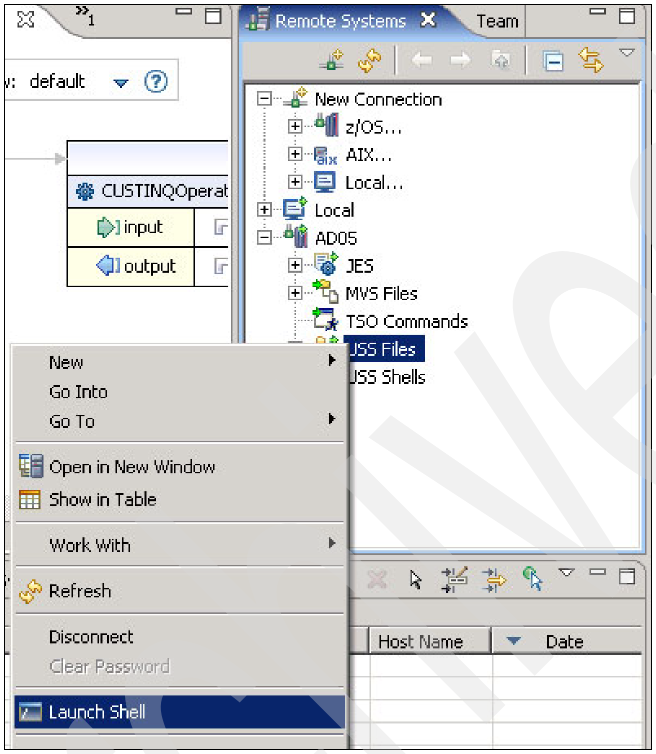
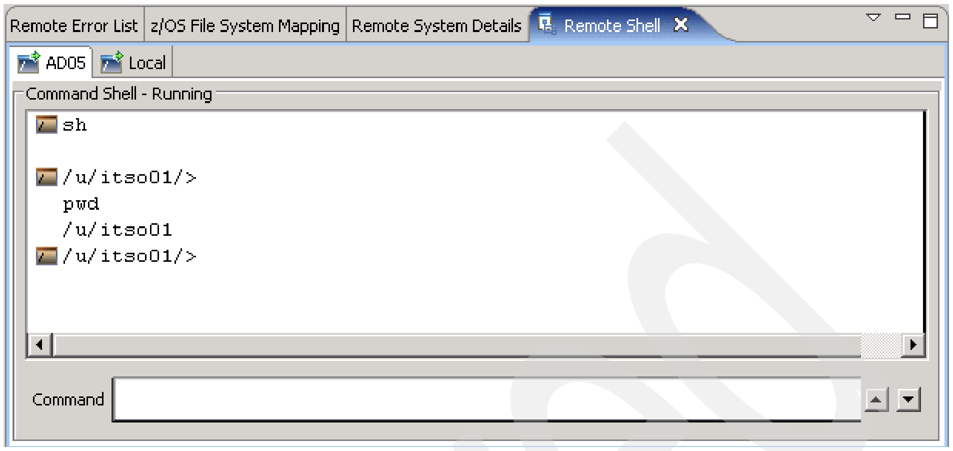


Figure 2-27 Remote Systems view - launching a USS shell

Figure 2-28 shows a USS Shell window.



```
sh

/u/itso01/>
pwd
/u/itso01
/u/itso01/>
```

Figure 2-28 USS Shell window

Browsing a PIPELINE resource definition

A PIPELINE is a CICS resource that is used when a CICS application serves as a Web Service provider or requester. It provides information about the message handler programs that act on a service request and on the response. Typically, a single PIPELINE definition can be used by many applications.

A PIPELINE resource definition contains a reference to a shelf directory, a pickup directory and a pipeline configuration file. The configuration file is in XML format and details the message handlers that will act on Web Service requests and responses as they pass through the pipeline. We will use the CICS-supplied PIPELINE configuration file for a SOAP V1.1 handler.

You can check the TCPIPService definition. Using the Remote System view in the z/OS perspective, right-click **AD05** and select **Host Connection Emulator Support**, as shown in Figure 2-29.

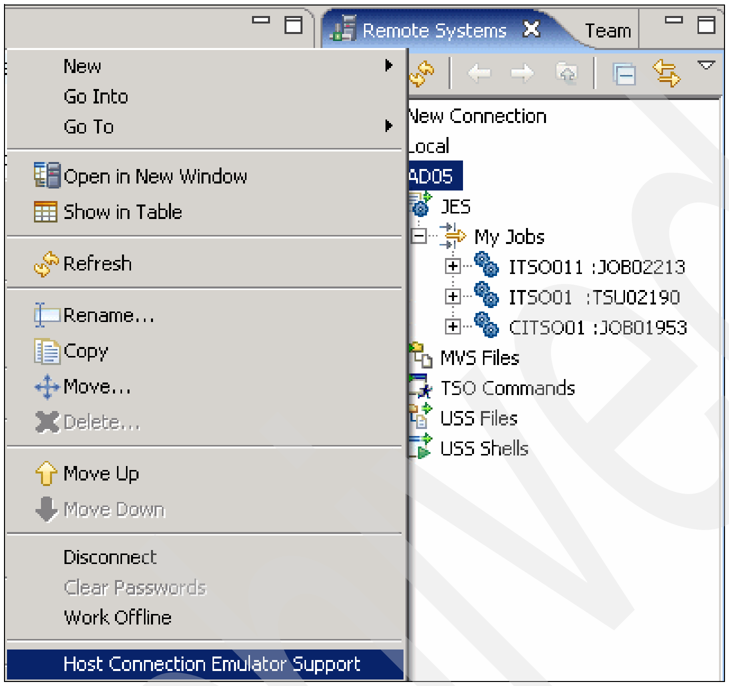
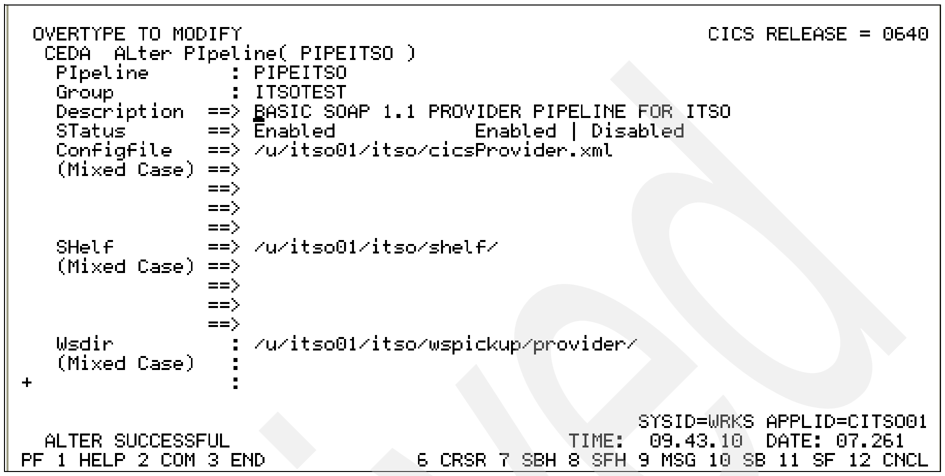


Figure 2-29 Host Connection Emulator Support

We assume that the c:/temp/CICS macro is active, as set up earlier in this scenario.

In the 3270 emulator panel, type `CEDA AL PI` and press Enter. After that specify PIPELINE as `PIPEITSO`, and GROUP as `ITSOTEST` and press Enter again. See Figure 2-30.



```
 OVERTYPE TO MODIFY                                          CICS RELEASE = 0640
  CEDA  ALter PIpeline( PIPEITSO )
   PIpeline       : PIPEITSO
   Group          : ITSOTEST
   Description  ==> BASIC SOAP 1.1 PROVIDER PIPELINE FOR ITSO
   STatus       ==> Enabled           Enabled | Disabled
   Configfile   ==> /u/itso01/itso/cicsProvider.xml
   (Mixed Case) ==>
                ==>
                ==>
                ==>
   SHelf        ==> /u/itso01/itso/shelf/
   (Mixed Case) ==>
                ==>
                ==>
                ==>
   Wsdir          : /u/itso01/itso/wspickup/provider/
   (Mixed Case)   :
+                 :

                                                        SYSID=WRKS APPLID=CITSO01
  ALTER SUCCESSFUL                                TIME:  09.43.10  DATE: 07.261
PF 1 HELP 2 COM 3 END           6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 2-30 Browsing PIPELINE

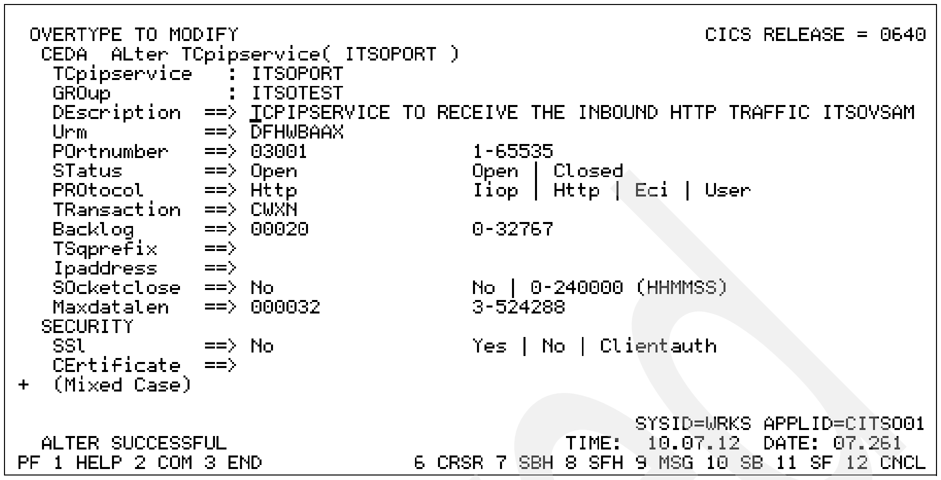
Note that the HFS directories are case-sensitive.

Browsing a TCPIPService resource definition

The TCIPIPService resource tells CICS to listen on a specified port to inbound service requests over HTTP.

As the client connects to your Web Services over an HTTP transport, CICS must provide a TCPIPService to receive the inbound HTTP traffic. Usually the CICS systems programmer defines the PIPELINE and the entry named TCPIPService, as shown below.

You can check the TCPIPService definition. Using the 3270 emulator panel used above, press F3 to clear the screen, type `CEDA AL TC`, and press Enter. After that, specify TCPIPService as ITSOPORT and GROUP as ITSOTEST and press Enter again.



```
 OVERTYPE TO MODIFY                                              CICS RELEASE = 0640
  CEDA  ALter TCpipservice( ITSOPORT )
   TCpipservice    : ITSOPORT
   GROup           : ITSOTEST
   DEscription  ==> TCPIPSERVICE TO RECEIVE THE INBOUND HTTP TRAFFIC ITSOVSAM
   Urm          ==> DFHWBAAX
   POrtnumber   ==> 03001              1-65535
   STatus       ==> Open               Open | Closed
   PROtocol     ==> Http               Iiop | Http | Eci | User
   TRansaction  ==> CWXN
   Backlog      ==> 00020              0-32767
   TSqprefix    ==>
   Ipaddress    ==>
   SOcketclose  ==> No                 No | 0-240000 (HHMMSS)
   Maxdatalen   ==> 000032             3-524288
  SECURITY
   SSl          ==> No                 Yes | No | Clientauth
   CErtificate  ==>
+  (Mixed Case)

                                                          SYSID=WRKS APPLID=CITSO01
  ALTER SUCCESSFUL                                  TIME:  10.07.12  DATE: 07.261
PF 1 HELP 2 COM 3 END             6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

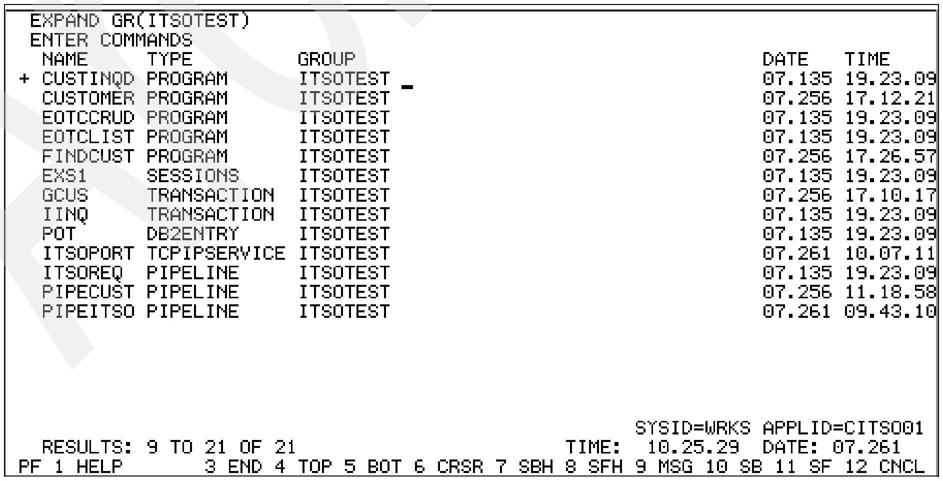
Figure 2-31 Browsing TCPIPService

Note that there is only one TCPIPService opened for each port. Since we will use the port 3001, our TCPIPService will be ITSOPORT.

Installing a CICS GROUP

As with any other resources defined, the PIPELINE and the TCPIPService must be installed in order to work.

In our example, both are installed on group ITSOTEST. To confirm this information, use the 3270 emulator panel. Type `CEDA EXPAND GR(ITSOTEST)` and press Enter.



```
 EXPAND GR(ITSOTEST)
 ENTER COMMANDS
  NAME      TYPE         GROUP                                       DATE   TIME
+ CUSTINQD PROGRAM      ITSOTEST _                                   07.135 19.23.09
  CUSTOMER PROGRAM      ITSOTEST                                     07.256 17.12.21
  EOTCCRUD PROGRAM      ITSOTEST                                     07.135 19.23.09
  EOTCLIST PROGRAM      ITSOTEST                                     07.135 19.23.09
  FINDCUST PROGRAM      ITSOTEST                                     07.256 17.26.57
  EXS1     SESSIONS     ITSOTEST                                     07.135 19.23.09
  GCUS     TRANSACTION  ITSOTEST                                     07.256 17.10.17
  IINQ     TRANSACTION  ITSOTEST                                     07.135 19.23.09
  POT      DB2ENTRY     ITSOTEST                                     07.135 19.23.09
  ITSOPORT TCPIPSERVICE ITSOTEST                                     07.261 10.07.11
  ITSOREQ  PIPELINE     ITSOTEST                                     07.135 19.23.09
  PIPECUST PIPELINE     ITSOTEST                                     07.256 11.18.58
  PIPEITSO PIPELINE     ITSOTEST                                     07.261 09.43.10




                                                          SYSID=WRKS APPLID=CITSO01
  RESULTS: 9 TO 21 OF 21                            TIME:  10.25.29  DATE: 07.261
PF 1 HELP      3 END 4 TOP 5 BOT 6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 2-32 Browsing a GROUP

2.5.5 Deploying assets to CICS Transaction Server for z/OS

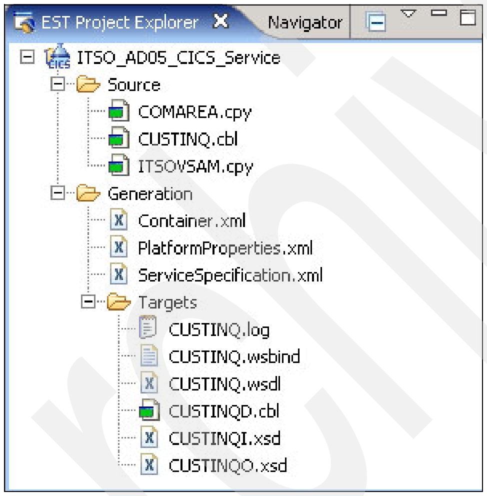
In the following sections we cover the steps necessary to deploy a CICS Web Service.

We will connect to z/OS, generate the JCL to create the load module of the CUSTINQD program, and illustrate other tasks required to deploy the service.

The only deployment performed in this section is that of the CUSTINQD load module. This is an essential step—if the CICS runtime cannot find this program, the service request fails.

Moving the Web Services generated assets to z/OS

After performing the development tasks, you should have the assets shown in Figure 2-33.

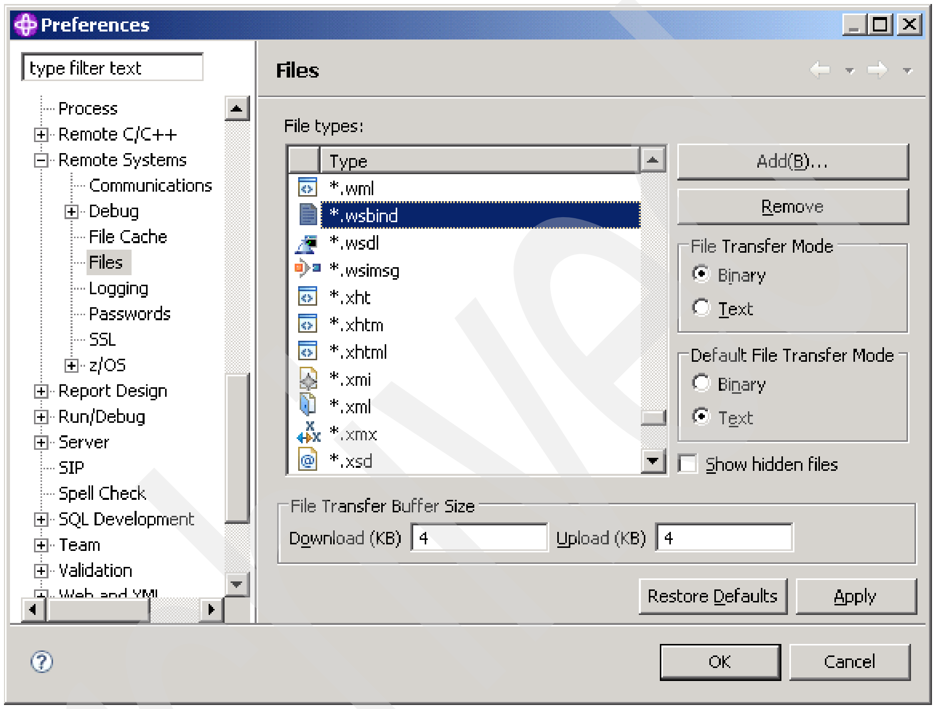


```
ITSO_AD05_CICS_Service
  Source
    COMAREA.cpy
    CUSTINQ.cbl
    ITSOVSAM.cpy
  Generation
    Container.xml
    PlatformProperties.xml
    ServiceSpecification.xml
    Targets
      CUSTINQ.log
      CUSTINQ.wsbind
      CUSTINQ.wsdl
      CUSTINQD.cbl
      CUSTINQI.xsd
      CUSTINQO.xsd
```

Figure 2-33 CICS Web Services assets generated

The assets CUSTINQ.wsbind and CUSTINQD.cbl should both be copied to the z/OS system.

Because the WSBind file is an EBCDIC text file, it needs to be transmitted in binary to the target z/OS system because of some special control symbols. Make sure this type of file is transmitted as binary using the Preferences dialog in **Windows** → **Preferences** → **Remote systems** → **Files.** Figure 2-34 shows how to do this.



Files

File types:

Type
*.wml
*.wsbind
*.wsdl
*.wsimsg
*.xht
*.xhtm
*.xhtml
*.xmi
*.xml
*.xmx
*.xsd

File Transfer Mode: [✓ Binary] Text

Default File Transfer Mode: Binary [✓ Text]

Show hidden files

File Transfer Buffer Size: Download (KB) 4 Upload (KB) 4

Figure 2-34 Setting the file transfer mode in WDz

You can use the Remote Systems Explorer facilities of WDz to copy this file to an HFS directory on the target z/OS system. In the EST perspective, select file CUSTINQD.wsbind, right-click and select **Copy**. Switch to the z/OS perspective. In the Remote Systems Explorer, under the connection you created earlier, expand **USS Files** and navigate to u/itso01/itso/wspickup/provider. Right-click this directory and select **Paste**.

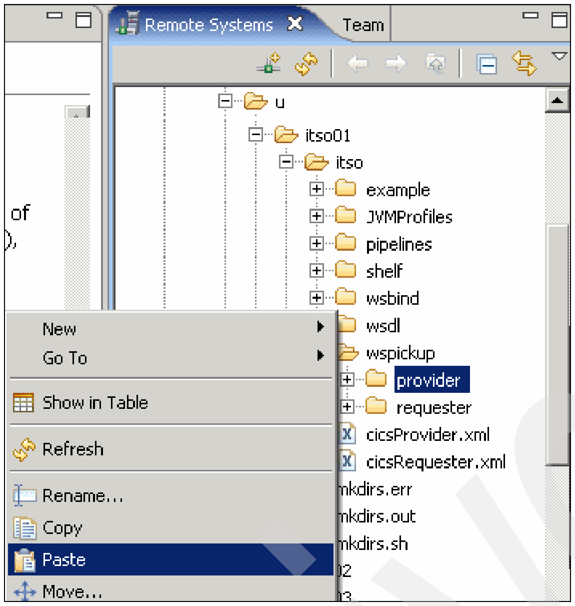


Figure 2-35 Copy the WSBind file

The CUNSTINQ.cbl file is an ASCII text file. Make sure this type of file be transmitted as text using the preferences dialog in **Windows** → **Preferences** → **Remote systems** → **Files.**

The COBOL program source must be copied to the COBOL source data set you will be using on z/OS. In our case this is ITSO01.ITSO.COBOL. Using the Enterprise Service Tools perspective and the EST Project Explorer, right-click **CUNSTINQ.cbl** and select **Copy**. Switch to the z/OS perspective. In the Remote Systems Explorer, under the connection you created earlier, expand **My Datasets** and navigate to your COBOL source data set, which in our case is ITSO01.ITSO.COBOL. Right-click this data set and select **Paste(B)**.

Note: Due to a known problem, the previous step may not allow you to select **Paste(B)**. If that occurs, you should retry starting with the **Copy**.

If the Duplicate Name Collision pop-up appears, simply click **Overwrite** and click **OK.**

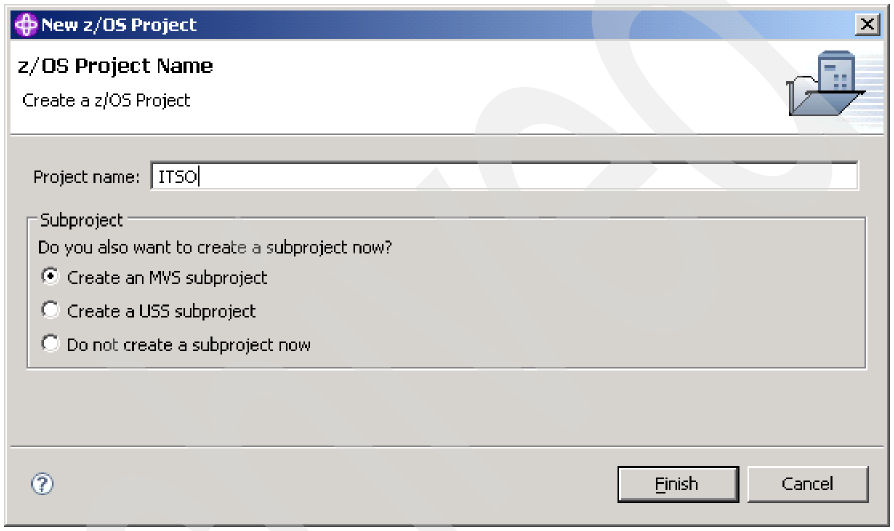
Creating a new z/OS project

As we have already seen, it is much more convenient to create a z/OS Project and then move the assets to the subproject that you are working on. This simplifies the task because you now have all the assets that are being used, grouped in the same location.

To create a new z/OS Project, do the following:

1. Using the z/OS Projects view (on the left), right-click the blank area and select **New → z/OS Project...**

   Type a name in the Project Name field, select **Create an MVS Subproject** and click **Finish**. See Figure 2-36.



New z/OS Project

z/OS Project Name

Create a z/OS Project

Project name: ITSO

Subproject

Do you also want to create a subproject now?

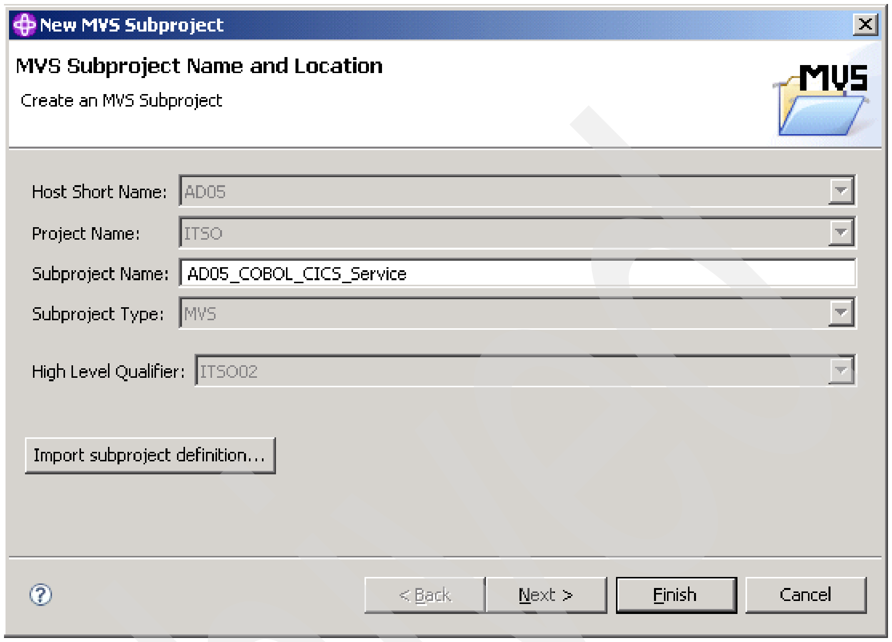
[✓ Create an MVS subproject]

Create a USS subproject

Do not create a subproject now

Figure 2-36 Creating a new z/OS Project

2. Type a name in the Subproject Name field and click **Next**, as shown in Figure 2-37.


New MVS Subproject

MVS Subproject Name and Location

Create an MVS Subproject

Host Short Name: AD05

Project Name: ITSO

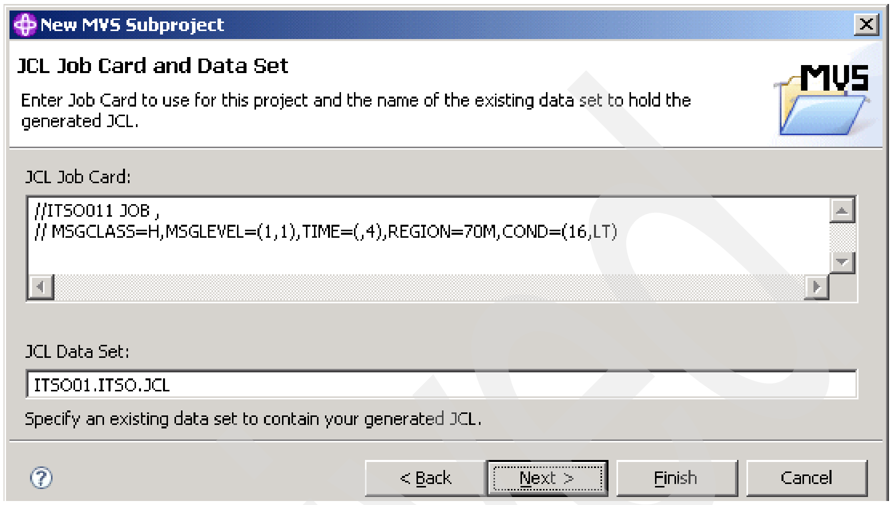
Subproject Name: AD05_COBOL_CICS_Service

Subproject Type: MVS

High Level Qualifier: ITSO02

Figure 2-37 New MVS Subproject - Subproject Name

3. Change the name of the job from <jobname> to the name you like to use and change the name of the JCL Data Set to the name of the data set you will be using for the compile and link-edit JCL. Click **Next**. See Figure 2-38.



Enter Job Card to use for this project and the name of the existing data set to hold the generated JCL.

JCL Job Card:

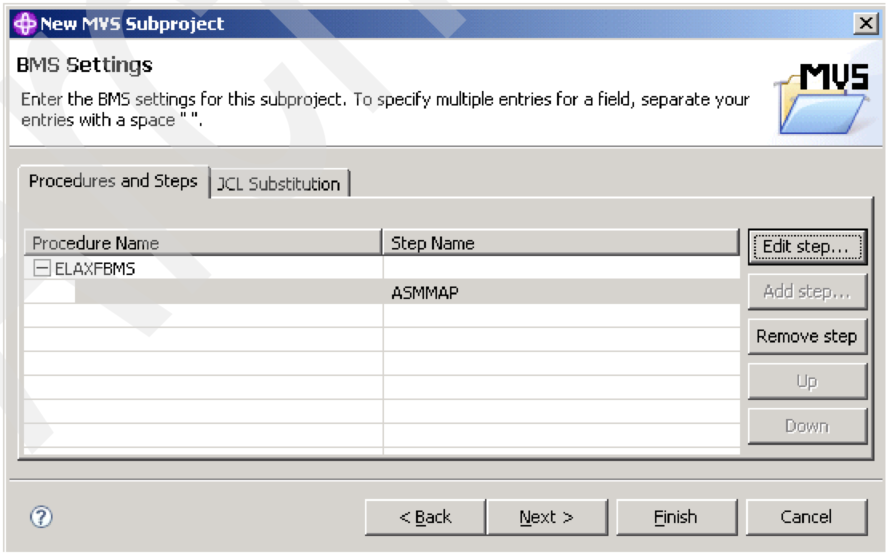
```
//ITSO011 JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,4),REGION=70M,COND=(16,LT)
```

JCL Data Set: ITSO01.ITSO.JCL

Specify an existing data set to contain your generated JCL.

Figure 2-38 New MVS Subproject - JCL job card and JCL data set

4. On the BMS Settings panel, click **Next**.
5. On the BMS Settings panel, expand procedure name **ELAXFBMS**. Select **ASMMAP** and click **Edit step**. See Figure 2-39.



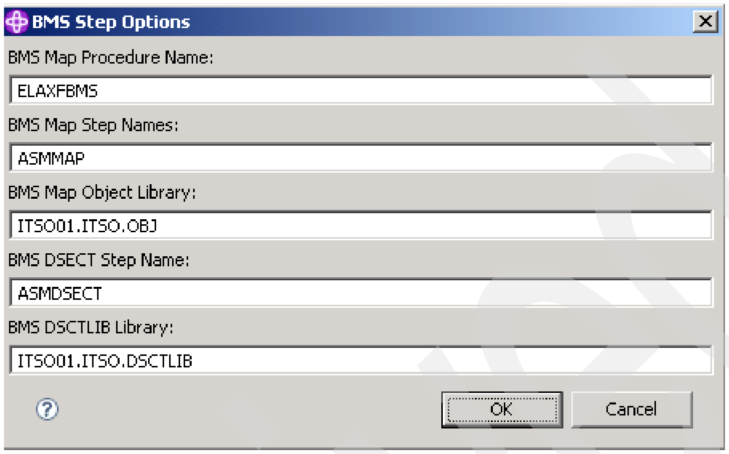
Enter the BMS settings for this subproject. To specify multiple entries for a field, separate your entries with a space " ".

Procedure Name	Step Name
ELAXFBMS	
	ASMMAP

Figure 2-39 New MVS Subproject - BMS settings

6. On the BMS Step Options window, change the name of BMS Map Object Library and the name of BMS DSCTLIB Library to the names you are using on your z/OS system. Click **OK.** You will return to the BMS Step Options window. Click **Next**. See Figure 2-40.

.



BMS Map Procedure Name: ELAXFBMS

BMS Map Step Names: ASMMAP

BMS Map Object Library: ITSO01.ITSO.OBJ

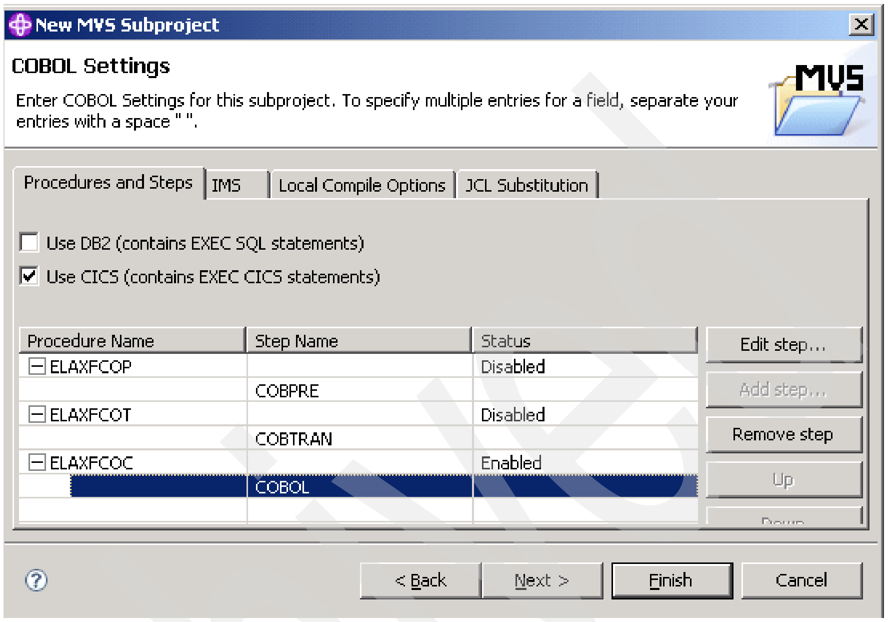
BMS DSECT Step Name: ASMDSECT

BMS DSCTLIB Library: ITSO01.ITSO.DSCTLIB

Figure 2-40 New Subproject - BMS Step Options

7. On the windows PLI Settings, C / C ++ Settings, and Assembler Settings, click **Next**.

8. On the COBOL Settings window make sure that the Use CICS checkbox is checked. Expand Procedure Name **ELAXFCOC**, select the **COBOL** step name and click **Edit step**. See Figure 2-41.



New MVS Subproject

COBOL Settings

Enter COBOL Settings for this subproject. To specify multiple entries for a field, separate your entries with a space " ".

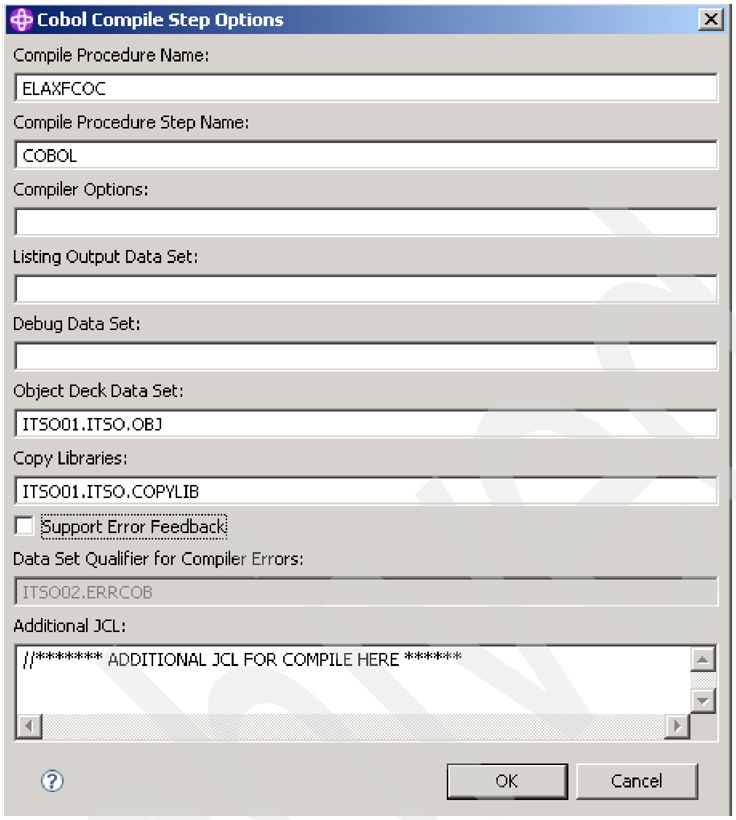
☐ Use DB2 (contains EXEC SQL statements)

☑ Use CICS (contains EXEC CICS statements)

Procedure Name	Step Name	Status
ELAXFCOP		Disabled
	COBPRE	
ELAXFCOT		Disabled
	COBTRAN	
ELAXFCOC		Enabled
	COBOL	

Figure 2-41 New MVS Subproject - COBOL Settings

9. On the COBOL Compile Step Options panel, change Debug Dataset to blank, specify the value for Object Deck Dataset (which in our case is ITSO01.ITSO.OBJ), specify the value for Copy Libraries (which in our case is ITSO01.ITSO.COPYLIB), and make sure that Support Error Feedback checkbox is unchecked. Click **OK**. You will return to the COBOL Settings panel. Click **Next**.


Cobol Compile Step Options

Compile Procedure Name: ELAXFCOC

Compile Procedure Step Name: COBOL

Compiler Options:

Listing Output Data Set:

Debug Data Set:

Object Deck Data Set: ITSO01.ITSO.OBJ

Copy Libraries: ITSO01.ITSO.COPYLIB

☐ Support Error Feedback

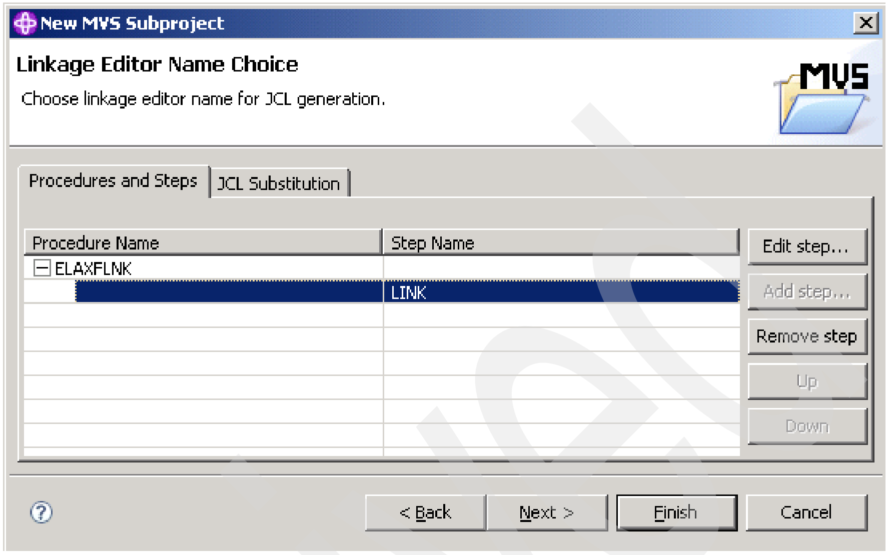
Data Set Qualifier for Compiler Errors: ITSO02.ERRCOB

Additional JCL:

```
//******** ADDITIONAL JCL FOR COMPILE HERE *******
```

Figure 2-42 New MVS Subproject - Cobol options

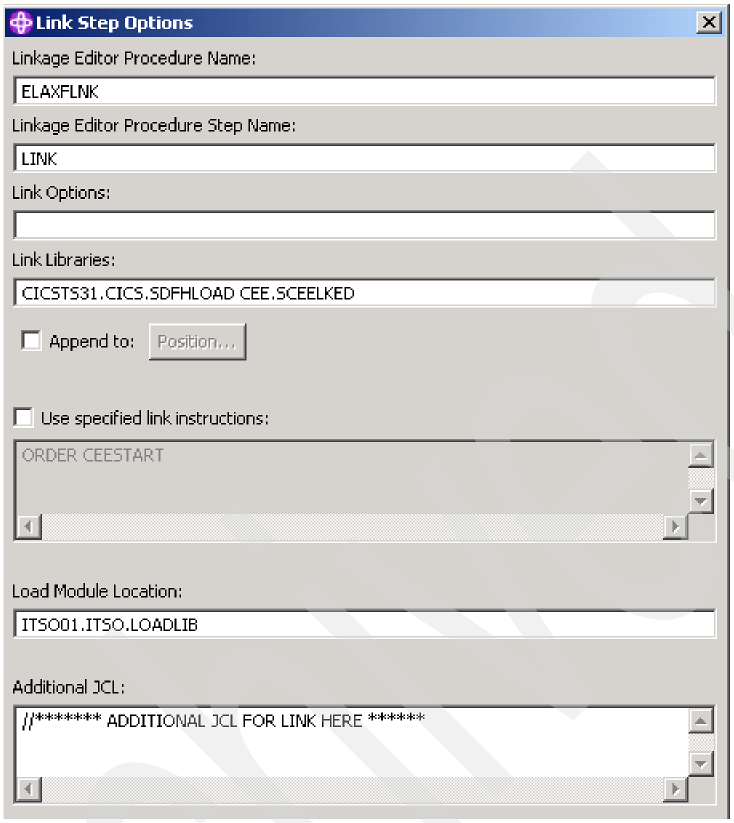
10. On the Linkage Editor Name Choice window expand Procedure Name ELAXFLNK, select LINK step name and click **Edit step**. See Figure 2-43.



Procedure Name	Step Name
ELAXFLNK	
	LINK

Figure 2-43 New MVS Subproject - Linkage Editor name

11. On the Link Step Options window specify the value for the Link Libraries. In our case we used CICSTS31.CICS.SDFHLOAD CEE.SCEELKED. Also, specify the value for Load Module Location, which in our case is TSO01.ITSO.LOADLIB. Click **OK**. You will return to the Linkage Editor Name Choice panel. Click **Next**.


Link Step Options

Linkage Editor Procedure Name: ELAXFLNK

Linkage Editor Procedure Step Name: LINK

Link Options:

Link Libraries: CICSTS31.CICS.SDFHLOAD CEE.SCEELKED

☐ Append to: Position...

☐ Use specified link instructions:

```
ORDER CEESTART
```

Load Module Location: ITSO01.ITSO.LOADLIB

Additional JCL:

```
//******* ADDITIONAL JCL FOR LINK HERE ******
```

Figure 2-44 New MVS Project - Link Step Option

12. On the Application Entry Point window click **Finish** to create the project.

Adding resources to the MVS subproject

To make the data sets available to your remote subproject, you need to add them. In our example we just want to add the CUSTINQD program generated by the Web Service wizard.

Switch to the Remote Systems view (on the right). Expand the MVS Files until you see the CUSTINQD.cbl program that you have moved to your COBOL data set (in our case ITSO01.ITSO.COBOL).

Right-click **CUSTINQD.cbl** and select **Add To Subproject**, as shown in Figure 2-45.

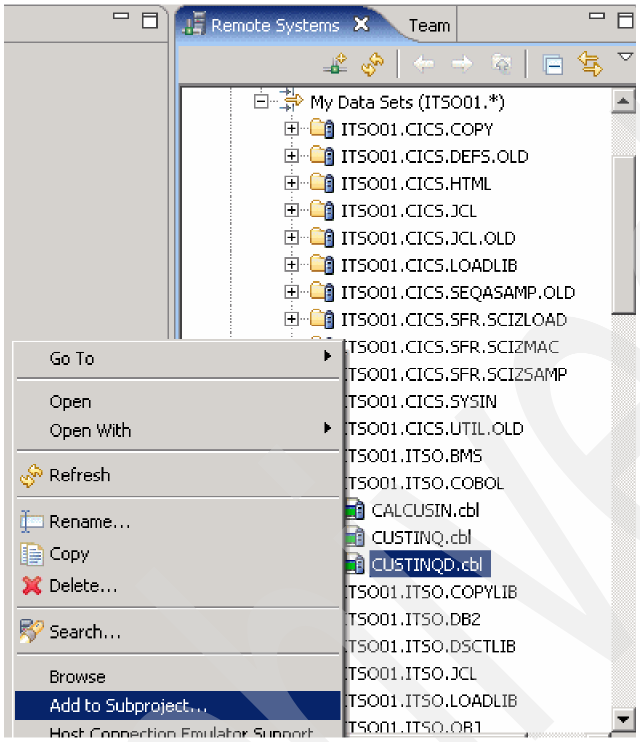
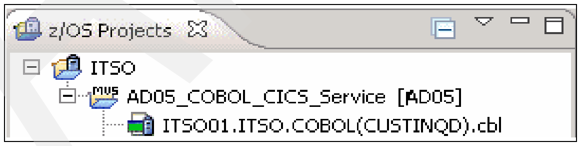


Figure 2-45 Adding resources to the MVS subproject

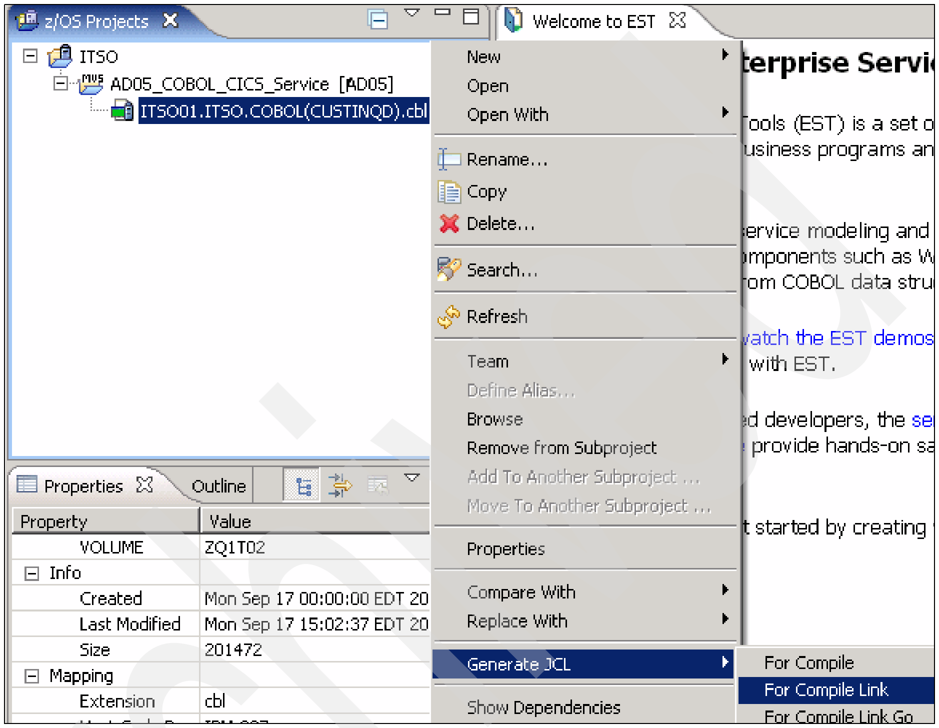
On the Add Resources panel click **Finish** to add the data set to the project.

Switch to the z/OS Projects view, expand the AD05_COBOL_CICS_Service subproject, and you will see that the .cbl member is now defined, as shown in Figure 2-46. In our case the name is ITSO02.ITSO.COBOL(CUSTINQD).cbl.


z/OS Projects
ITSO
AD05_COBOL_CICS_Service [AD05]
ITSO01.ITSO.COBOL(CUSTINQD).cbl
Figure 2-46 MVS subproject - Cobol program resource

Generating JCL to compile and Link CUSTINQD.cbl

To generate JCL, right-click the COBOL member and select **Generate JCL for Compile Link**, as shown in Figure 2-47.

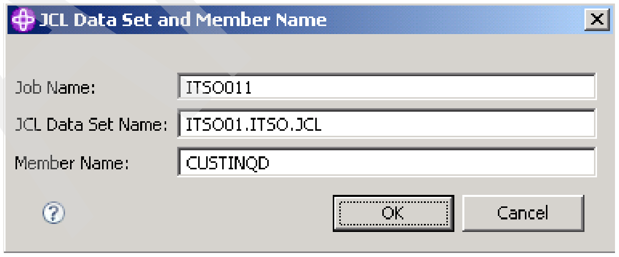


Property	Value
VOLUME	ZQ1T02
Info	
Created	Mon Sep 17 00:00:00 EDT 20
Last Modified	Mon Sep 17 15:02:37 EDT 20
Size	201472
Mapping	
Extension	cbl

terprise Servi
Tools (EST) is a set o
usiness programs an
ervice modeling and
mponents such as W
om COBOL data stru
vatch the EST demos
with EST.
d developers, the se
provide hands-on sa
t started by creating

Figure 2-47 Generating JCL to compile and link the COBOL program

You will then see a windown asking about the Job Name, JCL Dataset Name, and Member Name, as shown in Figure 2-48. Accept the defaults and click **OK**.



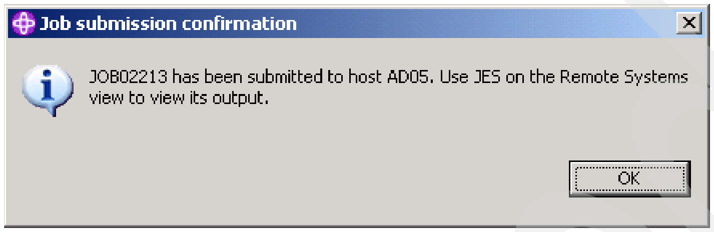
Job Name:	ITSO011
JCL Data Set Name:	ITSO01.ITSO.JCL
Member Name:	CUSTINQD

Figure 2-48 Generating JCL - default values

You can check the JCL by double-clicking in the source of the JCL.

To submit this job on z/OS, make sure that the z/OS connection AD05 is connected. Then right-click the JCL member and select **Submit**. When a pop-up appears, click **OK**.

You will then see an informational window with the job name ID, as shown in Figure 2-49.

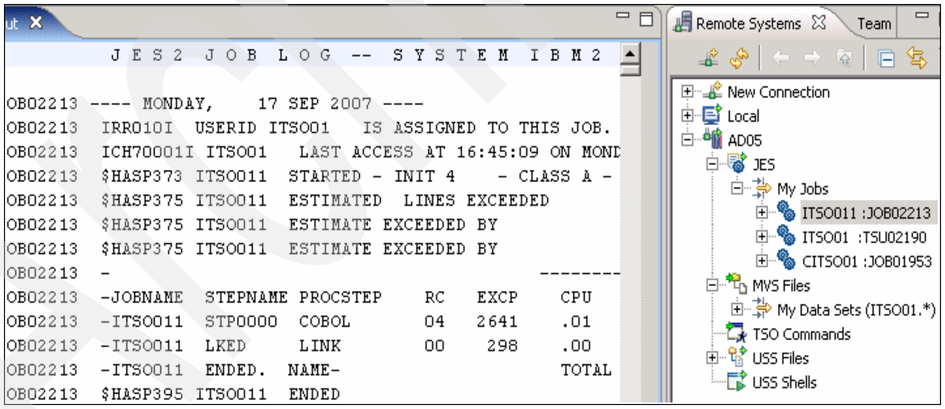


```
JOB02213 has been submitted to host AD05. Use JES on the Remote Systems
view to view its output.
```

Figure 2-49 Submitting JCL

You may check job execution using JES on the Remote System view (on the right). Right-click **My Jobs** and select **Refresh**. Double-click the name of the job that appears on the panel above (ITSO01.JOBnnnnn in our example).

The Job Output Retrieval pop-up window appears (Figure 2-50), asking for the lines. Just click **OK**. It will show you the job output. Make sure the COBOL and LINK return codes are 04 and 00.



```
         J E S 2  J O B  L O G  --  S Y S T E M  I B M 2

OB02213 ---- MONDAY,    17 SEP 2007 ----
OB02213  IRR010I  USERID ITSO01    IS ASSIGNED TO THIS JOB.
OB02213  ICH70001I ITSO01   LAST ACCESS AT 16:45:09 ON MOND
OB02213  $HASP373 ITSO011  STARTED - INIT 4    - CLASS A -
OB02213  $HASP375 ITSO011  ESTIMATED  LINES EXCEEDED
OB02213  $HASP375 ITSO011  ESTIMATE EXCEEDED BY
OB02213  $HASP375 ITSO011  ESTIMATE EXCEEDED BY
OB02213  -                                          --------
OB02213  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU
OB02213  -ITSO011  STP0000  COBOL       04   2641    .01
OB02213  -ITSO011  LKED     LINK        00    298    .00
OB02213  -ITSO011  ENDED.  NAME-                    TOTAL
OB02213  $HASP395 ITSO011  ENDED
```

Figure 2-50 Output Job

Note: Your load module has been created in the load library you specified earlier (in our case ITSO01.ITSO.LOADLIB) on z/OS. This library must be available to your CICS region through the DFHRPL concatenation in the CICS startup job. If the CICS program autoinstall is not active, you need to define a PPT for CUSTINQD. We will not discuss the PPT creation here since it is not our objective.

If you want to purge this job, just right-click its name and select **Purge.**

Installing or rescanning PIPELINE

When we install each PIPELINE resource, CICS scans the directory specified in the PIPELINE's WSDIR attribute (the pickup directory). For each Web Service binding file in the directory (files with the .wsbind suffix), CICS installs a WEBSERVICE and a URIMAP if one does not already exist. Existing resources are replaced if the information in the binding file is newer than the existing resources.

If a new WSBind file is copied to the pickup directory, a PIPELINE SCAN is required. The PERFORM PIPELINE SCAN command initiates the scan of the PIPELINE pickup directory.

To do this, switch to the z/OS perspective. On the Remote Systems tab (on the right) right-click the **AD05** connection, and select **Host Connection Emulator Support**.

When the CICS signon is complete, be sure you maximize the windows so as to better view the output.

Let us now inspect the CICS WEBSERVICE definition, as follows:

1. Type `CEMT PERFORM PIPELINE(PIPEITSO) SCAN` and press the Enter key. You should receive the RESPONSE NORMAL message, as shown in Example 2-4.

Example 2-4 Performing a PIPELINE SCAN in CICS

```
CEMT PERFORM PIPELINE(PIPEITSO) SCAN
  STATUS:  RESULTS
   Pip(PIPEITSO) Sca
SYSID=WRKS APPLID=CITSO01
   RESPONSE: NORMAL                            TIME:  08.34.12  DATE: 03.31.08
 PF 1 HELP      3 END      5 VAR       7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

2. Press F3 to end this session.

Before we inspect the CICS PIPELINE, let us inspect the WEBSERVICE and URIMAP resources defined to CICS for the CUSTINQ program.

3. To inspect the CICS WEBSERVICE definition, first delete the previous command, type `CEMT INQ WEBSERVICE(CUSTINQ)` and press the Enter key. Scroll through the defined WEBSERVICE definitions (press F7/F8) looking for the Web Service that we created for our CUSTINQ program. See Figure 2-5.

Example 2-5 Checking the CICS WEBSERVICE

```
CEMT INQ WEBSERVICE(CUSTINQ)
  STATUS:  RESULTS - OVERTYPE TO MODIFY
   Webs(CUSTINQ                         ) Pip(PIPEITSO)
      Ins Uri($322570 ) Pro(CUSTINQ ) Com                      Dat(20070515)

SYSID=WRKS APPLID=CITSO01
   RESPONSE: NORMAL                              TIME:  08.39.07  DATE: 03.31.08
 PF 1 HELP       3 END       5 VAR        7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

4. Move the cursor to the CUSTINQ WEBSERVICE definition and press Enter to see the details, as shown in Example 2-6.

Example 2-6 WEBSERVICE details

```
CEMT INQ WEBSERVICE(CUSTINQ)
  RESULT - OVERTYPE TO MODIFY
    Webservice(CUSTINQ)
    Pipeline(PIPEITSO)
    Validationst( Novalidation )
    State(Inservice)
    Urimap($322570)
    Program(CUSTINQ)
    Pgminterface(Commarea)
    Container()
    Datestamp(20070515)
    Timestamp(23:22:57)
    Wsdlfile()
    Wsbind(/u/itso01/itso/wspickup/provider/CUSTINQ.wsbind)
    Endpoint(http://wtsc47.itso.ibm.com:3001/itso/CUSTINQ)
    Binding(CUSTINQBinding)




                                                      SYSID=WRKS APPLID=CITSO01
                                                TIME:  08.41.00  DATE: 03.31.08
```

```
 PF 1 HELP 2 HEX 3 END       5 VAR        7 SBH 8 SFH       10 SB 11 SF
```

After the PIPELINE is installed or rescanned, it automatically installs the WEBSERVICE and URIMAP resources that are associated with a WSBind file. The example above shows that this is related to program CUSTINQ (the existing application) and an auto URIMAP ($322570).

5. To check the URIMAP, take the URIMAP name above and type CEMT INQ URIMAP($322570), as shown in Example 2-7.

Example 2-7 Browsing the URIMAP

```
CEMT INQ URIMAP($322570)
  STATUS:  RESULTS - OVERTYPE TO MODIFY
   Uri($322570 ) Pip Ena     Http
      Host(*                        ) Path(/itso/CUSTINQ                  )

                                                        SYSID=WRKS APPLID=CITSO01
   RESPONSE: NORMAL                                  TIME:  08.43.19  DATE: 03.31.08
 PF 1 HELP       3 END       5 VAR        7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

6. Move the cursor to the URI($322570) and press Enter. You will see the details about the URIMAP, as shown in Example 2-8.

Example 2-8 Displaying URIMAP details

```
CEMT INQ URIMAP($322570)
  RESULT - OVERTYPE TO MODIFY
    Urimap($322570)
    Usage(Pipe)
    Enablestatus( Enabled )
    Analyzerstat(Noanalyzer)
    Scheme(Http)
    Redirecttype( None )
    Tcpipservice()
    Host(*)
    Path(/itso/CUSTINQ)
    Transaction(CPIH)
    Converter()
    Program()
    Pipeline(PIPEITSO)
    Webservice(CUSTINQ)
    Userid()
    Certificate()
    Ciphers()
 +  Templatename()

                                                        SYSID=WRKS APPLID=CITSO01
```

```
                                                    TIME:  08.45.13  DATE: 03.31.08
 PF 1 HELP 2 HEX 3 END       5 VAR        7 SBH 8 SFH       10 SB 11 SF
```

The example shows the definition for the auto generated URIMAP $322570. It shows a path of /itso/CUSTINQ, which was taken from the WSBind file. When a request comes into CICS with a local URI of /itso/CUSTINQ, this URIMAP definition will match and the request will then be associated with WEBSERVICE CUSTINQ, which has a reference to the WSBind file.

The WSBind file contains the name of the conversion program that will be invoked by CICS Web Service.

7. Press F3 to end your CEMT session and remove the text from the screen. Then type `CEDA DISPLAY GROUP(*) PROGRAM(CUSTINQD)` to install the CUSTINQD program. Then type `i` next to the CUSTINQD program and press Enter. See Example 2-9.

Example 2-9   Installing the CICS program

```
DISPLAY GROUP(*) PROGRAM(CUSTINQD)
 ENTER COMMANDS
  NAME     TYPE         GROUP                                        DATE   TIME
  CUSTINQD PROGRAM      ITSOTEST i                                INSTALL SUCCESSFUL

                                                          SYSID=WRKS APPLID=CITSO01
  RESULTS: 1 TO 1 OF 1                             TIME:  08.49.31  DATE: 08.091
PF 1 HELP       3 END 4 TOP 5 BOT 6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

You should receive the message `INSTALL SUCCESSFUL` on the screen. Then press F3 to end your CEDA session and close the terminal emulation by clicking **X** on the tab name.

Note that some basic CICS tasks were not discussed here, even though they need to be performed. For example, making the converter driver program available by adding it to a library that is part of the DFHRPL concatenation. In addition, a CICS PROGRAM resource must be defined if program auto-install is not enabled for the target CICS system.

2.5.6 Testing the CICS-based Web Service

Now you are ready to invoke the Web Service you created. We will use the generated WSDL file and invoke the CICS Service using WDz.

1. Switch to the Enterprise Tools perspective. On the EST Project Explorer tab, under the project ITSO_AD05_CICS_Service, Generation, Targets, right-click **CUSTINQ.wsdl** and select **Web Services Test with Web Services Explorer**, as shown in Figure 2-51.

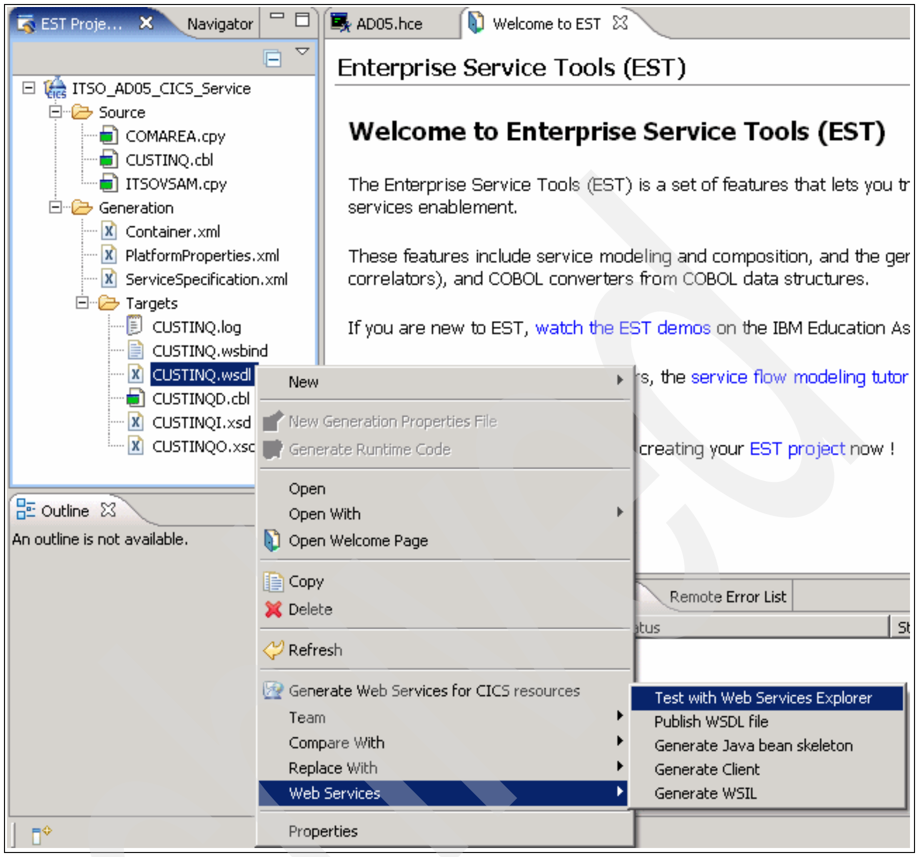
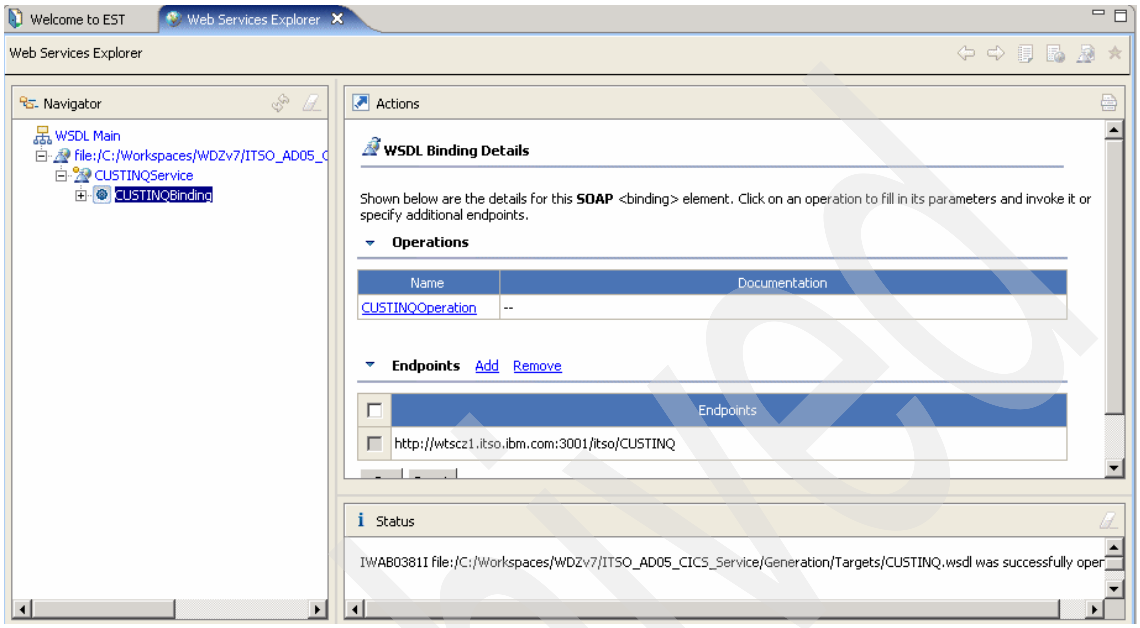

EST Proje...
Navigator
AD05.hce
Welcome to EST
ITSO_AD05_CICS_Service
Source
COMAREA.cpy
CUSTINQ.cbl
ITSOVSAM.cpy
Generation
Container.xml
PlatformProperties.xml
ServiceSpecification.xml
Targets
CUSTINQ.log
CUSTINQ.wsbind
CUSTINQ.wsdl
CUSTINQD.cbl
CUSTINQI.xsd
CUSTINQO.xsd
Outline
An outline is not available.
Enterprise Service Tools (EST)
Welcome to Enterprise Service Tools (EST)
The Enterprise Service Tools (EST) is a set of features that lets you tr
services enablement.
These features include service modeling and composition, and the ger
correlators), and COBOL converters from COBOL data structures.
If you are new to EST, watch the EST demos on the IBM Education As
rs, the service flow modeling tutor
creating your EST project now !
Remote Error List
New
New Generation Properties File
Generate Runtime Code
Open
Open With
Open Welcome Page
Copy
Delete
Refresh
Generate Web Services for CICS resources
Team
Compare With
Replace With
Web Services
Properties
Test with Web Services Explorer
Publish WSDL file
Generate Java bean skeleton
Generate Client
Generate WSIL

Figure 2-51 Selecting the Web Services Explorer for testing

The Web Service will be launched in a Web Browser view. Resize the view to see all areas of the Web Services Explorer. You can double-click the Web Service Explorer title to display the full screen. See Figure 2-52.



Shown below are the details for this **SOAP** <binding> element. Click on an operation to fill in its parameters and invoke it or specify additional endpoints.

Operations

Name	Documentation
CUSTINQOperation	--

Endpoints Add Remove

	Endpoints
☐	http://wtsc z1.itso.ibm.com:3001/itso/CUSTINQ

Status

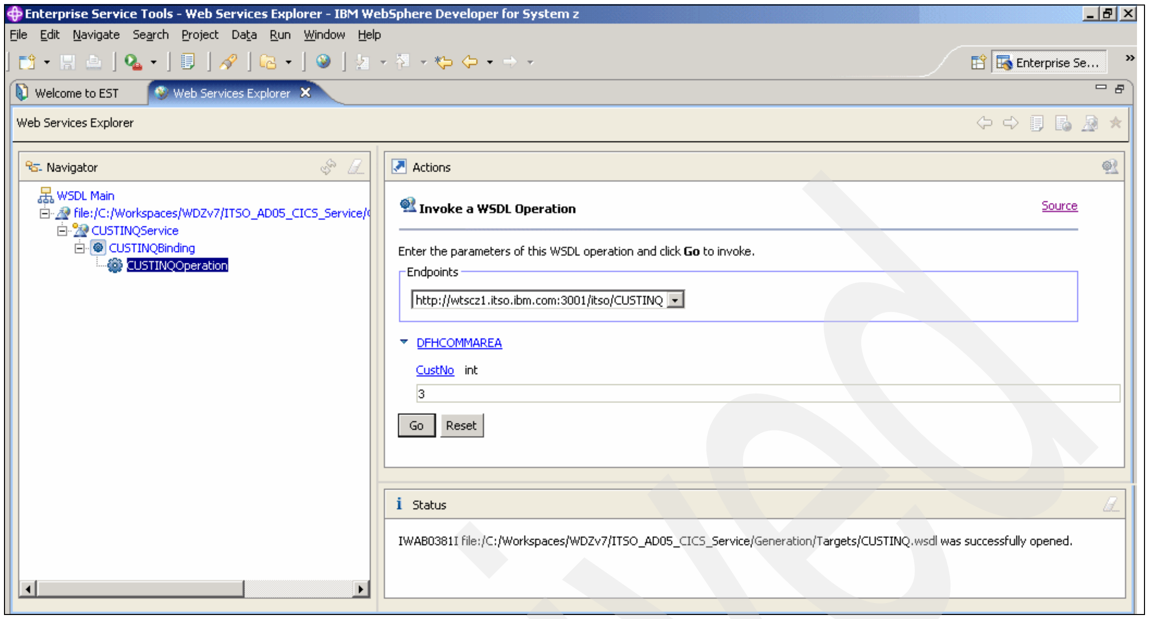
IWAB0381I file:/C:/Workspaces/WDZv7/ITSO_AD05_CICS_Service/Generation/Targets/CUSTINQ.wsdl was successfully oper

Figure 2-52 Web Services Explorer window

2. Click the link for the **CUSTINQOperation** operations in the Actions section (on the right).

   Note that the Actions section is now replaced with information about the CUSTINQOperation operation and its parameters. It also lists the endpoint associated with this request.

3. Enter a value of 3 for CustNo and click **Go** to invoke the operation.



Enter the parameters of this WSDL operation and click **Go** to invoke.

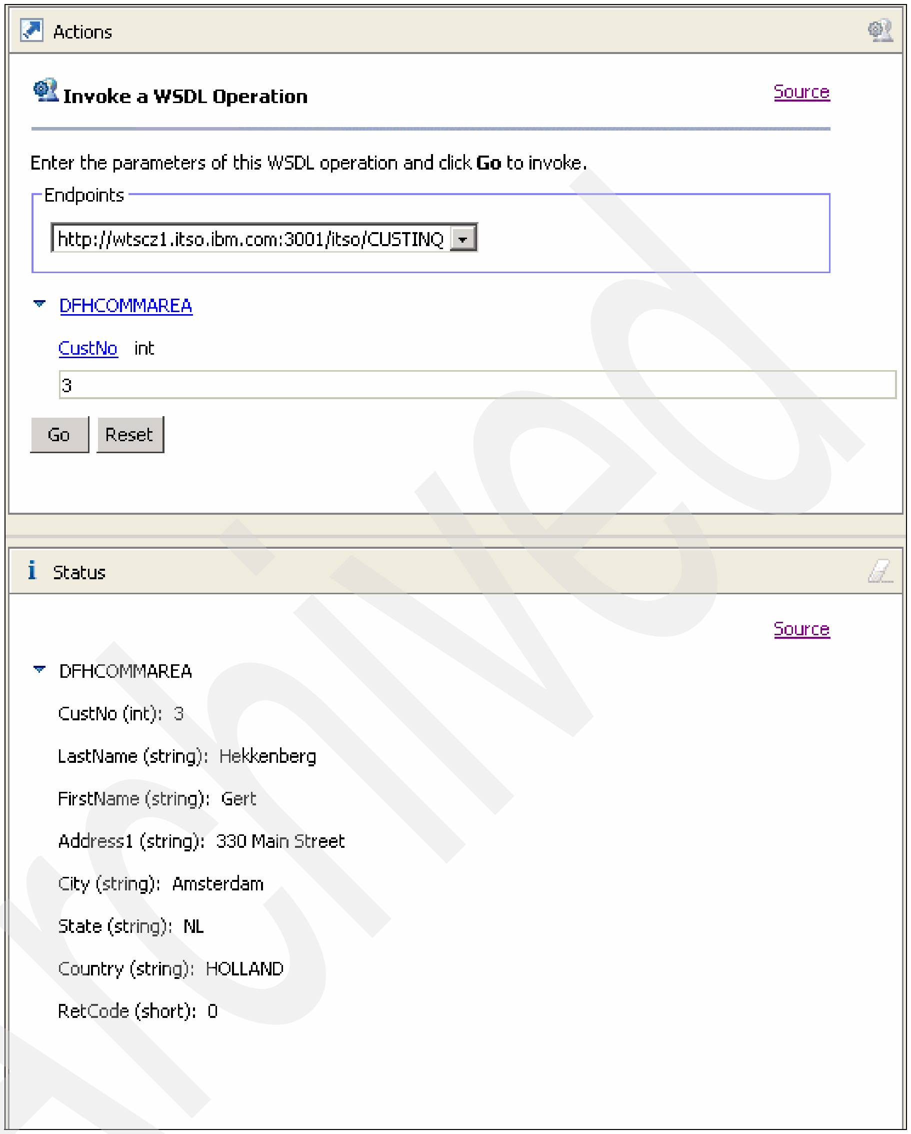
http://wtscz1.itso.ibm.com:3001/itso/CUSTINQ

3

IWAB0381I file:/C:/Workspaces/WDZv7/ITSO_AD05_CICS_Service/Generation/Targets/CUSTINQ.wsdl was successfully opened.

Figure 2-53 Web Services Explorer window

You should see the results of your customer query in the Status section of the Web Services Explorer. Verify that the results returned by your Web Service match the results you previously obtained using the 3270 emulation CICS interface. See Figure 2-54 on page 71.


Actions

Invoke a WSDL Operation

Source

Enter the parameters of this WSDL operation and click **Go** to invoke.

Endpoints

http://wtscz1.itso.ibm.com:3001/itso/CUSTINQ

DFHCOMMAREA

CustNo int

3

Go Reset

Status

Source

DFHCOMMAREA

CustNo (int): 3

LastName (string): Hekkenberg

FirstName (string): Gert

Address1 (string): 330 Main Street

City (string): Amsterdam

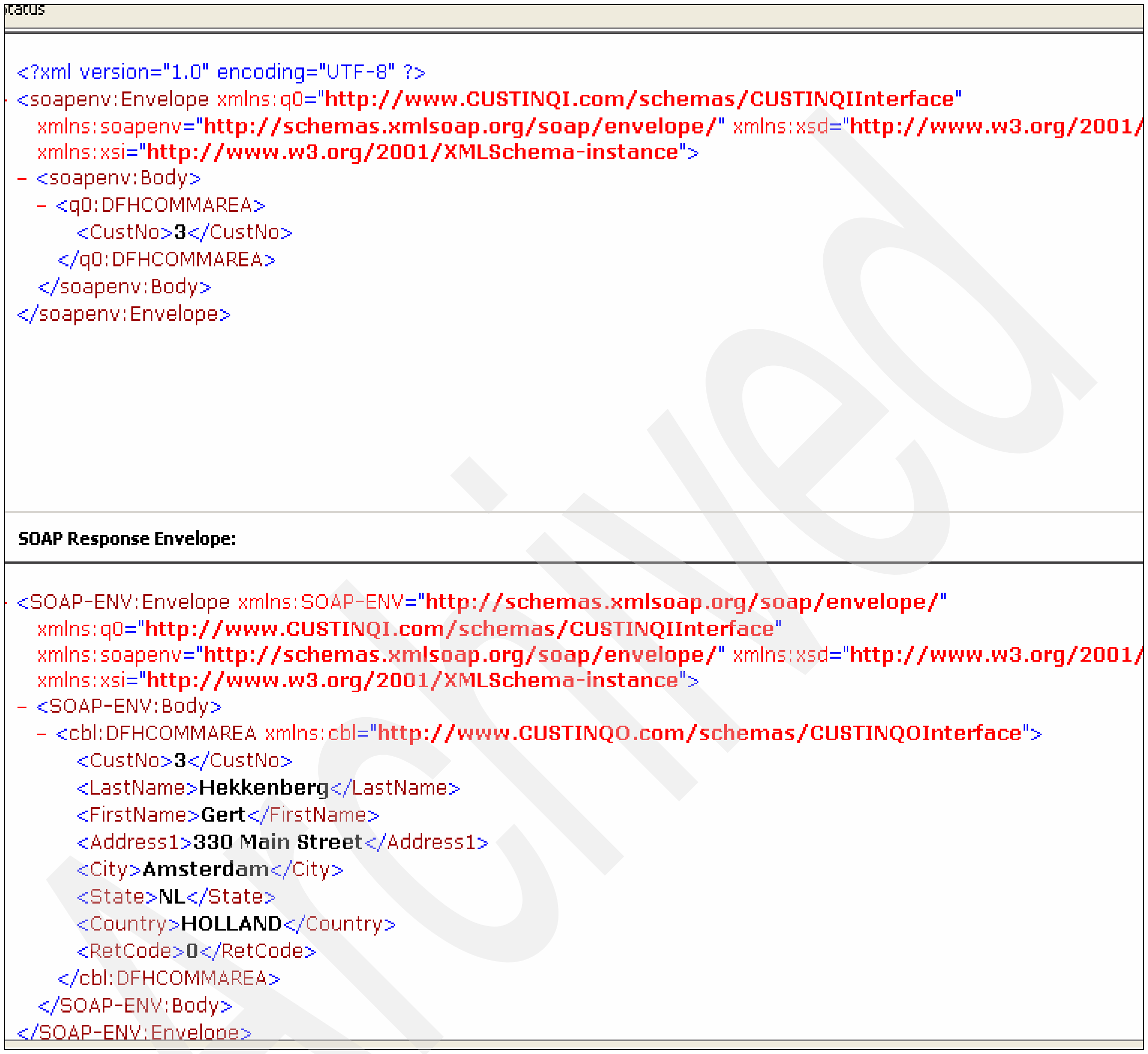
State (string): NL

Country (string): HOLLAND

RetCode (short): 0

Figure 2-54 Querying the result

4. Click **Source**, on the side of Status, to view the XML SOAP Request/Response envelopes. See Figure 2-55.



```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope xmlns:q0="http://www.CUSTINQI.com/schemas/CUSTINQIInterface"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <soapenv:Body>
  - <q0:DFHCOMMAREA>
      <CustNo>3</CustNo>
    </q0:DFHCOMMAREA>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP Response Envelope:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:q0="http://www.CUSTINQI.com/schemas/CUSTINQIInterface"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <SOAP-ENV:Body>
  - <cbl:DFHCOMMAREA xmlns:cbl="http://www.CUSTINQO.com/schemas/CUSTINQOInterface">
      <CustNo>3</CustNo>
      <LastName>Hekkenberg</LastName>
      <FirstName>Gert</FirstName>
      <Address1>330 Main Street</Address1>
      <City>Amsterdam</City>
      <State>NL</State>
      <Country>HOLLAND</Country>
      <RetCode>0</RetCode>
    </cbl:DFHCOMMAREA>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 2-55 SOAP envelope result

2.5.7 Creating the Web Service client

To get an end-to-end application for testing this Web Service we need to create a Web Services client. The following is a step-by-step procedure to create a simple client.

1. Open the J2EE perspective, select your project and click **File** → **New** → **Other**, as shown in Figure 2-56.

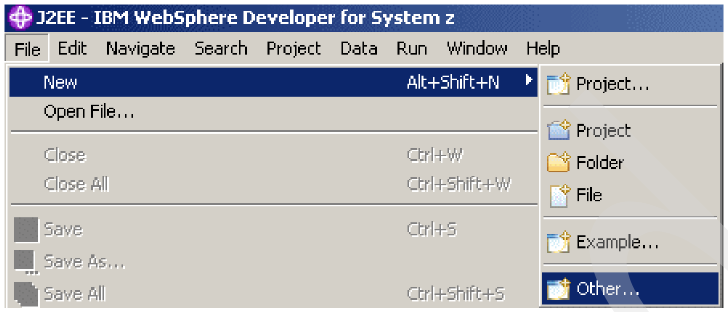


Figure 2-56 New Web Services client

2. Select the **Web Service Client** wizard and click **Next**, as shown in Figure 2-57.

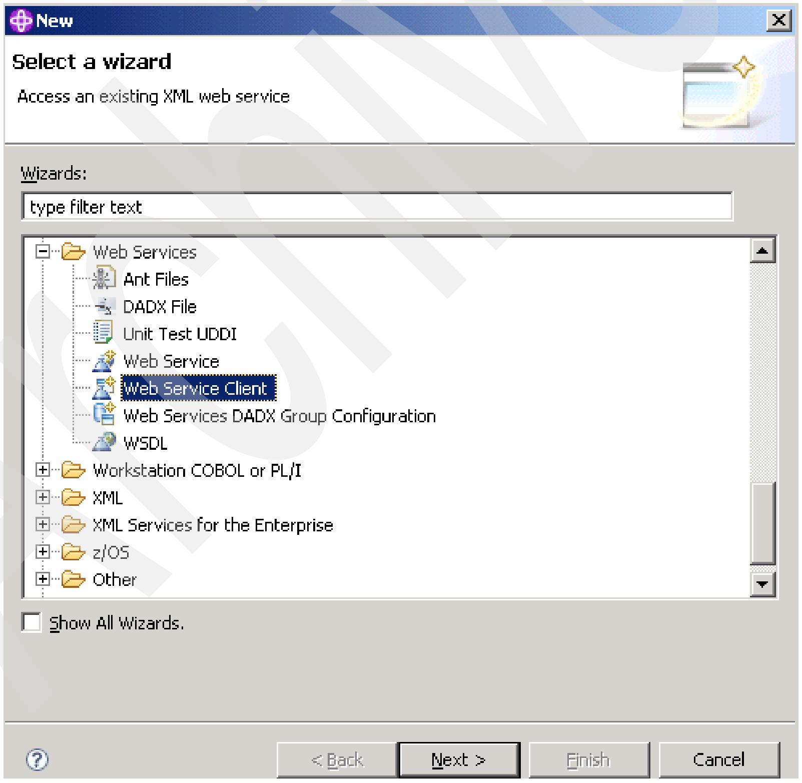
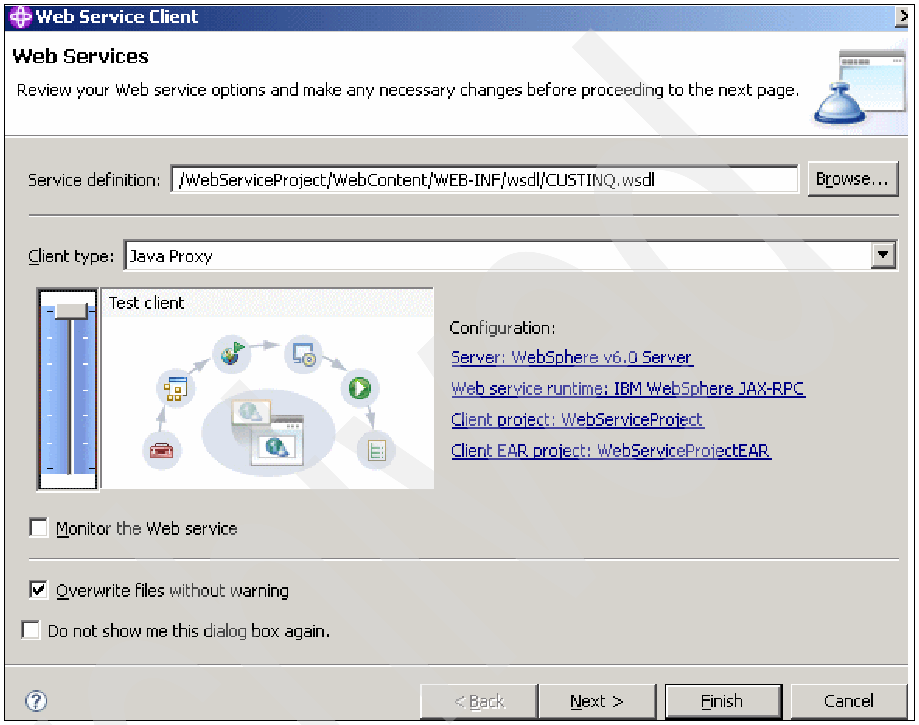


Figure 2-57 Web Service Client wizard

3. At the Web Service Client window, raise the slider bar (behind Client type) to the top to have the highest level of configuration created, inform the CUSTINQ.wsdl file at the service definition field, and click **Next.** See Figure 2-58.



Web Services

Review your Web service options and make any necessary changes before proceeding to the next page.

Service definition: /WebServiceProject/WebContent/WEB-INF/wsdl/CUSTINQ.wsdl

Client type: Java Proxy

Configuration:
Server: WebSphere v6.0 Server
Web service runtime: IBM WebSphere JAX-RPC
Client project: WebServiceProject
Client EAR project: WebServiceProjectEAR

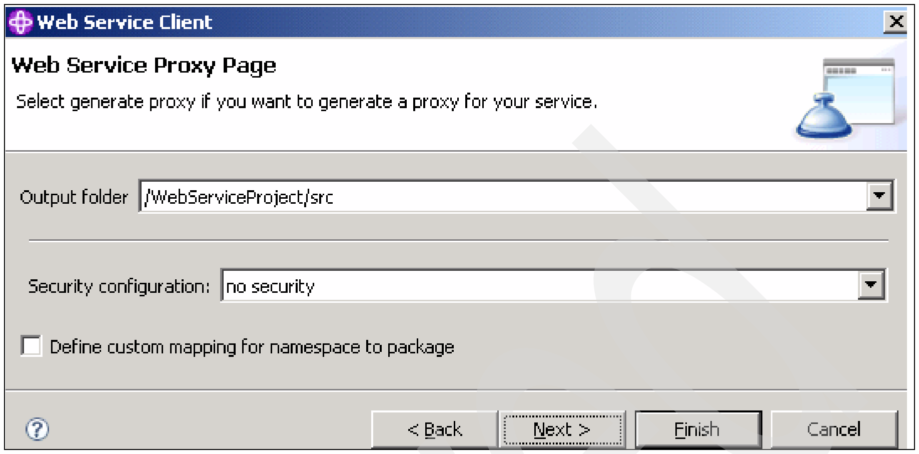
☐ Monitor the Web service

☑ Overwrite files without warning

☐ Do not show me this dialog box again.

Figure 2-58 Web Service Client service definition

4. Select **Next** at the Web Service Proxy Page, as shown in Figure 2-59.


Web Service Client

Web Service Proxy Page

Select generate proxy if you want to generate a proxy for your service.

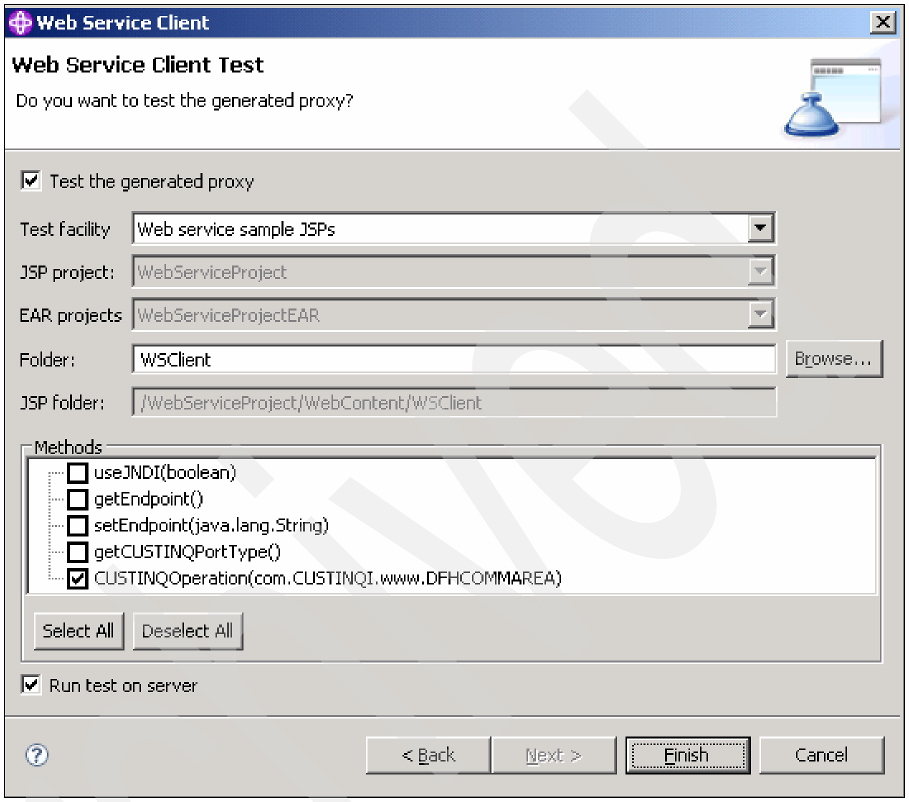
Output folder /WebServiceProject/src

Security configuration: no security

☐ Define custom mapping for namespace to package

Figure 2-59 Web Service Proxy Page

5. You can change the Folder name to WSClient and click **Finish** to create the client. See Figure 2-60.


Web Service Client

Web Service Client Test

Do you want to test the generated proxy?

[✓] Test the generated proxy

Test facility: Web service sample JSPs

JSP project: WebServiceProject

EAR projects: WebServiceProjectEAR

Folder: WSClient

JSP folder: /WebServiceProject/WebContent/WSClient

Methods
- [ ] useJNDI(boolean)
- [ ] getEndpoint()
- [ ] setEndpoint(java.lang.String)
- [ ] getCUSTINQPortType()
- [✓] CUSTINQOperation(com.CUSTINQI.www.DFHCOMMAREA)

[✓] Run test on server

Figure 2-60 Web Service client Test

The client is created now.

6. Right-click the ear project, select **Export** and **EAR file**, as shown in Figure 2-61.

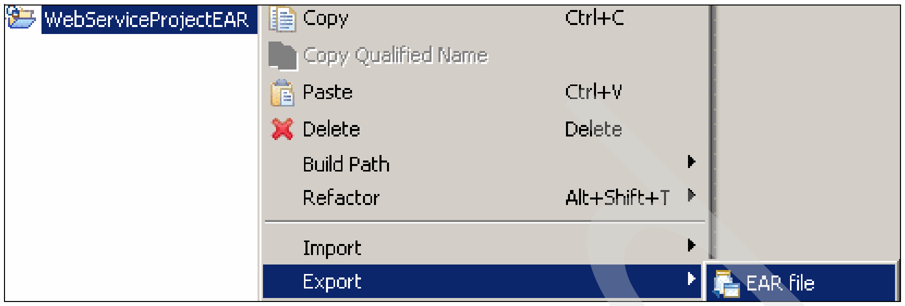
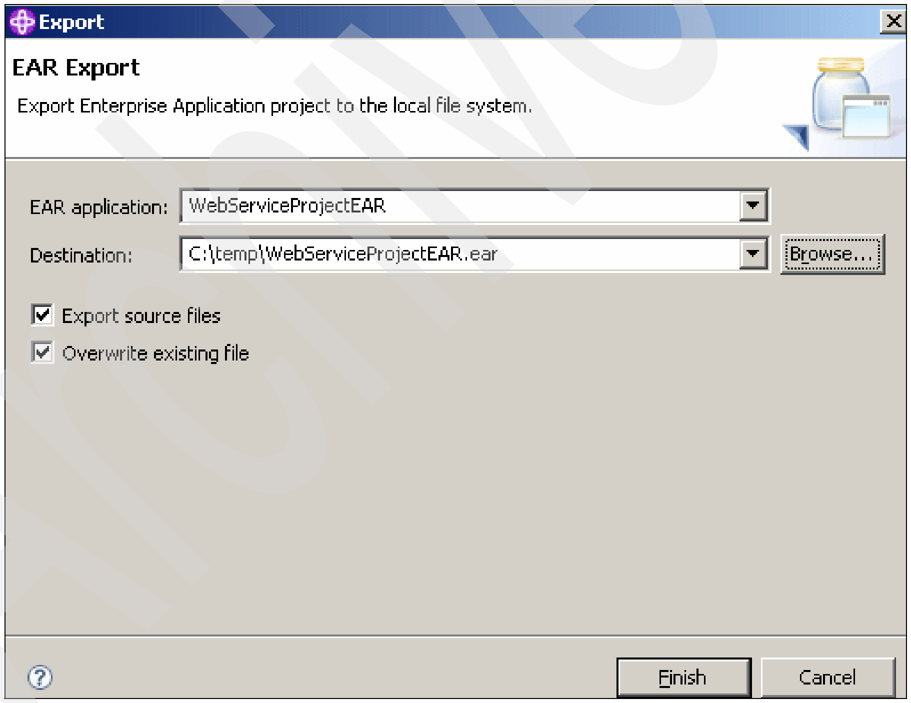


Figure 2-61 Exporting the client

7. Select the destination to the file and click **Finish**, as shown in Figure 2-62.



Export Enterprise Application project to the local file system.

EAR application: WebServiceProjectEAR

Destination: C:\temp\WebServiceProjectEAR.ear

☑ Export source files

☑ Overwrite existing file

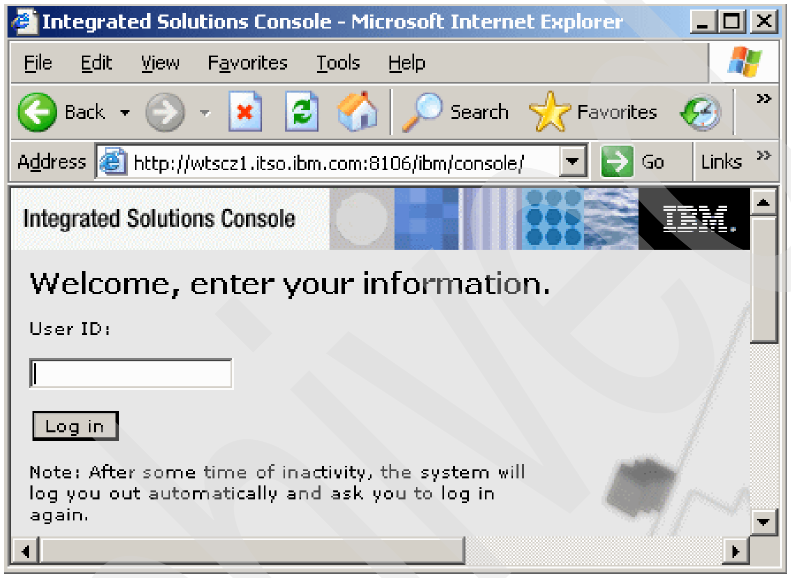
Figure 2-62 Finishing the export of the client

The client is ready to be installed at the WebSphere Application Server.

2.5.8 Deployment to WebSphere Application Server

After all the development steps the application can be deployed to WebSphere Application Server using the WebSphere Admin console.

1. Log on to the WebSphere Admin console. If you have WebSphere security enabled, you must provide a user ID and password to access the console.


Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://wtscz1.itso.ibm.com:8106/ibm/console/ Go Links
Integrated Solutions Console
IBM.
Log in

Welcome, enter your information.

User ID:

Note: After some time of inactivity, the system will log you out automatically and ask you to log in again.

Figure 2-63 WebSphere Admin console logon

2. Install a new enterprise application by selecting **Application** → **Install New Application.** See Figure 2-64.

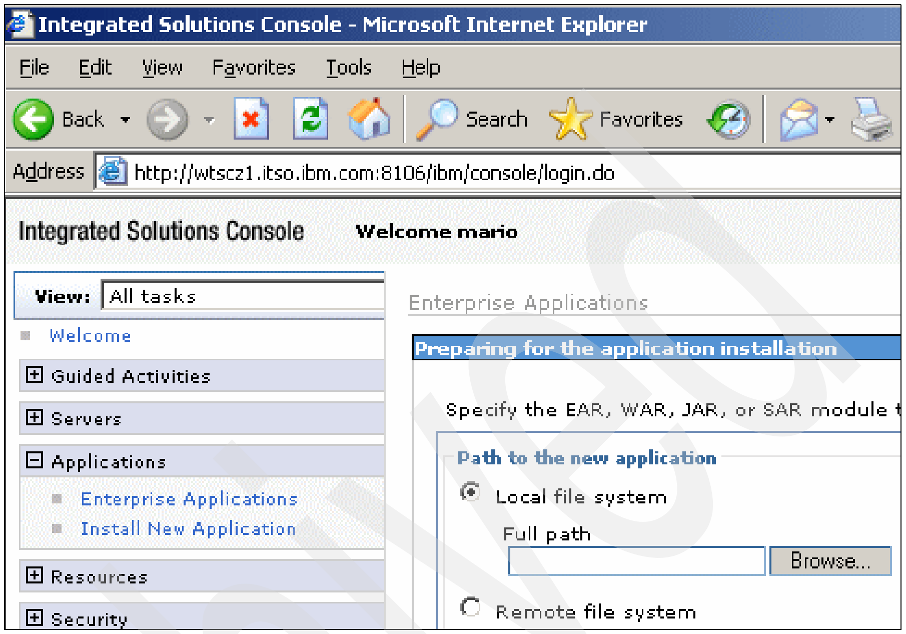
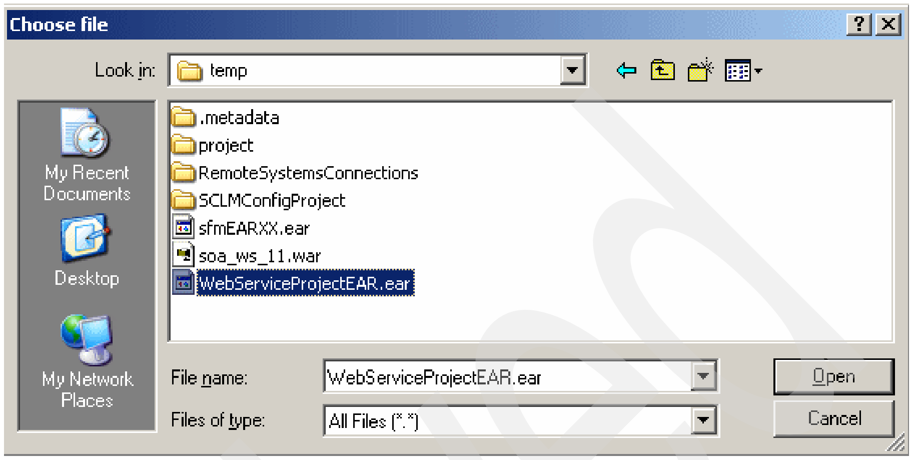

Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://wtsczl.itso.ibm.com:8106/ibm/console/login.do
Integrated Solutions Console
Welcome mario
View: All tasks
Welcome
Guided Activities
Servers
Applications
Enterprise Applications
Install New Application
Resources
Security
Enterprise Applications
Preparing for the application installation
Specify the EAR, WAR, JAR, or SAR module t
Path to the new application
Local file system
Full path
Browse...
Remote file system

Figure 2-64 WebSphere console install new application

3. Select **Local file system** and click **Browse**.

4. At the Choose File screen pane select the **WebServiceProjectEAR.ear** and click **Open.** See Figure 2-65.

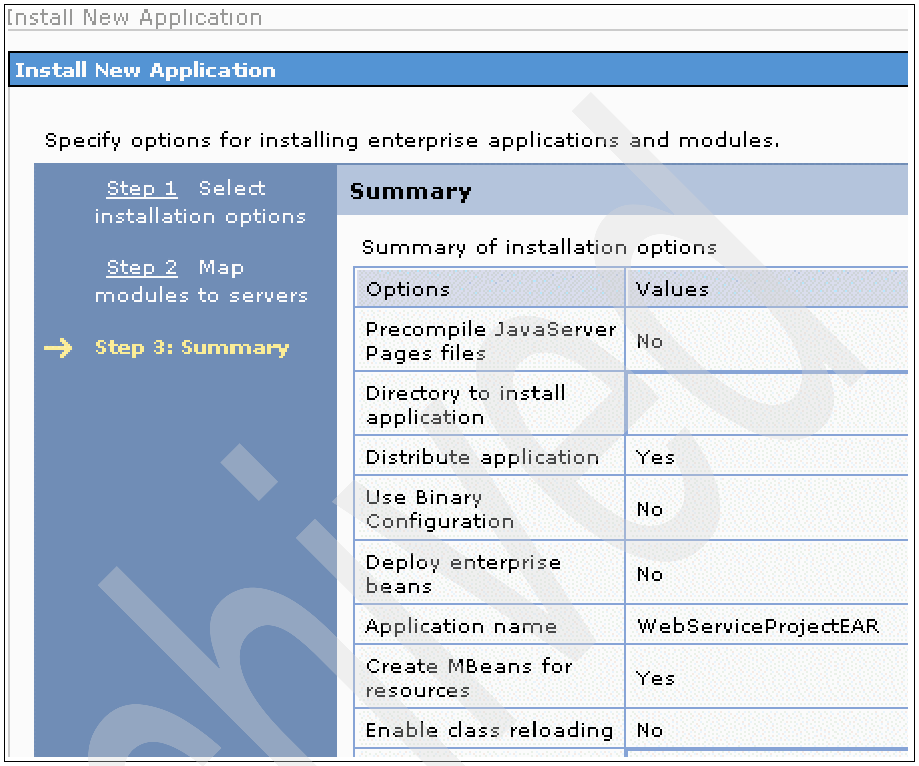


.metadata
project
RemoteSystemsConnections
SCLMConfigProject
sfmEARXX.ear
soa_ws_11.war
WebServiceProjectEAR.ear

File name: WebServiceProjectEAR.ear
Files of type: All Files (*.*)

Figure 2-65 Local file selection

5. Click **Next** until Step 3: Summary, and then click **Finish** to install the application. See Figure 2-66.


Install New Application

Install New Application

Specify options for installing enterprise applications and modules.

Step 1 Select installation options

Step 2 Map modules to servers

Step 3: Summary

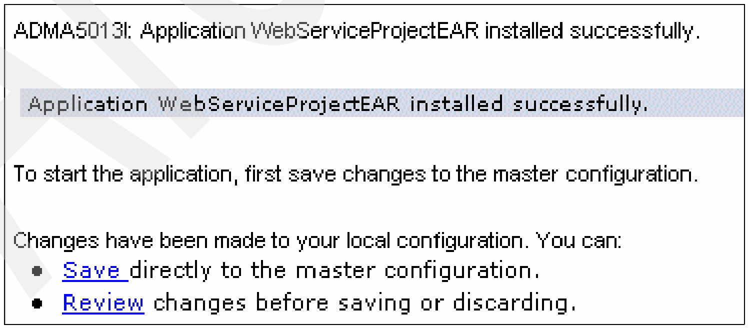
Summary

Summary of installation options

Options	Values
Precompile JavaServer Pages files	No
Directory to install application	
Distribute application	Yes
Use Binary Configuration	No
Deploy enterprise beans	No
Application name	WebServiceProjectEAR
Create MBeans for resources	Yes
Enable class reloading	No

Figure 2-66 WebSphere Admin console application deployment summary

6. When it finishes, check if the installation was successful and click **Save.**


ADMA5013I: Application WebServiceProjectEAR installed successfully.

Application WebServiceProjectEAR installed successfully.

To start the application, first save changes to the master configuration.

Changes have been made to your local configuration. You can:
- Save directly to the master configuration.
- Review changes before saving or discarding.

Figure 2-67 WebSphere Admin console save configuration

7. Expand **Applications** → **Enterprise Applications,** select the **WebServiceProjectEAR**, and click **Start**.

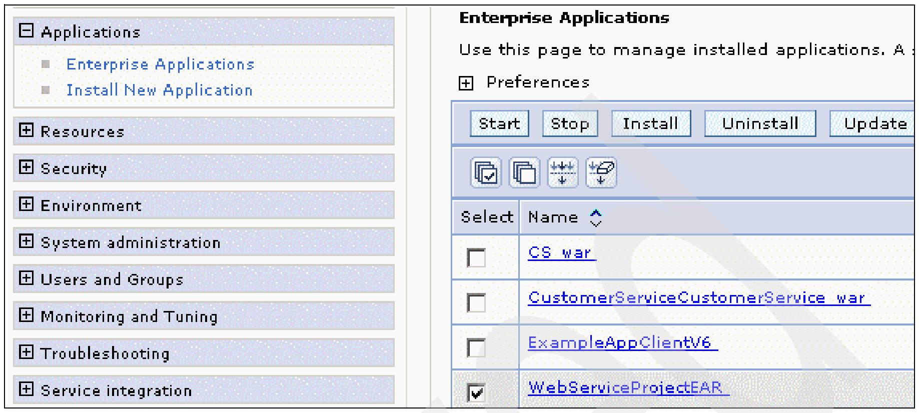
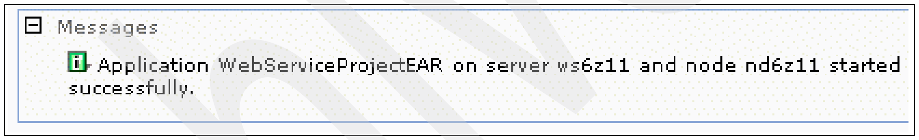


Figure 2-68 WebSphere console start application

8. If you receive the message shown in Figure 2-69, you are ready to test.

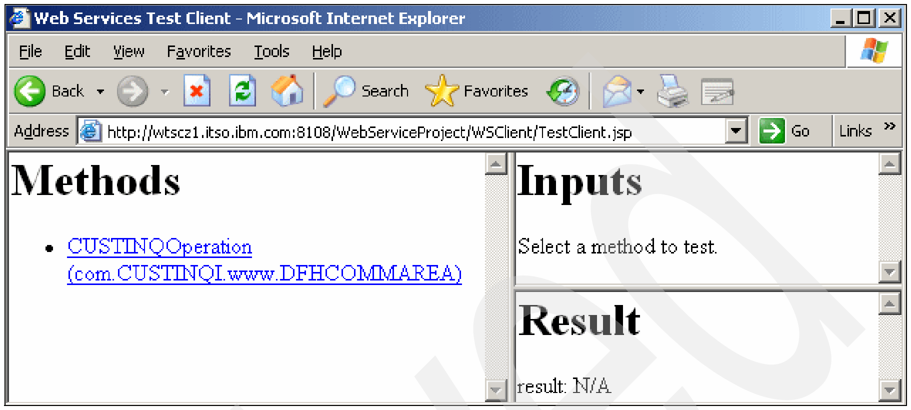


Application WebServiceProjectEAR on server ws6z11 and node nd6z11 started successfully.

Figure 2-69 Success of the installation

2.5.9 Testing the Web Service client

1. Open the browser and call the TestClient.jsp.


Web Services Test Client - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://wtsc21.itso.ibm.com:8108/WebServiceProject/WSClient/TestClient.jsp
Go Links

Methods

- CUSTINQOperation (com.CUSTINQI.www.DFHCOMMAREA)

Inputs

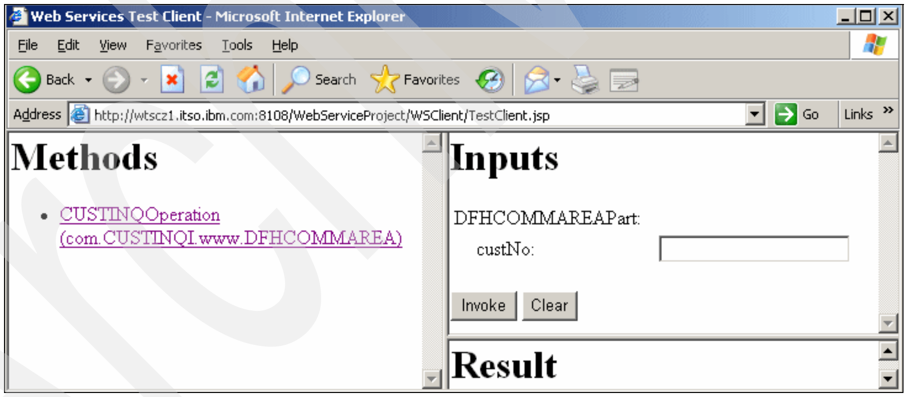
Select a method to test.

Result

result: N/A

Figure 2-70 Testing application initial page

2. Click the **CUSTINQOperation** method, shown in Figure 2-71.


Web Services Test Client - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://wtsc21.itso.ibm.com:8108/WebServiceProject/WSClient/TestClient.jsp
Go Links
Invoke
Clear

Methods

- CUSTINQOperation (com.CUSTINQI.www.DFHCOMMAREA)

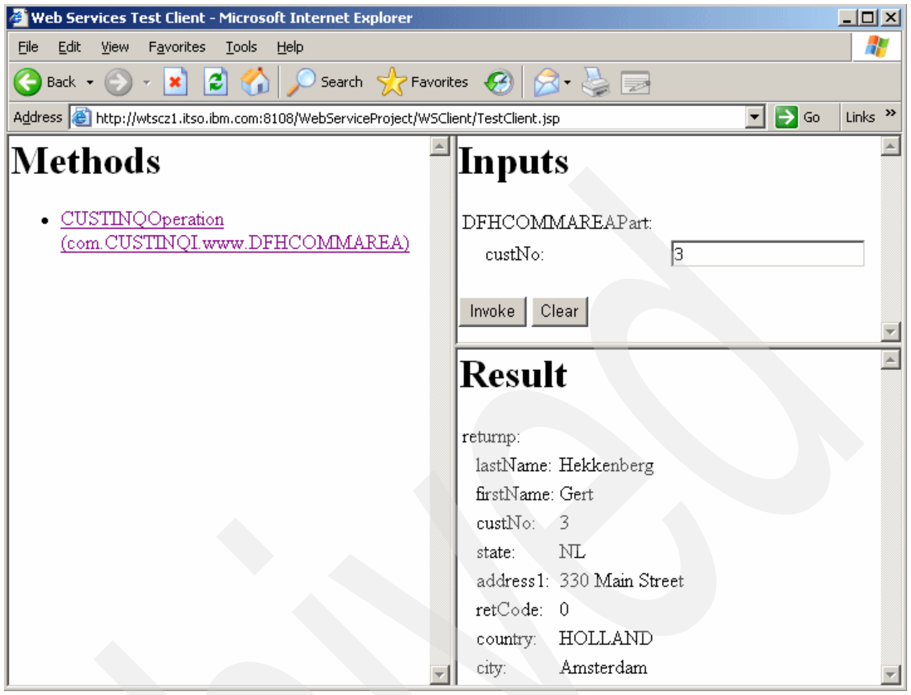
Inputs

DFHCOMMAREAPart:
custNo:

Result

Figure 2-71 Testing application input

3. Input the customer number and click **Invoke.**


Web Services Test Client - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address http://wtsczl.itso.ibm.com:8108/WebServiceProject/WSClient/TestClient.jsp
Go
Links
Invoke
Clear

Methods

- CUSTINQOperation (com.CUSTINQI.www.DFHCOMMAREA)

Inputs

DFHCOMMAREAPart:

custNo: 3

Result

```
returnp:
  lastName:  Hekkenberg
  firstName: Gert
  custNo:    3
  state:     NL
  address1:  330 Main Street
  retCode:   0
  country:   HOLLAND
  city:      Amsterdam
```

Figure 2-72   Testing application results

Check the information about the customer in the Result section.

2.6 Additional material

A WDz Project Interchange file is included with this book. This file contains a WDz workspace with all the components discussed in this chapter. After downloading the file, you can import it into your workbench and look at the various artefacts. If you would adjust those artefacts to the naming conventions of your own environment, you should be able to deploy the solution to your own CICS system.

Refer to Appendix B, "Additional material" on page 397 for instructions on how to download.

3

Accessing DB2 from WebSphere using Web Services

In this chapter we discuss the next generation technology that is used to expose data and stored procedures in DB2 as Web Services.

We will also explain how DB2 can be used as a Web Service consumer using the SOAP User Defined Function (UDF).

IBM Data Server Developer Workbench provides a comprehensive suite of integrated tools for database administrators and application developers. This chapter focuses on the Web Service capabilities built into this product.

This chapter is organized into the following sections.

- IBM Data Server Developer Workbench overview
- Building Web Services:
  - Configuring the workbench for developing and deploying Web services:
    - Creating a database connection
    - Creating a data development project
  - Creating a Web Service

  - Adding database operations to the Web Services:
    - Creating a Web Service operation from SQL script
    - Creating a Web Service operation from a stored procedure
- Deploying Web Services:
  - Automatic WSDL generation and specifying REST-style (HTTP GET/POST binding) and SOAP-style HTTP bindings
  - Web Service artifacts
  - Deploying to z/OS
- Testing:
  - Using the integrated Web Services Explorer for the SOAP-style HTTP binding
  - Using the browser for the REST-style HTTP GET/POST binding
- Web 2.0 features:
  - HTML output via XSLT
  - XMLHTTPRequest (AJAX)
- DB2 as Web Service consumer using the SOAP UDF
- Supplied material

3.1 IBM Data Server Developer Workbench overview

IBM Data Server Developer Workbench is a product in development. At the time of writing, a trial version (Version 9.5) was publicly available at the following URL:

http://www14.software.ibm.com/webapp/download/preconfig.jsp?id=2007-10-30+16%3A22%3A46.718236R&S_TACT=104CBW71&S_CMP=

For data application developers, the workbench provides key features to create Web services that expose database operations (SQL SELECT or calls to stored procedures) to client applications.

Besides the simple exposure of database operations as SOAP Web Service operations, the tooling also provides some interesting features which can be leveraged by Web 2.0[1] by providing REST service bindings and XSL processing capabilities. Operations can be accessed using "plain" HTTP. Table 3-1 shows the supported REST.

Note: This tooling has support for a comprehensive list of databases, Web Servers, and SOAP engines. For the complete list of prerequisites for the supported Web servers, SOAP engines, databases, and message protocols, refer to Appendix A, "Appendix A" on page 391.

Table 3-1 Supported REST bindings

	HTTP GET	**HTTP POST (URL-encoded)**	**HTTP POST (XML)**
Input	Parameter list in URL query string	Parameter list URL-encoded in POST request document	XML
Output	XML, HTML, TEXT	XML, HTML, TEXT	XML, HTML, TEXT

3.2 Building a Web Service

Web Services provides a technology foundation for implementing a Service-Oriented Architecture (SOA). A major focus during the development of this technology is to make the functional building blocks accessible over standard

[1] Web 2.0 is a terminology definition that encapsulates a number of browser-based presentation/data collection technologies such as AJAX, Java Script, RSS, Mashups, XForms, and JSON.

Internet protocols that are independent of platforms and programming languages to ensure that very high levels of interoperability are possible.

Web Services are self-contained software services that can be accessed using simple protocols over a network. Web Services can perform a wide variety of tasks, ranging from simple request-reply to full business process interactions. You can use the workbench to create Web Services that expose database operations (SQL SELECT or calls to stored procedures) to client applications.

Web Services accept XML messages or URL-encoded parameter strings as requests for database operations[2]. For operations that perform SQL inserts, updates, and deletes, Web Services return a count of the number of new, changed, or deleted rows.

The Web Services implementation generated through this tooling performs the following actions when it receives a message that contains a request:

1. Looks in the message to determine the requested database operation.
2. Extracts the input parameters from the message.
3. Prepares and executes the statement.
4. Retrieves the result of the operation.
5. Generates the output XML message that contains the result.
6. Sends the output message to the requesting client application.
7. Optionally transforms the input and output messages if XSL transformations have been configured on the input/output messages.

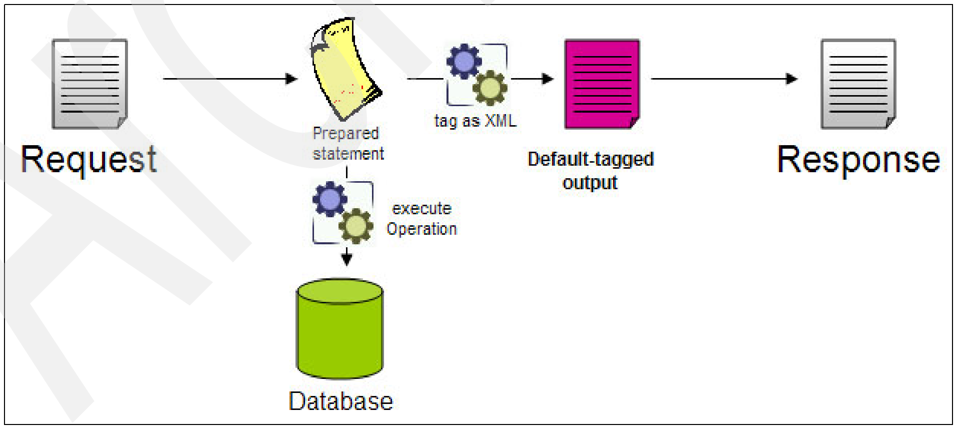


Figure 3-1 Default flow of messages

[2] In the rest of this documentation, XML messages and URL-encoded parameter strings are both referred to as “messages,” except where a distinction between the two is necessary. Web Services returns results and result sets in XML messages.

XML messages that request an operation and XML messages that return the results of an operation are tagged according to a default XML schema that the workbench generates for each operation. This default schema performs the following functions:

- Maps input and output parameters to XML tags.
- Maps columns to XML tags.
- Maps SQL data types to XML data types.
- Maps Web service operations to XML tags.
- Provides the general structure of input and output messages.

In our example we create a Web Service from an SQL script and a stored procedure parameter list. This is a *bottom-up* approach where you design the implementation of SQL and stored procedure calls and use the Web Service wizard to generate the Web service implementation artifacts and WSDL. It may be faster and no XSD or WSDL design skills are needed. However, if complex objects (for example, Java collection types) are used then the resulting WSDL may be hard to understand and less interoperable.

3.2.1 Configuring the Workbench for developing and deploying Web Services

Before you can develop Web Services, you must create a data development project and associate that project with a supported database.

Creating a database connection

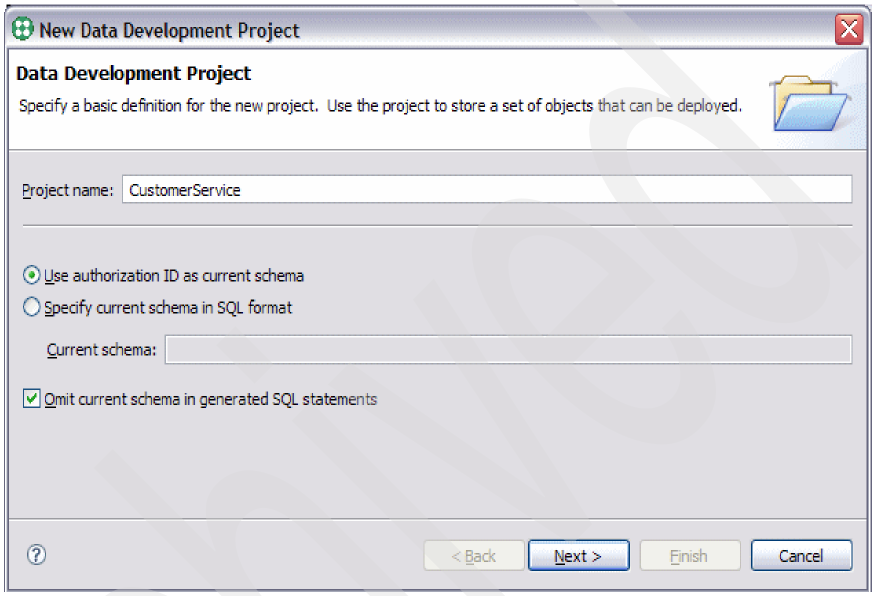
First you need to connect to the database that you will be working with. To connect to this database, use the wizard, as follows:

1. Right-click in the Database Explorer view, and select **New Connection** from the pop-up menu.
2. On the first page of the wizard, select a database manager, a JDBC™ driver, and specify other connection details. See one of the related topics for specific information about the database that you want to connect to.
3. Optionally, on the “Limit the objects retrieved from the database” page, specify filtering options. For best performance, you should use filters when you are connecting to a large database.
4. Complete all other wizard steps and click **Finish**.

Creating a Data Development Project[3]

Proceed as follows to create a Data Development Project:

1. Right-click in the Data Project Explorer view, and select **New** → **Data Development Project**. Enter the project name and accept the defaults as shown in Figure 3-2.



Specify a basic definition for the new project. Use the project to store a set of objects that can be deployed.

Project name: CustomerService

(●) Use authorization ID as current schema
( ) Specify current schema in SQL format

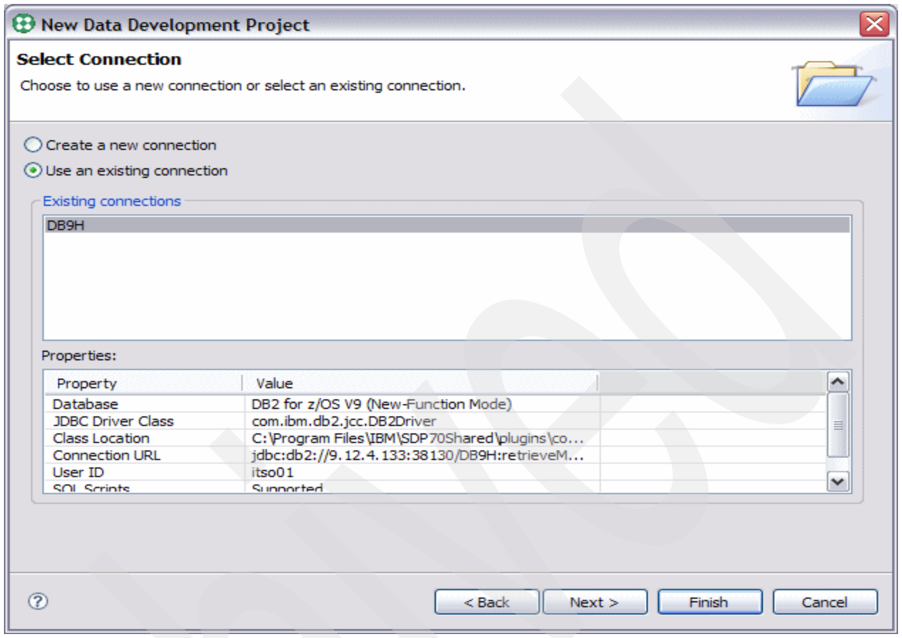
Current schema:

[✓] Omit current schema in generated SQL statements

Figure 3-2 New Data Development project

[3] This step can also be accomplished using the "Create Web Service wizard"

2. Choose **Use an existing connection** as shown in Figure 3-3.
3. Click **Finish**.


New Data Development Project

Select Connection

Choose to use a new connection or select an existing connection.

Create a new connection [✓ Use an existing connection]

Existing connections

DB9H

Properties:

Property	Value
Database	DB2 for z/OS V9 (New-Function Mode)
JDBC Driver Class	com.ibm.db2.jcc.DB2Driver
Class Location	C:\Program Files\IBM\SDP70Shared\plugins\co...
Connection URL	jdbc:db2://9.12.4.133:38130/DB9H:retrieveM...
User ID	itso01
SQL Scripts	Supported

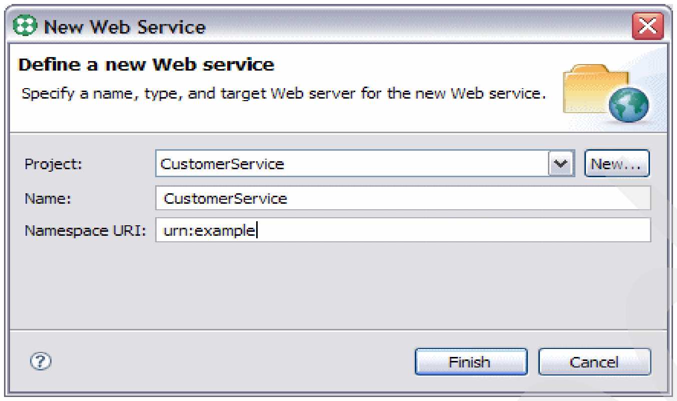
Figure 3-3 Choose existing connection

3.2.2 Creating a Web Service

When you create a Web Service, you create an empty container that you subsequently add database operations to. To define a Web Service, right-click on the **Web services** folder and select **New Web Service**. The New Web Service wizard opens.

Name the Web service “CustomerService” and click **Finish**. The service is displayed in the Web Service folder in the Data Development Project.

Note: Optionally, you can change the default namespace URI. For example, you might want to use a namespace URI to group Web services that have similar functions, have one owner, or access the same data source. For this example we will use the default urn:example.


New Web Service

Define a new Web service

Specify a name, type, and target Web server for the new Web service.

Project: CustomerService

Name: CustomerService

Namespace URI: urn:example

Figure 3-4 New Web Service dialog

Note: The name specified for the Web Service will be used as the name for the automatically generated Web project during the build and deploy step discussed in 3.3, “Deploying a Web Service” on page 100.

3.2.3 Adding database operations to the Web Service

To add SQL operations[4] to Web Services, you can create new SQL scripts in the New Operation window or drag and drop existing scripts that are in a data development project onto a Web Service.

To add stored procedures options to Web Services, you can create new stored procedure CALL statements in the “new Operation” wizard or you may drag and drop existing stored procedures from the Database Explorer.

Note: You can include one SQL statement per operation.

In the SQL statements, you can use all data types that are supported by JDBC 3.0, except for ARRAY, DISTINCT, JAVA_OBJECT, OTHER, REF, and STRUCT. You can also use DB2's XML data type and the ROWID data type that DB2 for z/OS supports.

Use named parameters or positional parameters in WHERE clauses and statements that accept values. There are two advantages to using named parameters:

[4] SELECT, INSERT, UPDATE, DELETE SQL statements

- Names make the default XML schema more verbose and can describe the meaning of the parameter to the client applications.
- You can assign a parameter to more than one host variable, which is useful for UNION ALL views.

Named parameters begin with a colon and are typically named after their corresponding host variable, as in this example where :custno is the parameter and custno is the host variable:

```
SELECT * FROM customer where custno=:custno
```

In the resulting XML schema, the entry for the parameter might look like this:

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

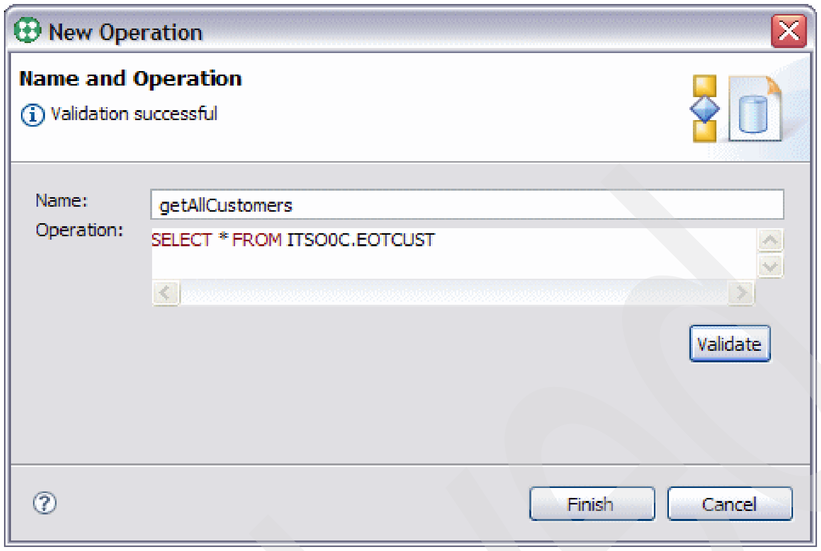
The names are case-sensitive and must be valid XML tag names. If the name of a parameter contains characters or character sequences that are not legal in XML, the workbench automatically applies SQL/XML escaping rules to the name. For example, a parameter named xml is transformed to _xFFFF_xml. So, if you were to use this parameter as an input parameter in an HTTP GET request, the URL would have to look similar to this example:

```
http://localhost:8080/ContextRoot/rest/MyService?_xFFFF_xml=1234
```

Creating a Web Service operation from an SQL script

You can create operations that run any SQL statement that is supported by your database. Create a new SQL script in the New operation wizard. In the Data Project Explorer, right-click the Web service that you want to add the operation to. Select **New Operation**. Opening the wizard this way lets you add the operation to one Web service. Enter the name for the operation and the SQL statement as shown in Figure 3-5 on page 94.

If needed, validate the SQL statement. Then click **Finish**.


New Operation

Name and Operation

Validation successful

Name: getAllCustomers

Operation:
```
SELECT * FROM ITSO0C.EOTCUST
```

Figure 3-5 New Operation dialog

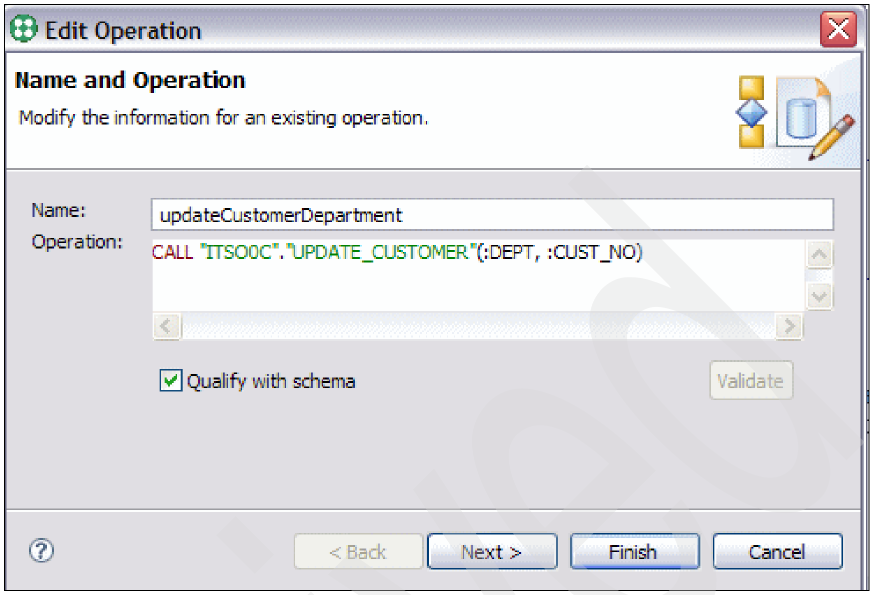
Creating a Web service operation from a stored procedure

You can create operations that call any stored procedure that is deployed in your database. The stored procedure must be deployed on the database that is associated with the Web Service in which you want to create the operation. You can include one stored procedure call per operation. For this example, we will use an existing stored procedure deployed on our database called UPDATE_CUSTOMER.

Note: For a stored procedure call that returns result sets you may want to use the verbose version of the stored procedure metadata you need to execute the stored procedure to allow tooling to retrieve the appropriate result set metadata first. Otherwise the generated schema will only contain a common anonymous result set schema definition.

Steps:

1. Drag the stored procedure from the Database Explorer and drop it onto the Web Service in the Data Project Explorer. Right-click and choose **Edit**. Change the operation name. The operation name defaults to the stored procedure name. In this example we change the operation name to updateCustomerDepartment, as shown in Figure 3-6 on page 95.



Name: updateCustomerDepartment

Operation: CALL "ITSO0C"."UPDATE_CUSTOMER"(:DEPT, :CUST_NO)

✓ Qualify with schema

Figure 3-6   Edit operation to change the Operation name of the stored procedure.

2. Clicking **Next** invokes a panel that provides options for specifying constant values, if any, for the inputs and outputs. For this example, we do not have any constant value and we click **Finish**.

Finished! We have created a new operation in the Web service that calls a stored procedure.

Stored procedures that return result sets

In case the operation is a stored procedure call that returns result sets, you may want to use the verbose version of the stored procedure metadata you need to execute the stored procedure to allow tooling to retrieve the appropriate result set metadata first. Otherwise the generated schema will only contain a common anonymous result set schema definition.

Since the DB2 catalog does not contain metadata information about result sets a stored procedure may return, and the result sets may vary depending on the input, some extra care might be needed to make the generated XML schema in the WSDL document more meaningful for clients to consume.

We explain two approaches to handle this scenario based on the stored procedure behavior:

- The dynamic approach

  When dragging and dropping a stored procedure onto a Web Service, the DB2 catalog does not tell us how many result sets are really returned (and in DB2 even the number of result sets returned on every call might vary) nor do we get the result set metadata information for the result sets. To be able to create an XML schema for the output message representing this kind of result set, the schema is generated with an anonymous result set type that describes a generic result set. For our updateCustomerDepartment operation, we see that this is the case shown in Example 3-1[5].

  Example 3-1 Anonymous result set type

```
<element name="updateCustomerDepartmentResponse">
<complexType>
<sequence>
<element maxOccurs="1" minOccurs="0" name="rowset"
type="tns:anonymousResultSetType"/>
</sequence>
</complexType>
</element>
```

  In the response message XML schema definition, the tooling will add as many references to this anonymous result set type as the maximum number of result sets the stored procedure might return. For example, the schema for a stored procedure that returns at most two result sets, the output message element definition might look like Example 3-2.

  Example 3-2 Schema definition for two result sets

```
<xsd:element name="two_result_sets_anonymousResponse">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element maxOccurs="1" minOccurs="0" name="rowset"
type="tns:anonymousResultSetType" />
         <xsd:element maxOccurs="1" minOccurs="0" name="rowset2"
type="tns:anonymousResultSetType" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

  Even if this approach works for all kinds of stored procedures it has some drawbacks. The usage of the xsd:any element specification and not knowing the column names and data types make it difficult for Web service client

[5] The WSDL and schema referred here is created during the deploy step.

frameworks (which consume the WSDL document) to create Web service client code and may cause other interoperability issues.

- The static approach

  With the static result set approach we provide a way to make the WSDL document more verbose by including the actual result set metadata into the XML schema. This approach requires some additional user input. It also requires that the stored procedure return the same number of result sets in the same order on any request (therefore we call it the static approach). The tooling will execute the stored procedure and use the metadata of the returned result sets in order to generate a more verbose XML schema document.

  The stored procedure behind our updateCustomerDepartment operation can be considered static. We will show how to make this response element in this schema more meaningful by adding result set metadata to the XML schema than the one shown in Example 3-1 on page 96.

  The existing stored procedure does not return any value. We will modify it to return a SUCCESS string; see Example 3-3.

Example 3-3 Stored procedure

```
CREATE PROCEDURE UPDATE_CUSTOMER ( IN DEPT CHAR(8),
                                   IN CUST_NO INTEGER,
                                   OUT RESULT CHAR(20) )
VERSION VERSION1
   WLM ENVIRONMENT FOR DEBUG MODE DB9HWLM4 ALLOW DEBUG MODE
ISOLATION LEVEL CS
   RESULT SETS 1
   LANGUAGE SQL
-
P1: BEGIN
-- Declare variable
   DECLARE RESULT_TMP CHAR(20) DEFAULT 'SUCCESS';
-- Declare result set cursor
    DECLARE eotCustResult CURSOR WITH RETURN TO CALLER FOR
          SELECT * FROM ITSOOC.EOTCUST;

   UPDATE ITSOOC.EOTCUST
     SET DEPT = UPDATE_CUSTOMER.DEPT
     WHERE CUST_NO = UPDATE_CUSTOMER.CUST_NO;
    SET RESULT = RESULT_TMP;

   OPEN eotCustResult;

   RETURN 0;
```

```
END P1
```

After deploying this stored procedure to your database, you must delete the existing updateCustomerDepartment operation and drag and drop the modified stored procedure again to recreate the operation.

Right-click the Web service operation that makes the stored procedure call and select **Edit...** (updateCustomerDepartment). Now, click **Next** in the Edit Operation window[6]. This will bring you to the Generate XML Schema for Stored Procedure page.

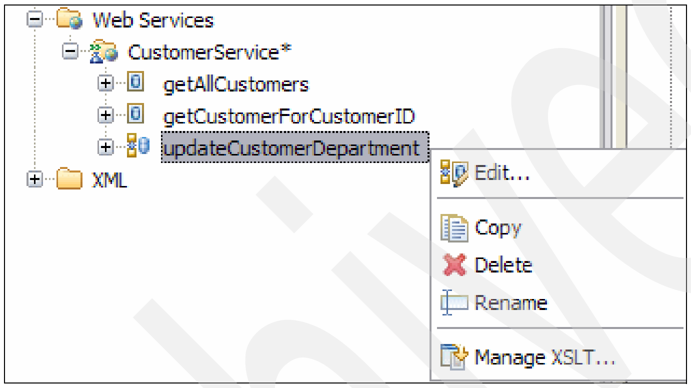


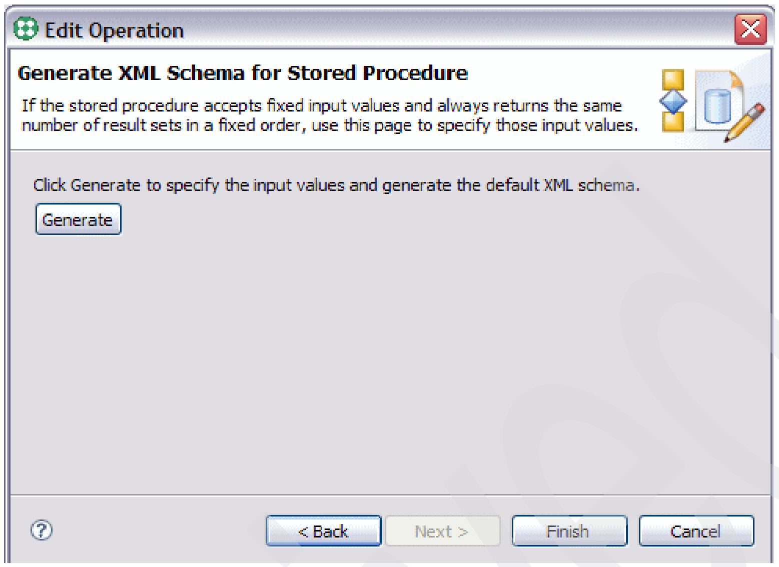
Figure 3-7 Editing the operation to create a more meaningful schema

Click **Generate** in Figure 3-8 on page 99. Supply the parameters as shown in Figure 3-9 or according to your data availability. Click **Ok** and finish in the subsequent window. After the deployment, you will see the Response schema that is generated, as shown in Example 3-4.

If you had not clicked **Generate** in Figure 3-8 on page 99 and proceeded to deployment, you would have the schema that is generated as shown in Example 3-5 after the build and deploy step.

The material in this chapter refers to the scenario where we did not use the **Generate** button.

[6] Notice that the SQL has changed to include the additional parameter RESULT that was added to the callout of the stored procedure: `CALL "ITSOOC"."UPDATE_CUSTOMER"(:DEPT, :CUST_NO, :RESULT)`

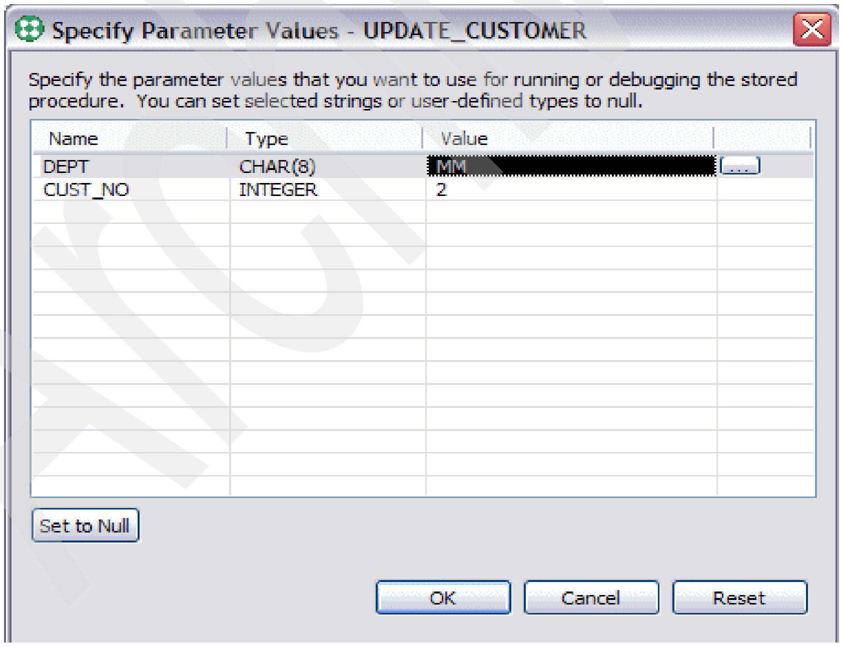

Edit Operation

Generate XML Schema for Stored Procedure

If the stored procedure accepts fixed input values and always returns the same number of result sets in a fixed order, use this page to specify those input values.

Click Generate to specify the input values and generate the default XML schema.

Figure 3-8   Clicking Generate to simulate the stored procedure call


Specify Parameter Values - UPDATE_CUSTOMER

Specify the parameter values that you want to use for running or debugging the stored procedure. You can set selected strings or user-defined types to null.

Name	Type	Value
DEPT	CHAR(8)	MM
CUST_NO	INTEGER	2

Figure 3-9   Simulate the run

Example 3-4 Schema when using Generate

```
<element name="updateCustomerDepartmentResponse">
<complexType>
<sequence>
<element name="RESULT" nillable="true" type="xsd:string"/>
<element name="rowset">
<complexType>
<sequence>
<element maxOccurs="unbounded" minOccurs="0" name="row">
<complexType>
<sequence/>
</complexType>
</element>
</sequence>
</complexType>
</element>
</sequence>
</complexType>
</element>
```

Example 3-5 Schema when not using Generate

```
<element name="updateCustomerDepartmentResponse">
<complexType>
<sequence>
<element name="RESULT" nillable="true" type="xsd:string"/>
<element maxOccurs="1" minOccurs="0" name="rowset"
type="tns:anonymousResultSetType"/>
</sequence>
</complexType>
</element>
```

3.3 Deploying a Web Service

We will walk through the steps involved in generating the WSDL and Web Archive (WAR) file and deploying them to WebSphere Application Server. To generate the WAR file, do the following:

1. Right-click the Web Service and select **Deploy**.

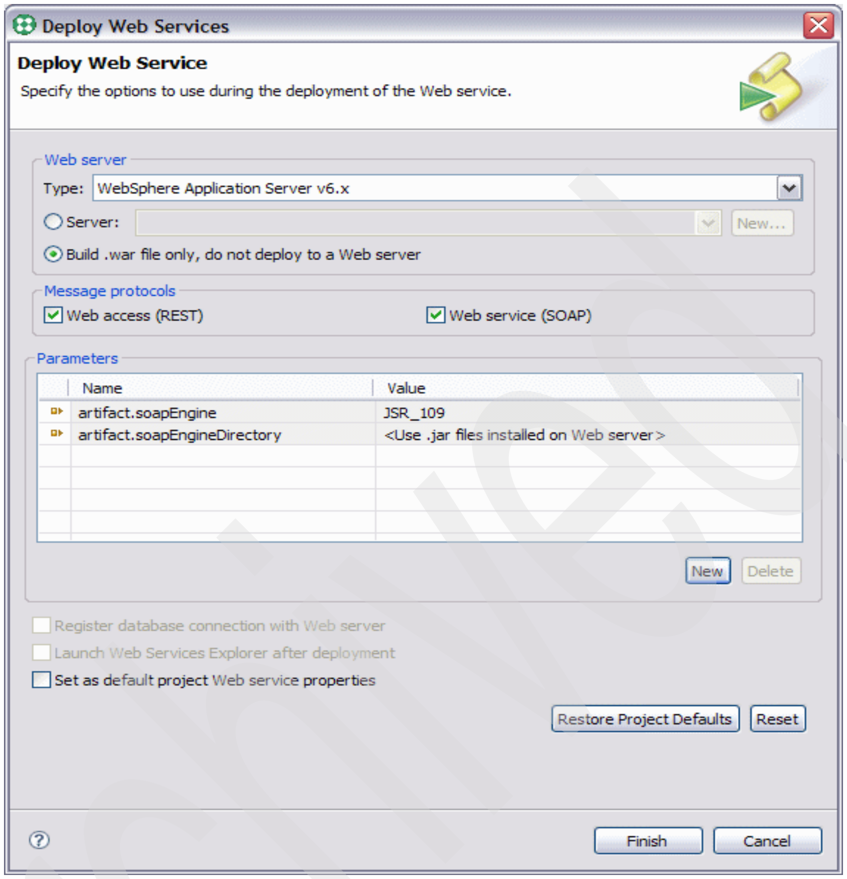
2. In the section under Web Server, choose the **Build .war file only, do not deploy to a Web server** option.
3. In the Message protocols section, select one or both message protocols for the messages that the Web Service will support:
   - Select **Web access (REST)** if one or more operations in the Web Service will be accessed by messages of either of the following types:
     - HTTP POST (text/xml)
     - HTTP GET/POST (url-encoded)
   - Select **Web service (SOAP)** if one or more of the operations in the Web Service will be accessed by messages that are wrapped in SOAP envelopes.
4. Specify values for parameters that are specific to the selected Web Application Server, as shown in Table 3-2. For this example, we will be using WebSphere Application Server Version 6.1 on z/OS, and these parameters are specific to this Web Application Server.

Table 3-2 Deployment parameters

Parameter Name	Description	Value Used
artifact.soapEngine	Specifies the SOAP engine to use, if you selected Web service (SOAP) under Message protocols	JSR_109
artifact.soapEngineDirectory	Optional parameter that specifies the directory on the local file system with the JAR files for the SOAP engine.	<Use.jar files installed on Web server>

Important: Make sure to select WebSphere Application Server V6.x as Web Server Type, as shown in Figure 3-10 on page 102.

5. Click **Finish**. You have successfully generated a Web WAR module that can be deployed in the target Web Application Server.


Deploy Web Services

Deploy Web Service

Specify the options to use during the deployment of the Web service.

Web server

Type: WebSphere Application Server v6.x

○ Server:

[✓ Build .war file only, do not deploy to a Web server]

Message protocols

☑ Web access (REST) ☑ Web service (SOAP)

Parameters

Name	Value
artifact.soapEngine	JSR_109
artifact.soapEngineDirectory	<Use .jar files installed on Web server>

☐ Register database connection with Web server

☐ Launch Web Services Explorer after deployment

☐ Set as default project Web service properties

Figure 3-10 Deploy options

3.3.1 Web Service artifacts

After the wizard completes, switch to the Java or J2EE perspective and you will see a Web project generated. Figure 3-11 on page 103 shows the various artifacts generated by this tooling.

- The WSDL file is in the WEB-INF folder. This WSDL is used by the server for deployment purposes, and is accessible to external clients through HTTP. For example, if the server is running on port 8108 and the server address is wtscz1.itso.ibm.com, you can retrieve the WSDL at:

  `http://wtscz1.itso.ibm.com:8108/CustomerService/wsdl`

Note: In general, you can use the URL

`http://<server>:<port>/<contextRoot>/wsdl`

to get to the WSDL from the browser, or as input to the Web Service Explorer.

- *dswsRuntime.jar* represents the runtime classes that will be used to accomplish the tasks specified in 3.2, "Building a Web Service" on page 87.
- The JAX-RPC type mapping file contains information on the relationships between data types used in SQL and their equivalents in XML.
- The Web Services deployment descriptor contains information used by the server to deploy the Web Services in the project. The format of this file is defined by the Web Services for J2EE specification.

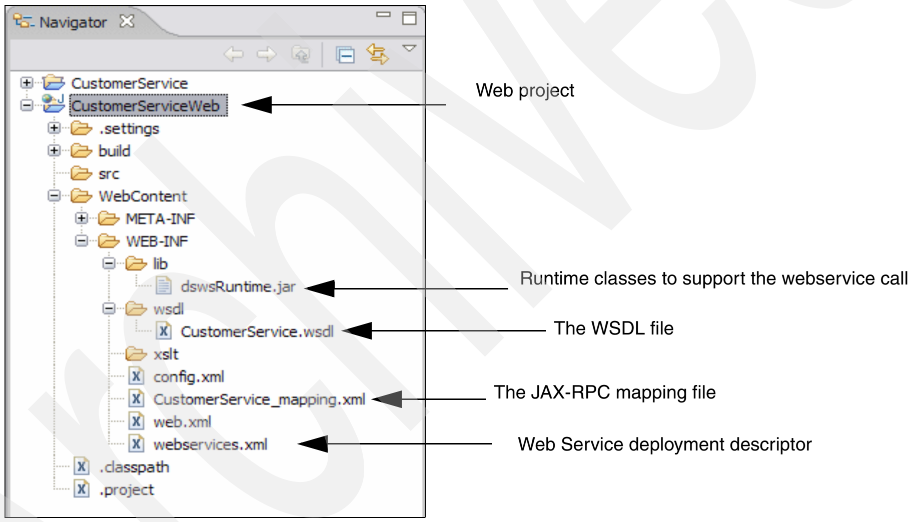


Figure 3-11 Web Services generated resources

The WAR file is generated in the following location:

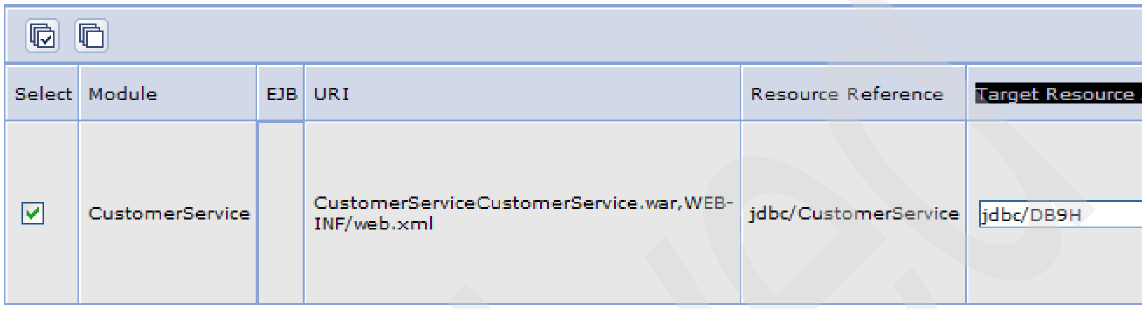
```
<workspace_directory>\<project_directory>\DataServerWebServices\<Web
service name[7]>
```

Deploying to WebSphere on z/OS

We will use the WebSphere Application Server Admin console to deploy this WAR file. Log on to the console. Expand **Applications** on the left pane and

[7] This is the name that was specified in Figure 3-4

choose **Install New Application**. Specify the location of your WAR file and make sure you specify the Context root as CustomerService. The value for the Context root must later be used as a qualifier in the URL to address the Web Service, so you have to make sure you remember it. Make sure you map the JDBC resource reference to the target resource JNDI name for the DB2 database to be used. See Figure 3-12[8]. If you encounter issues deploying, refer to the product documentation for any additional steps to deploy this Web service.



Select	Module	EJB	URI	Resource Reference	Target Resource
✓	CustomerService		CustomerServiceCustomerService.war,WEB-INF/web.xml	jdbc/CustomerService	jdbc/DB9H

Figure 3-12 Associating the Resource Reference to the data source

Accept the defaults for the remainder of the steps, and you are done deploying.

3.4 Testing a Web Service

We will now discuss different ways of testing the Web Service:

- Using the Web Services Explorer in the tooling and SOAP-style bindings, discussed in 3.4.1, "Using the integrated Web Services Explorer and SOAP-style bindings".
- Using a browser, HTTP and REST-style bindings, discussed in 3.4.2, "Using the browser to test the REST-style HTTP GET/POST binding" on page 109.

3.4.1 Using the integrated Web Services Explorer and SOAP-style bindings

Proceed as follows:

- Right-click the WSDL and invoke the Web Services Explorer menu shown in Figure 3-13.

[8] You will notice that the tooling has generated with a resource reference to access the database. Associate that resource reference with the data source configured inside WebSphere or your favorite App server.

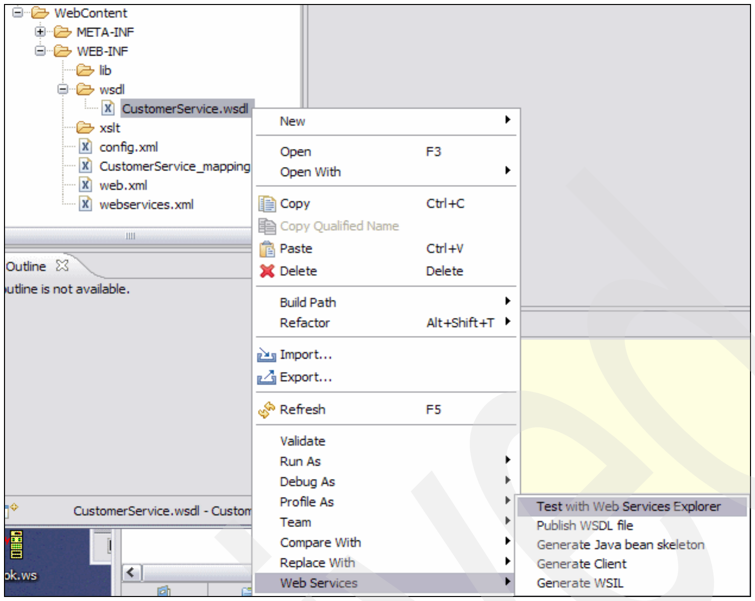
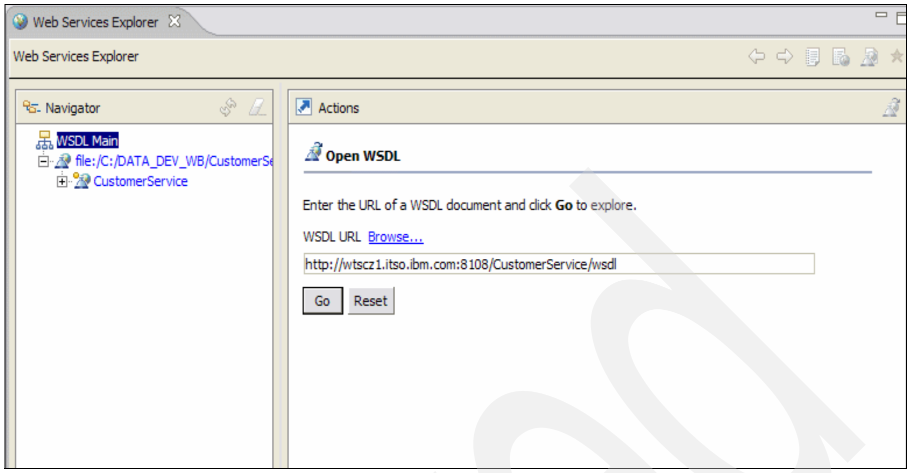


Figure 3-13 Invoking the built-in Web Services Explorer

- In the Web Services Explorer window, choose the WSDL main node at the root and enter the end point. We will not be testing this service running in a local environment. So for this example we use our z/OS-deployed environment end point, as shown in Figure 3-14:

```
http://wtscz1.itso.ibm.com:8108/CustomerService/wsdl
```

- The next window shows the Web Service bindings; see Figure 3-15 on page 106.
- Next we invoke the Web Service SOAP operation that will execute the SQL by choosing **CustomerServiceSOAP** → **getAllCustomers** and click **Go**.


Web Services Explorer

Navigator

WSDL Main
file:/C:/DATA_DEV_WB/CustomerSe
CustomerService

Actions

Open WSDL

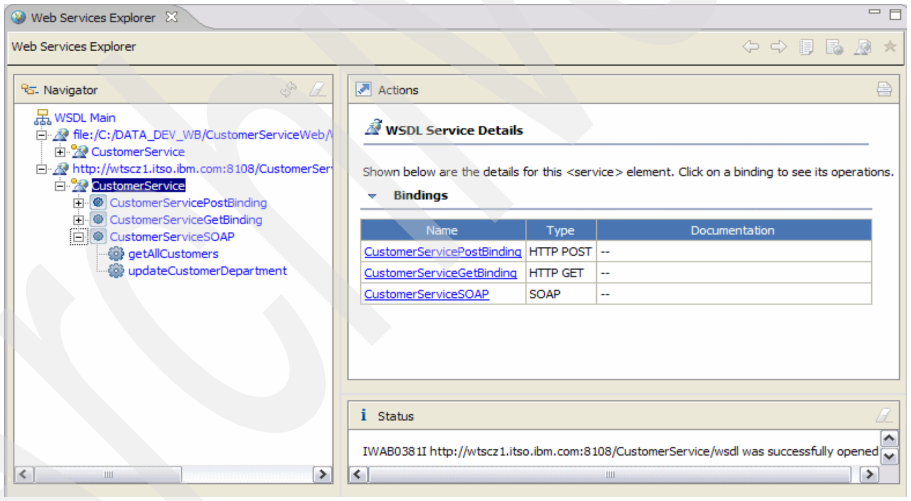
Enter the URL of a WSDL document and click **Go** to explore.

WSDL URL Browse...

http://wtscz1.itso.ibm.com:8108/CustomerService/wsdl

Go Reset

Figure 3-14 Setting the deployed end point


Web Services Explorer

Navigator

WSDL Main
file:/C:/DATA_DEV_WB/CustomerServiceWeb/
CustomerService
http://wtscz1.itso.ibm.com:8108/CustomerSer
CustomerService
CustomerServicePostBinding
CustomerServiceGetBinding
CustomerServiceSOAP
getAllCustomers
updateCustomerDepartment

Actions

WSDL Service Details

Shown below are the details for this <service> element. Click on a binding to see its operations.

Bindings

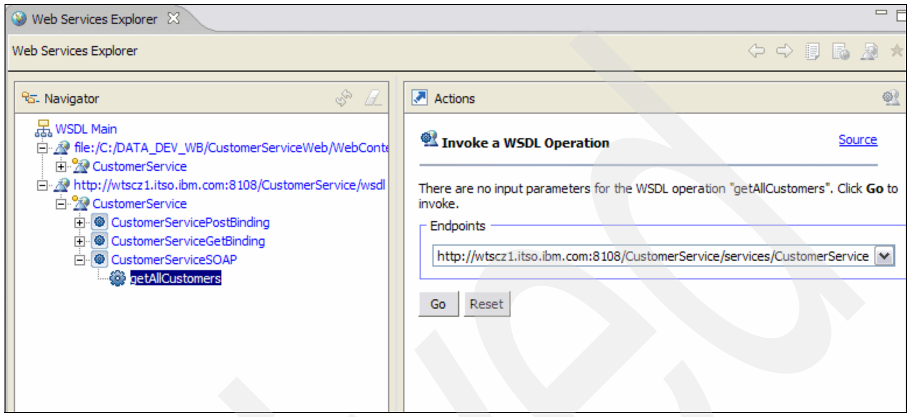
Name	Type	Documentation
CustomerServicePostBinding	HTTP POST	--
CustomerServiceGetBinding	HTTP GET	--
CustomerServiceSOAP	SOAP	--

Status

IWAB0381I http://wtscz1.itso.ibm.com:8108/CustomerService/wsdl was successfully opened

Figure 3-15 Viewing the Web Services bindings

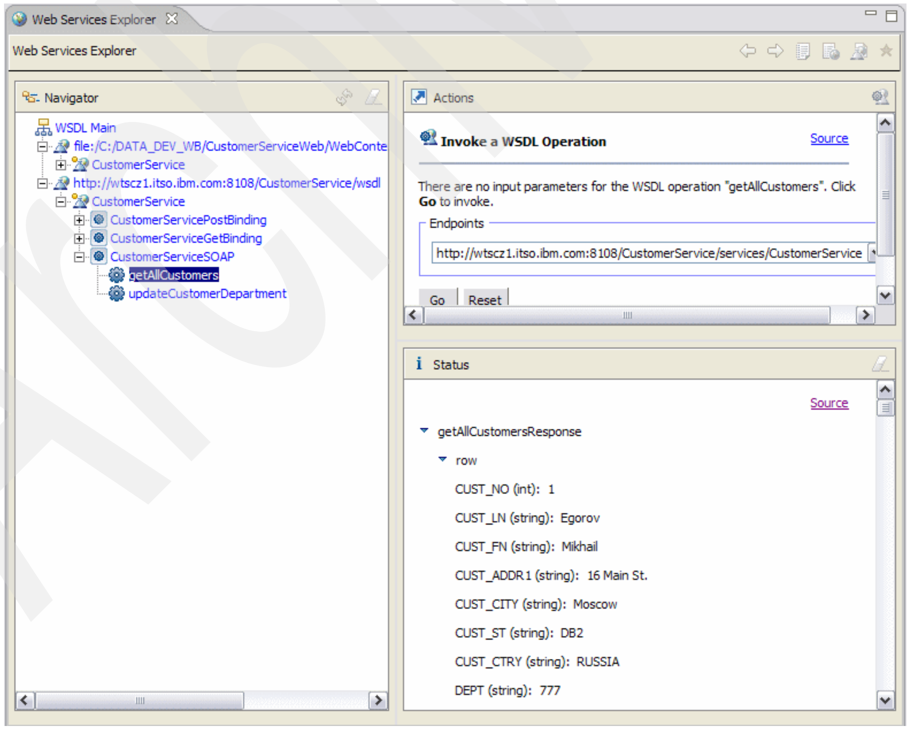
- Figure 3-17 shows the results in the Results pane. You may choose to view the results in the Source form or Form form. You can double-click the status pane to maximize and view the complete results. Choosing the Source form will let you see the SOAP envelope XML message.



There are no input parameters for the WSDL operation "getAllCustomers". Click **Go** to invoke.

http://wtscz1.itso.ibm.com:8108/CustomerService/services/CustomerService

Figure 3-16 Invoking the operation



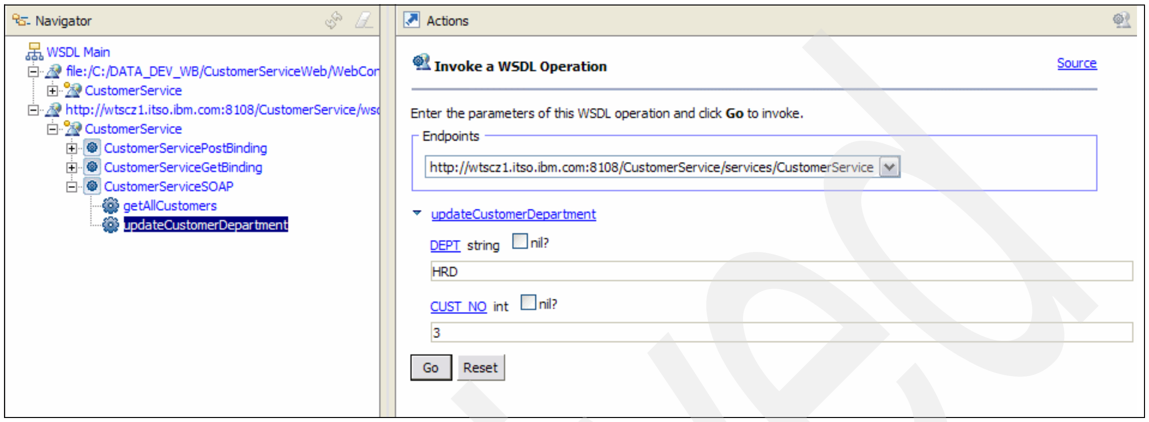
There are no input parameters for the WSDL operation "getAllCustomers". Click **Go** to invoke.

http://wtscz1.itso.ibm.com:8108/CustomerService/services/CustomerService

```
getAllCustomersResponse
  row
    CUST_NO (int): 1
    CUST_LN (string): Egorov
    CUST_FN (string): Mikhail
    CUST_ADDR1 (string): 16 Main St.
    CUST_CITY (string): Moscow
    CUST_ST (string): DB2
    CUST_CTRY (string): RUSSIA
    DEPT (string): 777
```

Figure 3-17 Web Service results

- Next we invoke the operation that will invoke the stored procedure (updateCustomerDepartment). Note that this operation takes in a Department number and customer ID as inputs. For this example, we use HRD and 3 as input values and click **Go**; see Figure 3-18.



Enter the parameters of this WSDL operation and click **Go** to invoke.

Endpoints

http://wtscz1.itso.ibm.com:8108/CustomerService/services/CustomerService

updateCustomerDepartment

DEPT string ☐ nil?

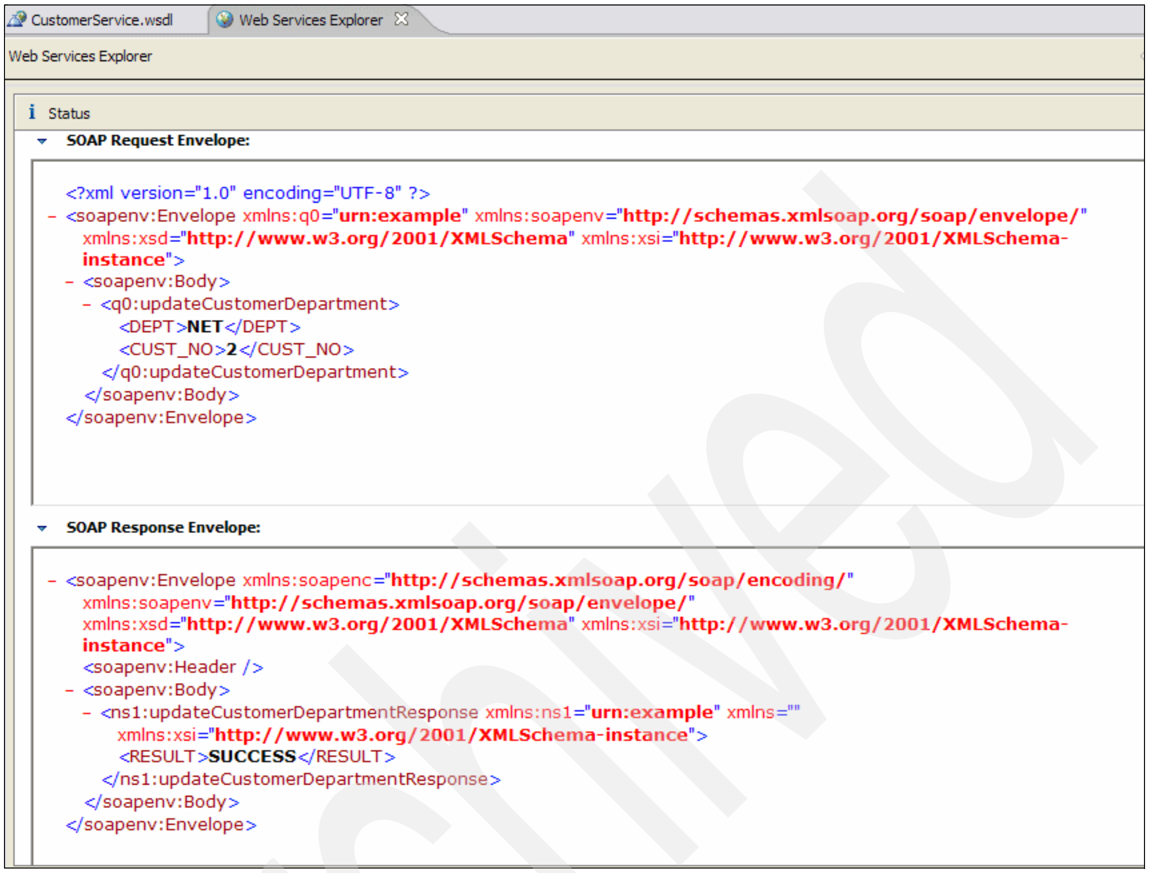
HRD

CUST_NO int ☐ nil?

3

Figure 3-18   Invoking the stored procedure operation

- The source message and response message are captured in Figure 3-19 on page 109.



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <soapenv:Envelope xmlns:q0="urn:example" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance">
  - <soapenv:Body>
    - <q0:updateCustomerDepartment>
        <DEPT>NET</DEPT>
        <CUST_NO>2</CUST_NO>
      </q0:updateCustomerDepartment>
    </soapenv:Body>
  </soapenv:Envelope>
```

```
- <soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    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>
    - <ns1:updateCustomerDepartmentResponse xmlns:ns1="urn:example" xmlns=""
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <RESULT>SUCCESS</RESULT>
      </ns1:updateCustomerDepartmentResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Figure 3-19 Results displayed in the source form with expanded status pane

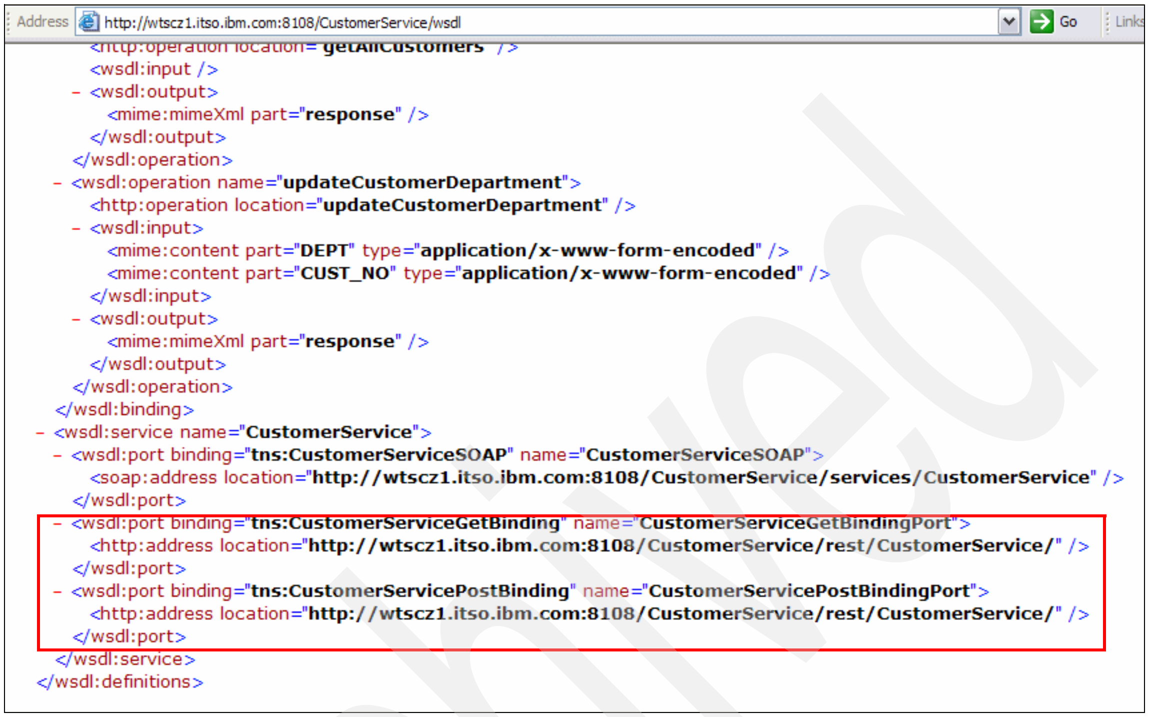
Note: You see the value SUCCESS in the response because the stored procedure had returned an output parameter that had a value SUCCESS on a successful update.

3.4.2 Using the browser to test the REST-style HTTP GET/POST binding

You may use the integrated Web Services Explorer or the browser to test the REST-style binding. In this section we will show how to test the REST bindings with the browser. Retrieve the WSDL at:

```
http://<your server>:<portnumber>/CustomerService/wsdl
```

and locate, in the WSDL, the bindings for REST in the Web Service definition section, as shown in Figure 3-20.


Address http://wtscz1.itso.ibm.com:8108/CustomerService/wsdl Go Links

```
      <http:operation location="getAllCustomers" />
      <wsdl:input />
    - <wsdl:output>
        <mime:mimeXml part="response" />
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="updateCustomerDepartment">
      <http:operation location="updateCustomerDepartment" />
    - <wsdl:input>
        <mime:content part="DEPT" type="application/x-www-form-encoded" />
        <mime:content part="CUST_NO" type="application/x-www-form-encoded" />
      </wsdl:input>
    - <wsdl:output>
        <mime:mimeXml part="response" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="CustomerService">
  - <wsdl:port binding="tns:CustomerServiceSOAP" name="CustomerServiceSOAP">
      <soap:address location="http://wtscz1.itso.ibm.com:8108/CustomerService/services/CustomerService" />
    </wsdl:port>
  - <wsdl:port binding="tns:CustomerServiceGetBinding" name="CustomerServiceGetBindingPort">
      <http:address location="http://wtscz1.itso.ibm.com:8108/CustomerService/rest/CustomerService/" />
    </wsdl:port>
  - <wsdl:port binding="tns:CustomerServicePostBinding" name="CustomerServicePostBindingPort">
      <http:address location="http://wtscz1.itso.ibm.com:8108/CustomerService/rest/CustomerService/" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Figure 3-20 REST bindings in the generated WSDL

Capture the URL from the http:address location tag. In our example, this is:

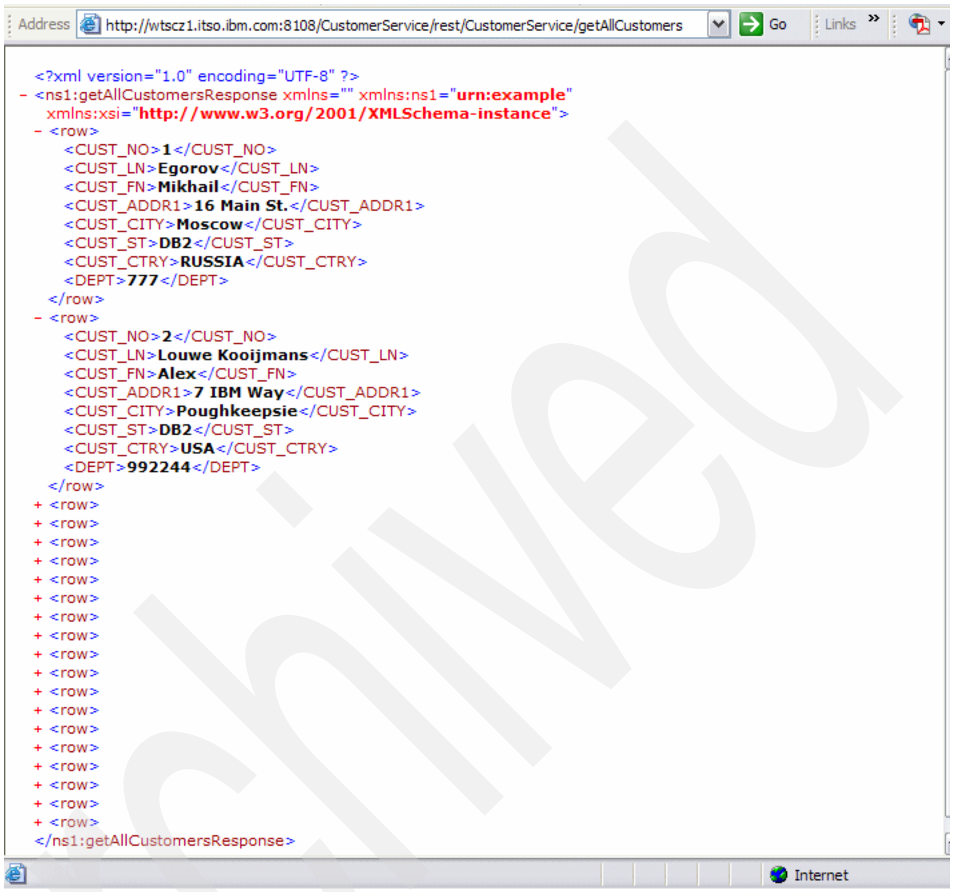
`http://wtscz1.itso.ibm.com:8108/CustomerService/rest/CustomerService/`

To invoke the operation, append the operation name to the above URL. In this example, this will result in the following URL for invoking the `getAllCustomers` operation:

```
http://wtscz1.itso.ibm.com:8108/CustomerService/rest/CustomerService/ge
tAllCustomers
```

Note: The `getAllCustomers` operation does not take any input. Hence there is no variation in the URL for invoking the GET and POST style REST bindings.

Enter this value in the browser. The results should look as shown in Figure 3-21.



Address http://wtscz1.itso.ibm.com:8108/CustomerService/rest/CustomerService/getAllCustomers Go Links

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <ns1:getAllCustomersResponse xmlns="" xmlns:ns1="urn:example"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <row>
      <CUST_NO>1</CUST_NO>
      <CUST_LN>Egorov</CUST_LN>
      <CUST_FN>Mikhail</CUST_FN>
      <CUST_ADDR1>16 Main St.</CUST_ADDR1>
      <CUST_CITY>Moscow</CUST_CITY>
      <CUST_ST>DB2</CUST_ST>
      <CUST_CTRY>RUSSIA</CUST_CTRY>
      <DEPT>777</DEPT>
    </row>
  - <row>
      <CUST_NO>2</CUST_NO>
      <CUST_LN>Louwe Kooijmans</CUST_LN>
      <CUST_FN>Alex</CUST_FN>
      <CUST_ADDR1>7 IBM Way</CUST_ADDR1>
      <CUST_CITY>Poughkeepsie</CUST_CITY>
      <CUST_ST>DB2</CUST_ST>
      <CUST_CTRY>USA</CUST_CTRY>
      <DEPT>992244</DEPT>
    </row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  + <row>
  </ns1:getAllCustomersResponse>
```

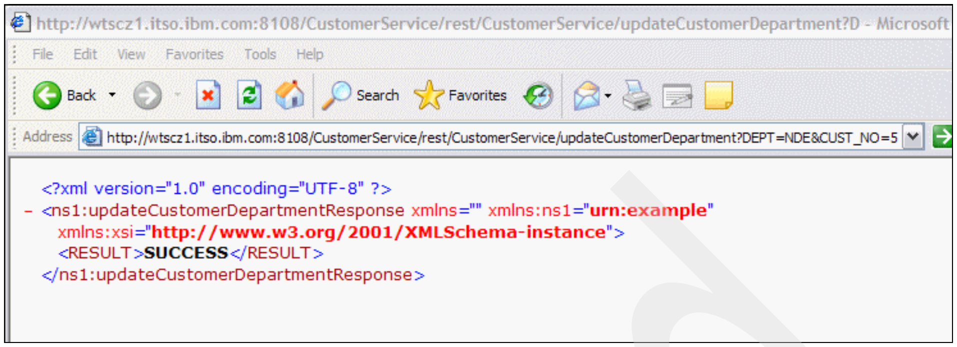
Internet

Figure 3-21 Results from invoking the REST-style HTTP GET/POST getAllCustomers operation

For the `updateCustomerDepartment` operation, to invoke the GET bindings, the URL for our example will appear like this:

```
http://wtscz1.itso.ibm.com:8108/CustomerService/rest/CustomerService/up
dateCustomerDepartment?DEPT=NDE&CUST_NO=3
```

The results should look as shown in Figure 3-22 on page 112.


http://wtscz1.itso.ibm.com:8108/CustomerService/rest/CustomerService/updateCustomerDepartment?D - Microsoft
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://wtscz1.itso.ibm.com:8108/CustomerService/rest/CustomerService/updateCustomerDepartment?DEPT=NDE&CUST_NO=5

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <ns1:updateCustomerDepartmentResponse xmlns="" xmlns:ns1="urn:example"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <RESULT>SUCCESS</RESULT>
  </ns1:updateCustomerDepartmentResponse>
```

Figure 3-22 REST-style Update operation call result in the browser

For the `updateCustomerDepartment` operation, to invoke the HTTP POST bindings where parameters are URL-encoded inside the POST request, we will leverage use the following HTML form. See Example 3-6.

Example 3-6 Sample form to submit HTTP POST request

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
      <title>Update Customer</title>
   </head>
   <body>
      <form
action="http://wtscz1.itso.ibm.com:8108/CustomerService/rest/CustomerSe
rvice/updateCustomerDepartment" accept-charset="utf-8"
enctype="application/x-www-form-urlencoded" method="POST">
Customer Number: <input name="CUST_NO" id="CUST_NO" />
New Department: <input name="DEPT" id="DEPT" />
             <input id="submit" value="submit" type="submit" />
      </form>
   </body>
</html>
```

This form shows in the browser as shown in Figure 3-23. The results of the operation will be as shown in Figure 3-22 on page 112.

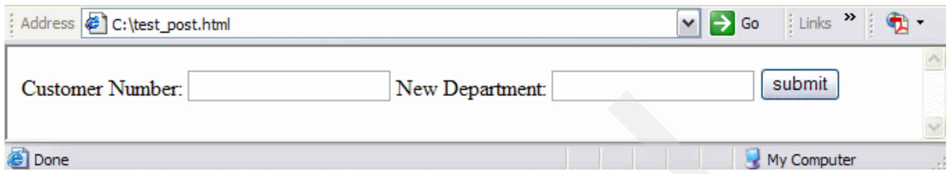


Figure 3-23 Form displayed in the browser

3.5 WEB 2.0 features

Besides the simple exposure of database operations as SOAP Web Service operations, the IBM Data Server Developer Workbench also provides some interesting features which can be leveraged by Web 2.0[9] by providing REST service bindings and XSL processing capabilities. In this section we go through the following capabilities:

- ► XSLT, discussed in 3.5.1, "XSLT" on page 113
- ► XMLHttpRequest (AJAX), discussed in 3.5.2, "XMLHttpRequest (AJAX)" on page 125

3.5.1 XSLT

Just returning the database data as XML provides great value to RESTful Web 2.0 applications. However, there might be situations where XML is not the format the client would like to process. For such cases the IBM Data Server Developer Workbench provides the capability to transform the returned data into different text-based document formats via XSL style sheet processing.

The IBM Data Server Developer Workbench applies default XML tagging rules to the output message[10] which determine the data types and the structure of the resulting XML. The generated schema of the XML output message can be used to develop an XSL style sheet. The style sheet can then be applied to the response message of an operation to transform the data into the desired format on every request.

See Figure 3-24 on page 114 for a graphical illustration of this process.

[9] http://en.wikipedia.org/wiki/web_2#Defining_Web_2.0

[10] Though we refer to output/response data in this section, the tooling has capabilities to transform the input message also. This process is similar to the output transformation and is not discussed.

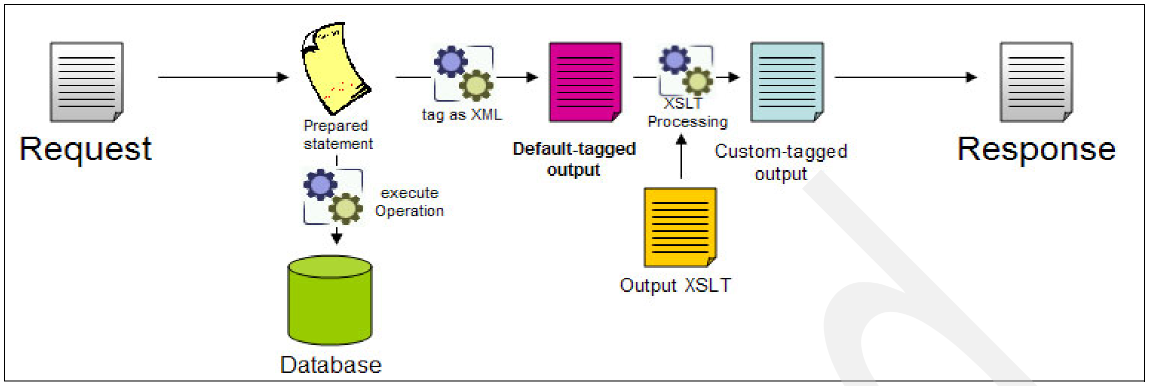
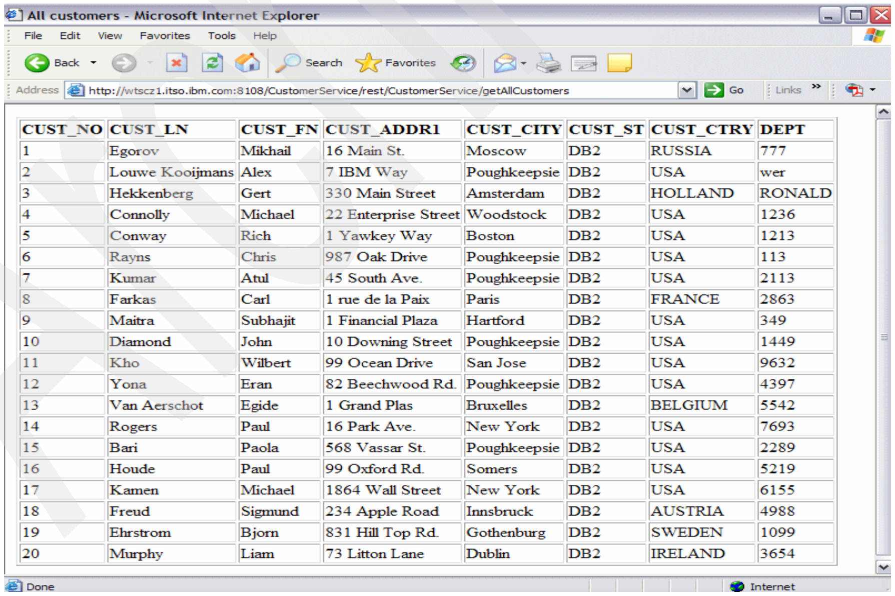


Figure 3-24 Flow with XSLT

We will use the `getAllCustomers` operation of the Web Service created in "Creating a Web Service operation from an SQL script" on page 93 to illustrate this concept.

Figure 3-21 on page 111 showed the results in the browser for the invoke of the `getAllCustomers` REST-style bindings-based operation. Our goal is to apply an XSL stylesheet to this XML output message so the results in the browser will look like Figure 3-25.


All customers - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
Go
Links
Done
Internet

http://wtscz1.itso.ibm.com:8108/CustomerService/rest/CustomerService/getAllCustomers

CUST_NO	CUST_LN	CUST_FN	CUST_ADDR1	CUST_CITY	CUST_ST	CUST_CTRY	DEPT
1	Egorov	Mikhail	16 Main St.	Moscow	DB2	RUSSIA	777
2	Louwe Kooijmans	Alex	7 IBM Way	Poughkeepsie	DB2	USA	wer
3	Hekkenberg	Gert	330 Main Street	Amsterdam	DB2	HOLLAND	RONALD
4	Connolly	Michael	22 Enterprise Street	Woodstock	DB2	USA	1236
5	Conway	Rich	1 Yawkey Way	Boston	DB2	USA	1213
6	Rayns	Chris	987 Oak Drive	Poughkeepsie	DB2	USA	113
7	Kumar	Atul	45 South Ave.	Poughkeepsie	DB2	USA	2113
8	Farkas	Carl	1 rue de la Paix	Paris	DB2	FRANCE	2863
9	Maitra	Subhajit	1 Financial Plaza	Hartford	DB2	USA	349
10	Diamond	John	10 Downing Street	Poughkeepsie	DB2	USA	1449
11	Kho	Wilbert	99 Ocean Drive	San Jose	DB2	USA	9632
12	Yona	Eran	82 Beechwood Rd.	Poughkeepsie	DB2	USA	4397
13	Van Aerschot	Egide	1 Grand Plas	Bruxelles	DB2	BELGIUM	5542
14	Rogers	Paul	16 Park Ave.	New York	DB2	USA	7693
15	Bari	Paola	568 Vassar St.	Poughkeepsie	DB2	USA	2289
16	Houde	Paul	99 Oxford Rd.	Somers	DB2	USA	5219
17	Kamen	Michael	1864 Wall Street	New York	DB2	USA	6155
18	Freud	Sigmund	234 Apple Road	Innsbruck	DB2	AUSTRIA	4988
19	Ehrstrom	Bjorn	831 Hill Top Rd.	Gothenburg	DB2	SWEDEN	1099
20	Murphy	Liam	73 Litton Lane	Dublin	DB2	IRELAND	3654

Figure 3-25 Desired output

The steps to perform the XSLT processing are:

1. Create an XSL stylesheet.
2. Associate the Web Service operation with the XSL stylesheet created in the previous step.

Creating an XSL stylesheet

An *XSLT stylesheet* is nothing but just another XML document. There are special processing instruction tags under the XSL namespace which can be interpreted by an XSLT processor. We start with the XSLT skeleton shown in Example 3-7.

Example 3-7 XSLT skeleton

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="html" encoding="UTF-8" media-type="text/html"/>
<xsl:template match="/*">
</xsl:template>
</xsl:stylesheet>
```

The *output* element defines some basic output information. The `method="html"` tells us that we want to generate an HTML document. The *encoding* tells us that the characters in our response document will be encoded as UTF-8. Finally, the *media-type* defines the MIME type used for the HTTP response message.

The *template* element contains a set of instructions (currently it is empty, but we will fill it up in the next section). A template is somewhat similar to a function definition as we know it from other programming languages. The match attribute with the value “/*” tells us that it will get called on the root element node of the source document. We will not have any other template definitions in this sample since it is a pretty simple one, but an XSL script may contain more than one template element. Inside one template element you can call other template elements (even recursion is possible).

Fist we add the static information for our HTML output. See Figure on page 116.

All tags that are not prefixed with the XSL namespace are not interpreted by the XSLT processor but copied to the output document. We know already that our HTML document contains an <html> tag which itself contains a <header> and a <body> element. Inside the body we will have a <table> element where we get one row per customer entry in our database.

Example 3-8 Adding the static part to the style sheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="html" encoding="UTF-8" media-type="text/html"/>
<xsl:template match="/*">
<html>
<head>
<title>All Customers</title>
</head>
<body>
<table border="1">
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Next we do our first dynamic part: we add the table header. Here we access information from the source document (the XML which gets returned by tooling) the first time. If you recall, the tooling uses the column from the database table names as element names in the XML response. We use those element names for our HTML table heading.

The *xsl:if* element is used to test, if we do have at least one row available. If there is no row we do not get the names of our columns. If the condition is true we address all elements in the first row (via an XPath expression). The *xsl:for-each* instruction tells us to repeat the including instructions for every element in the node list returned by the XPath expression in the *select* attribute.

The *xsl:value-of* instruction writes a string literal to the output. The *select* attribute contains an XPath expression (in this case it is an XPath function returning the local name of the element). This is going to be the name of the table header.

Next we finalize the HMTL sample by filling the table with content. We again use the xsl:for-each instruction to iterate over all rows. And inside we use xsl:for-each again to iterate over all columns. We use the xsl:value-of instruction to write the value out to the result document. We address the value using the XPath function text() to get the text node of the column tag. See Example 3-9 on page 117.

Example 3-9 Transformation code for the table header

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html" encoding="UTF-8" media-type="text/html"/>
<xsl:template match="/*">
<html>
         <head>
           <title>All customers</title>
         </head>
         <body>
            <table border="1">
               <tr>
                  <xsl:if test="//row">
                     <xsl:for-each select="//row[1]/*">
                        <td><b><xsl:value-of select="local-name()"/></b></td>
                     </xsl:for-each>
                   </xsl:if>
               </tr>
            </table>
         </body>
      </html>
</xsl:template>
</xsl:stylesheet
```

Save this file with the .xsl extension.t

Creating the default schema and associating the stylesheet

Right-click the **getAllCustomers** operation and select **Manage XSLT**, as shown in Figure 3-26 on page 118. In the “Configure XSL transformations” pane (Figure 3-27 on page 118) you find a section called “Transformation of Input Message” and one called “Transformation of Output Message” which allow to assign an XSLT style sheet to the input and/or output message. The **Browse** button opens a file dialog that allows you to select the appropriate XSLT file (you created before).

For this example we are only going to do output transformation. We specify the xsl file we created in “Creating an XSL stylesheet” on page 115 in the “Transformation of Output Message” section. However, you will notice that operations that have an XSLT style sheet assigned to their input and/or output message have disappeared in the SOAP binding—they are only available via the HTTP GET and POST REST bindings[11]. See figure Figure 3-28 on page 119. You cannot expose this operation with SOAP bindings because our XSLT returns HTML and not XML.

[11] Context is when viewing through the Web Services explorer.

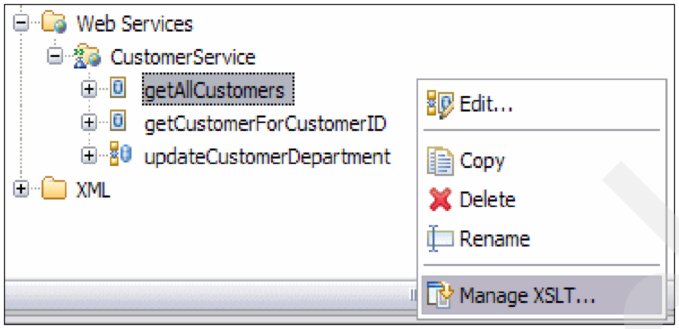
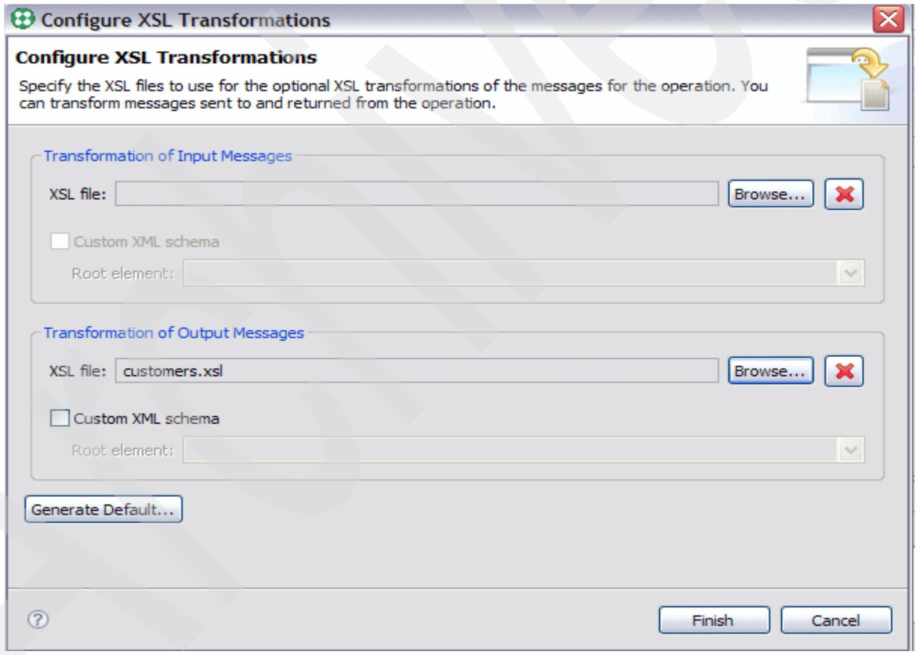


Figure 3-26 Manage XSLT menu

In the Configure XSL transformations pane, specify the .xsl file created and saved earlier, as shown in Figure 3-27.


Configure XSL Transformations

Configure XSL Transformations

Specify the XSL files to use for the optional XSL transformations of the messages for the operation. You can transform messages sent to and returned from the operation.

Transformation of Input Messages

XSL file:

Custom XML schema

Root element:

Transformation of Output Messages

XSL file: customers.xsl

Custom XML schema

Root element:

Figure 3-27 Specifying the stylesheet for output transformation

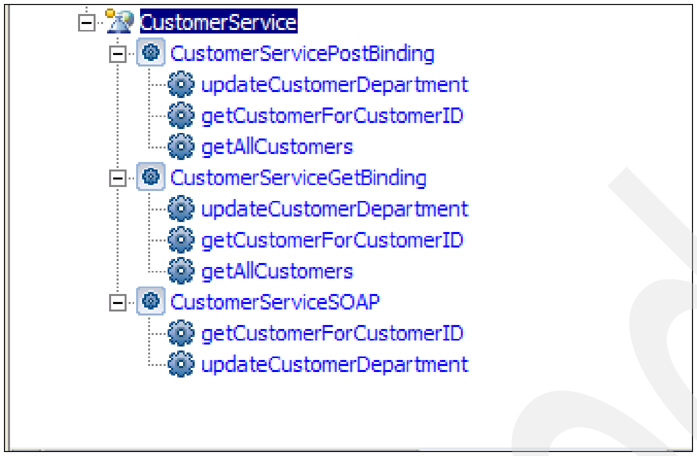


Figure 3-28 Missing SOAP binding for getAllCustomers as seen in WSDL explorer

If you need operations that have XSLT transformations to appear in the SOAP binding, you need to perform additional steps. You need a schema document that describes the custom input and/or output message format via an element definition (we cannot reference type definitions). The flow is shown in Example 3-10.

Example 3-10 Flow for SOAP enablement of XSLT based operations

```
custom (external) input schema -> XSLT -> default input schema -> runtime ->
default output schema -> XSLT -> custom (external) output schema
```

To add a custom schema, right-click the Web Service and select **Manage Custom Schemas**, as shown in Figure 3-29.

.

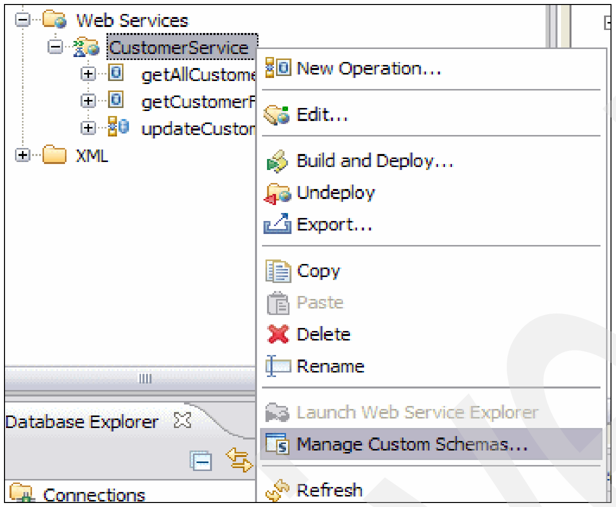
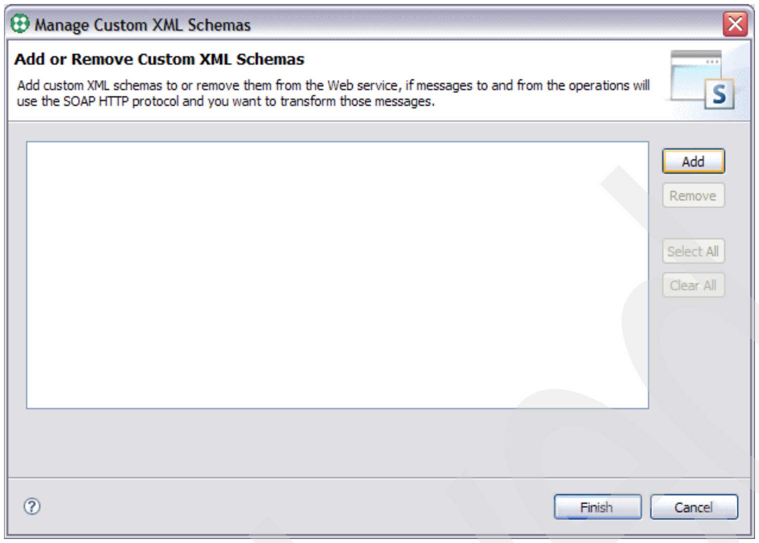


Figure 3-29 Manage Custom Schemas

In the Manage Custom XML Schemas pane, shown in Figure 3-30 on page 121, **Add** opens a file dialog where you can select an XML schema file that is getting added to the service.

Note: Custom schema has constraints. You can add more than one schema to the Web Service, but the target namespaces of all schemas need to be unique. You can add a schema that has the same target namespace as our Web Service. In that case we merge the custom schema with our default schema.

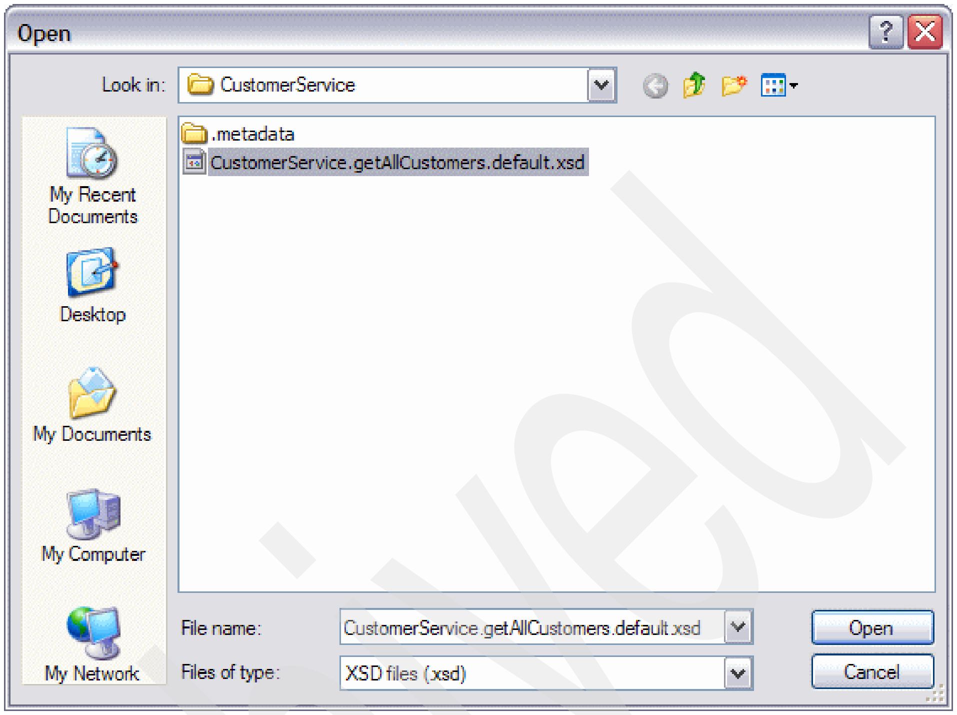

Manage Custom XML Schemas

Add or Remove Custom XML Schemas

Add custom XML schemas to or remove them from the Web service, if messages to and from the operations will use the SOAP HTTP protocol and you want to transform those messages.

Figure 3-30 Manage Custom XML Schemas

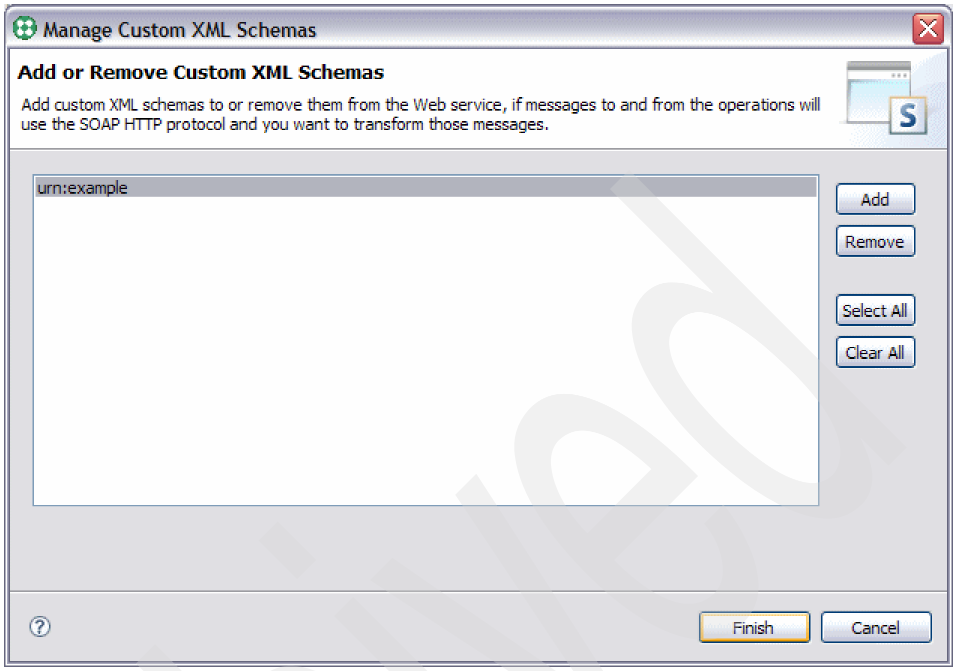
You may choose the Custom XML Schema, as shown in Figure 3-31 on page 122 and Figure 3-32 on page 123.



.metadata
CustomerService.getAllCustomers.default.xsd

File name: CustomerService.getAllCustomers.default.xsd
Files of type: XSD files (.xsd)

Figure 3-31 Choosing the Custom XML Schema - Step 1


Manage Custom XML Schemas

Add or Remove Custom XML Schemas

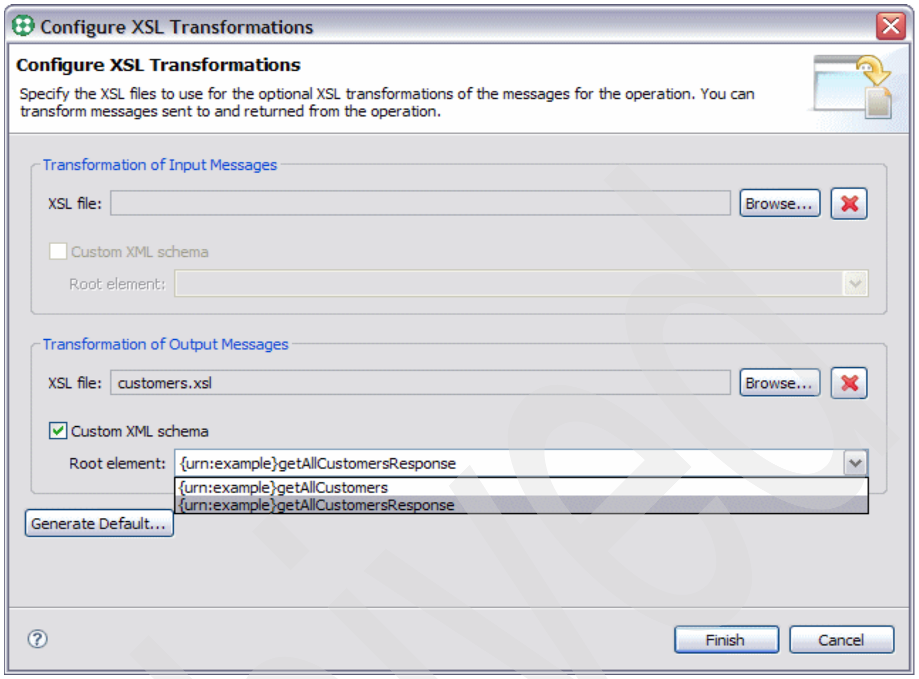
Add custom XML schemas to or remove them from the Web service, if messages to and from the operations will use the SOAP HTTP protocol and you want to transform those messages.

urn:example

Figure 3-32 Choosing the Custom XML Schema - Step 2

Note: The Custom XML Schema comes from somewhere else and the creator of the service must ensure that the XSLT matches that schema and the XSLT's need to generate XML.

Now you need to get back to your operation and assign a message element representing the custom input and/or output message. You do that by using the "Configure XSLT Transformations" dialog shown in Figure 3-33 on page 124. In case you did assign an XSLT to the input and/or output message, you can enable the binding to a schema message element by checking the "Custom XML schema" check box. Now you can select the element definition via the Root Element drop-down box (this box lists all element definitions from all added schemas).


Configure XSL Transformations

Configure XSL Transformations

Specify the XSL files to use for the optional XSL transformations of the messages for the operation. You can transform messages sent to and returned from the operation.

Transformation of Input Messages

XSL file:

[ ] Custom XML schema

Root element:

Transformation of Output Messages

XSL file: customers.xsl

[✓] Custom XML schema

Root element: {urn:example}getAllCustomersResponse

{urn:example}getAllCustomers

{urn:example}getAllCustomersResponse

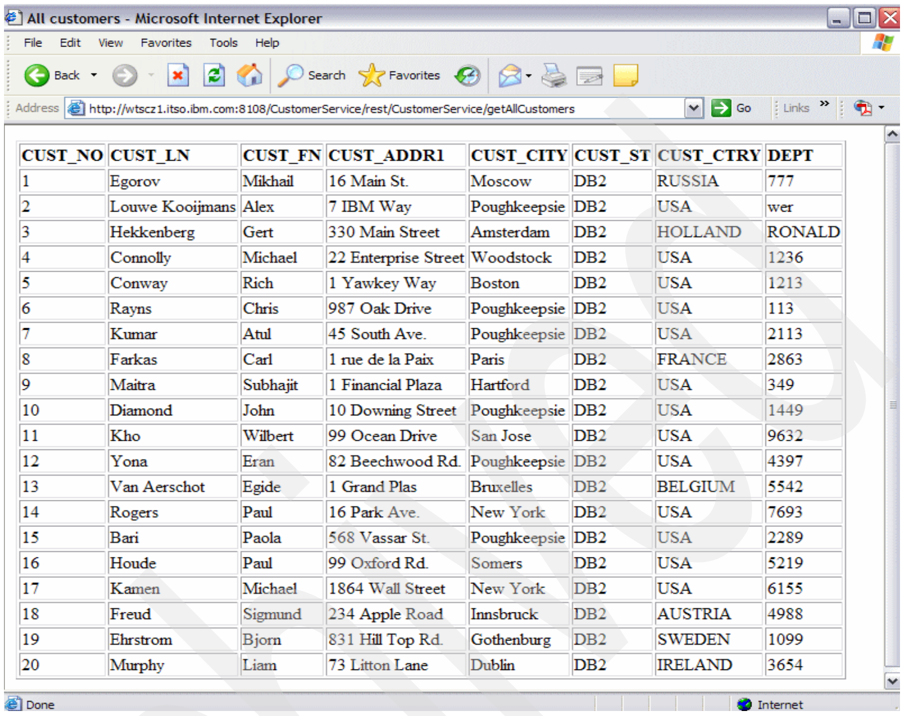
Figure 3-33 Associating the Custom XML Schema

After redeploying this application to WebSphere, access the REST end point with:

```
http://wtscz1.itso.ibm.com:8108/CustomerService/rest/CustomerService
/getAllCustomers
```

Note: You may have noticed a Generate Default button. This generates the XML schema that would be used if no XSLT(s) are assigned. This function is supposed to be used by people who want to know what the default schema would look like—for example, the default schema can be imported into XSLT mapping editors to generate XSLT mapping files to map from an external custom schema into our default schema.

The results should now look as shown in Figure 3-34.


All customers - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
Go
Links
Done
Internet

http://wtscz1.itso.ibm.com:8108/CustomerService/rest/CustomerService/getAllCustomers

CUST_NO	CUST_LN	CUST_FN	CUST_ADDR1	CUST_CITY	CUST_ST	CUST_CTRY	DEPT
1	Egorov	Mikhail	16 Main St.	Moscow	DB2	RUSSIA	777
2	Louwe Kooijmans	Alex	7 IBM Way	Poughkeepsie	DB2	USA	wer
3	Hekkenberg	Gert	330 Main Street	Amsterdam	DB2	HOLLAND	RONALD
4	Connolly	Michael	22 Enterprise Street	Woodstock	DB2	USA	1236
5	Conway	Rich	1 Yawkey Way	Boston	DB2	USA	1213
6	Rayns	Chris	987 Oak Drive	Poughkeepsie	DB2	USA	113
7	Kumar	Atul	45 South Ave.	Poughkeepsie	DB2	USA	2113
8	Farkas	Carl	1 rue de la Paix	Paris	DB2	FRANCE	2863
9	Maitra	Subhajit	1 Financial Plaza	Hartford	DB2	USA	349
10	Diamond	John	10 Downing Street	Poughkeepsie	DB2	USA	1449
11	Kho	Wilbert	99 Ocean Drive	San Jose	DB2	USA	9632
12	Yona	Eran	82 Beechwood Rd.	Poughkeepsie	DB2	USA	4397
13	Van Aerschot	Egide	1 Grand Plas	Bruxelles	DB2	BELGIUM	5542
14	Rogers	Paul	16 Park Ave.	New York	DB2	USA	7693
15	Bari	Paola	568 Vassar St.	Poughkeepsie	DB2	USA	2289
16	Houde	Paul	99 Oxford Rd.	Somers	DB2	USA	5219
17	Kamen	Michael	1864 Wall Street	New York	DB2	USA	6155
18	Freud	Sigmund	234 Apple Road	Innsbruck	DB2	AUSTRIA	4988
19	Ehrstrom	Bjorn	831 Hill Top Rd.	Gothenburg	DB2	SWEDEN	1099
20	Murphy	Liam	73 Litton Lane	Dublin	DB2	IRELAND	3654

Figure 3-34 Final results with XSLT processing

3.5.2 XMLHttpRequest (AJAX)

To illustrate this scenario, we will use the getCustomerForCustomerID[12] operation.

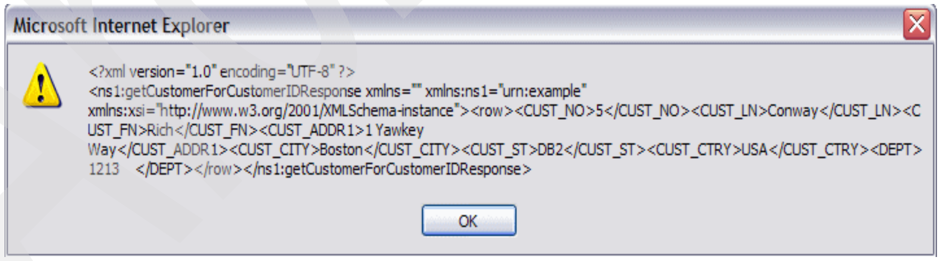
This is a simple HTML document that executes the Web Service call on load. It uses the HTTP POST XML binding of the Web Service, posting the request in the form required for the default tagged XML format for the getCustomerForCustomerID operation. It prints the XML response in a JavaScript™ alert box. See Example 3-11 on page 126 for the HTML and Figure 3-35 on page 127 for the output.

[12] This is a new operation that was created with the same documented pattern.

Note: If you have popups disabled in your browser, you must enable them before you load this HTML. Otherwise, you will not see any popup and get no indication as to why the load failed.

Example 3-11 Sample XMLHTTPrequest

```
<html>
<head>
<title>Simple HTTP POST (XML) Test</title>
</head>
<script type="text/javascript">

// create the XMLHttpRequest object
// count for differences by browser
var xmlhttp;

// Firefox, Opera, Mozilla, ...
if(window.XMLHttpRequest && !(window.ActiveXObject)) {
   try {
      xmlhttp = new XMLHttpRequest();
   } catch(e) {
      xmlhttp = false;
   }
// branch for IE/Windows ActiveX version
} else if(window.ActiveXObject) {
   try {
      xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
   } catch(e) {
      try {
         xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
      } catch(e) {
         xmlhttp = false;
      }
   }
}

// issue request
if(xmlhttp) {
   try {
      // define the operation endpoint URL


xmlhttp.open("POST","http://wtscz1.itso.ibm.com:8108/CustomerService/re
st/CustomerService/getCustomerForCustomerID",true);
//CustomerService/rest/CustomerService/getCustomerForCustomerID

      xmlhttp.onreadystatechange=function() {
         if (xmlhttp.readyState==4) {
            alert(xmlhttp.responseText);
         }
      }

      // set the request content type to XML
      xmlhttp.setRequestHeader("Content-Type", "text/xml");
      // send the request as XML (following the data web services
default schema)
      xmlhttp.send("<q0:getCustomerForCustomerID
xmlns:q0=\"urn:example\"><CUST_NO>5</CUST_NO></q0:getCustomerForCustome
rID>");

   } catch (e) {
      alert(e);
   }
}

</script>
<body>

</body>
</html>
```



```
<?xml version="1.0" encoding="UTF-8" ?>
<ns1:getCustomerForCustomerIDResponse xmlns="" xmlns:ns1="urn:example"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><row><CUST_NO>5</CUST_NO><CUST_LN>Conway</CUST_LN><C
UST_FN>Rich</CUST_FN><CUST_ADDR1>1 Yawkey
Way</CUST_ADDR1><CUST_CITY>Boston</CUST_CITY><CUST_ST>DB2</CUST_ST><CUST_CTRY>USA</CUST_CTRY><DEPT>
1213   </DEPT></row></ns1:getCustomerForCustomerIDResponse>
```

Figure 3-35   Sample XMLHTTPRequest output

3.6 DB2 as a Web Service consumer

A *User Defined Function (UDF)* is an extension or addition to the existing built-in functions of SQL, such as those provided in the SYSFUN schema in DB2. User-defined functions and stored procedures are sometimes referred to collectively as routines. We use DB2 UDFs to illustrate this scenario. We can use the workbench to create the UDFs that access the operations of the Web Service. We use WebSphere Developer for System z version 7 tooling for this section.

At this time, we can only access SOAP bindings from a UDF. We cover the following scenario:

Calling the Web Service operation getCustomerForCustomerID via the SOAP binding over HTTP.

3.6.1 Prerequisites

The prerequisites for this scenario are:

- Enable the DB2 XML Extender database.
- Enable the Web Services consumer UDFs for the database.
- Create a project that you want to use with the Web service UDF.
- Create a connection to the database that you just enabled. Refer to 3.2.1, “Configuring the Workbench for developing and deploying Web Services” on page 89 for details about this.

3.6.2 Procedure

Follow these steps:

1. Select **File** → **New** → **Other** from the menu. Then expand **Data**. The folder expands and you select **Web Service User-Defined Function** from the list of wizards. See Figure 3-36 on page 129. Click **Next** to proceed.

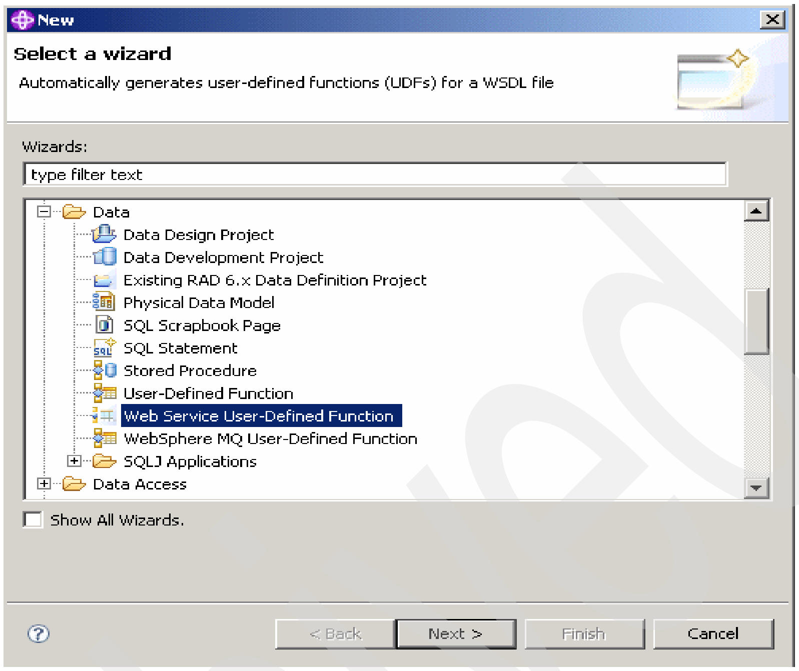


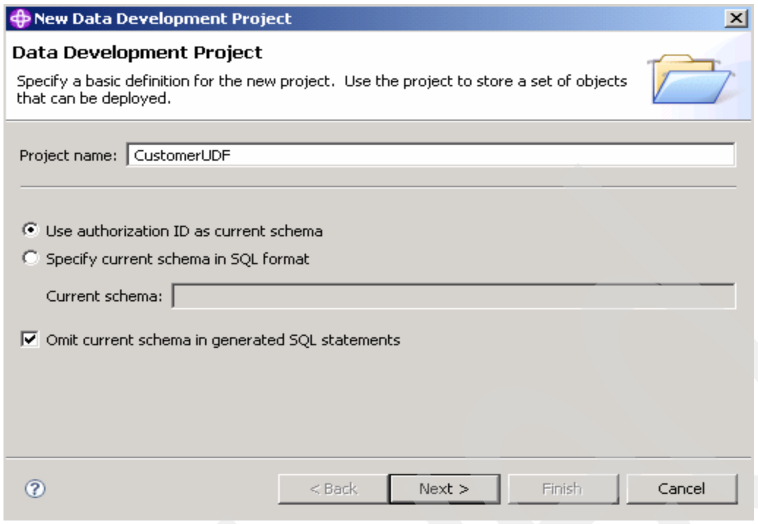
Figure 3-36 New Web Service User-Defined Function

2. Click **New** to invoke the “New Data Development project” wizard. In this wizard, as shown in Figure 3-37 on page 130, Figure 3-38 on page 130, and Figure 3-39 on page 131, perform the following:

   a. Enter the project name as CustomerUDF and leave the defaults as is.

   b. Choose the connection you created in the next window.

   c. Take the defaults in the next window and finish.



New Data Development Project

Data Development Project

Specify a basic definition for the new project. Use the project to store a set of objects that can be deployed.

Project name: CustomerUDF

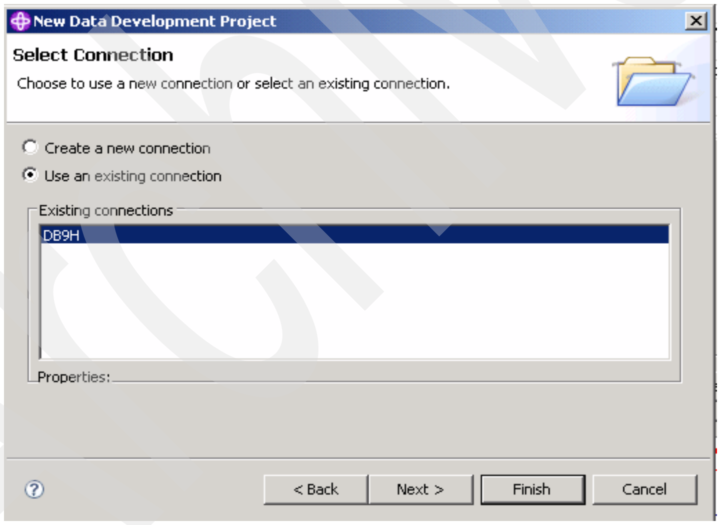
(●) Use authorization ID as current schema

( ) Specify current schema in SQL format

Current schema:

[✓] Omit current schema in generated SQL statements

Figure 3-37 Provide the project name



New Data Development Project

Select Connection

Choose to use a new connection or select an existing connection.

( ) Create a new connection

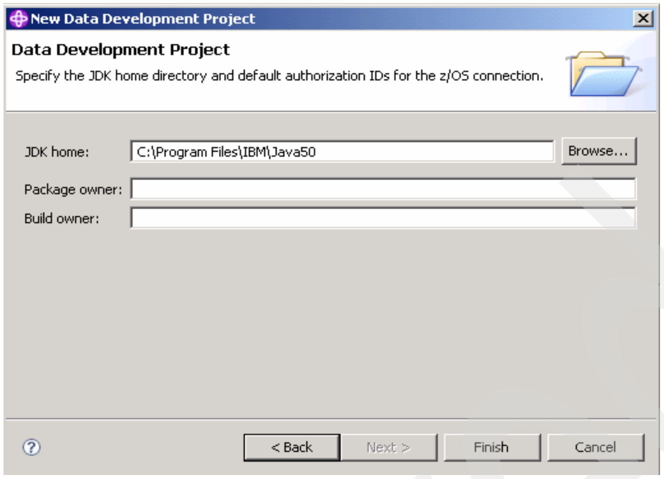
(●) Use an existing connection

Existing connections

DB9H

Properties:

Figure 3-38 Choose an existing connection


New Data Development Project

Data Development Project

Specify the JDK home directory and default authorization IDs for the z/OS connection.

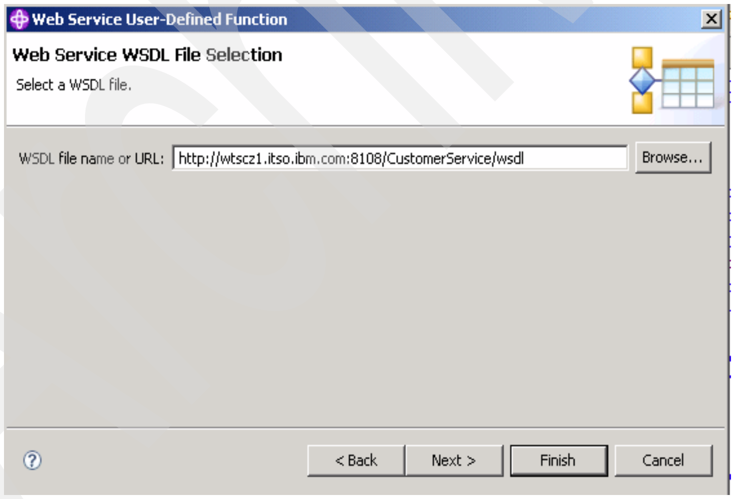
JDK home: C:\Program Files\IBM\Java50

Package owner:

Build owner:

Figure 3-39 Verify the JDK location

3. In the Web Service WSDL selection window, specify the WSDL file as deployed on the server and click **Next**. We use this WSDL file to generate the UDF. See Figure 3-40.


Web Service User-Defined Function

Web Service WSDL File Selection

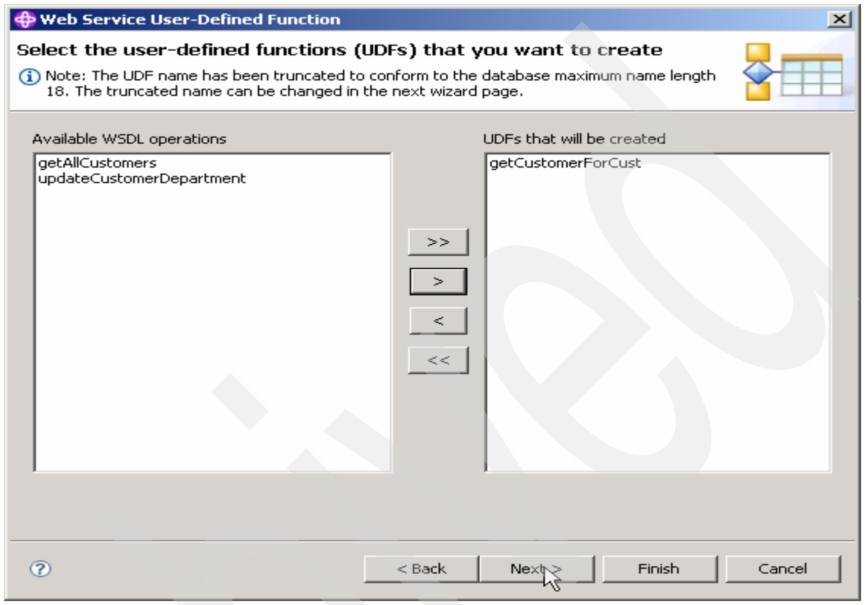
Select a WSDL file.

WSDL file name or URL: http://wtscz1.itso.ibm.com:8108/CustomerService/wsdl

Figure 3-40 Specify the WSDL location

4. Select the database. You see the database connection and a schema for which the UDF is generated. Click **Browse** to select a database schema from the work space. The wizard requires that the database is enabled for the Web Service consumer UDFs, and the DB2 XML Extender. If there is currently no

connection to the specified database, an additional message window asks for connection information. You can choose to immediately deploy the generated UDF into the database or generate a UDF in the workspace only. You can deploy the UDF later. See Figure 3-42 on page 133.


Web Service User-Defined Function

Select the user-defined functions (UDFs) that you want to create

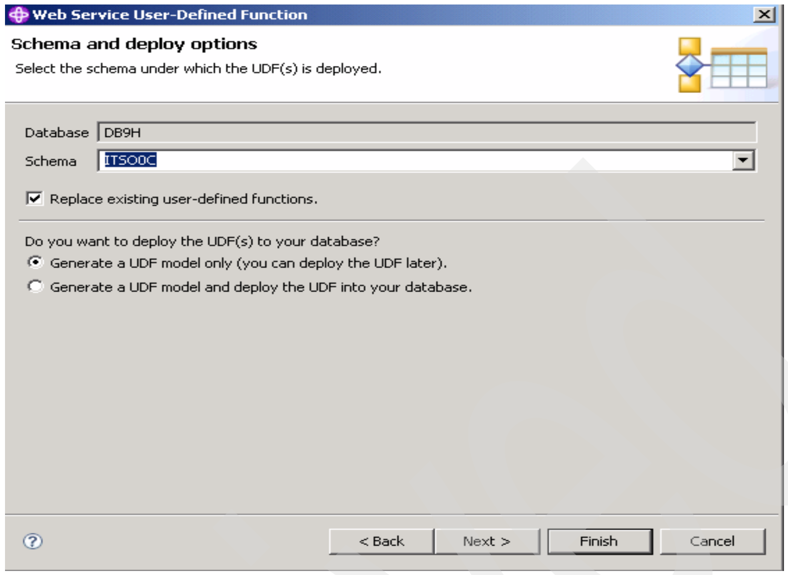
Note: The UDF name has been truncated to conform to the database maximum name length 18. The truncated name can be changed in the next wizard page.

Available WSDL operations
getAllCustomers
updateCustomerDepartment

UDFs that will be created
getCustomerForCust

Figure 3-41 Select the operation for UDF to call out

5. In the “Select the UDF you want to create” window choose the WSDL operation you want the UDF to invoke. For this example, we chose the getCustomerForCustomerID operation. Figure 3-41 shows the operation chosen. The wizard generates one UDF for every operation that is selected.


Web Service User-Defined Function

Schema and deploy options

Select the schema under which the UDF(s) is deployed.

Database: DB9H

Schema: ITSO0C

[✓] Replace existing user-defined functions.

Do you want to deploy the UDF(s) to your database?

[●] Generate a UDF model only (you can deploy the UDF later).

[ ] Generate a UDF model and deploy the UDF into your database.

Figure 3-42 Schema and deploy options

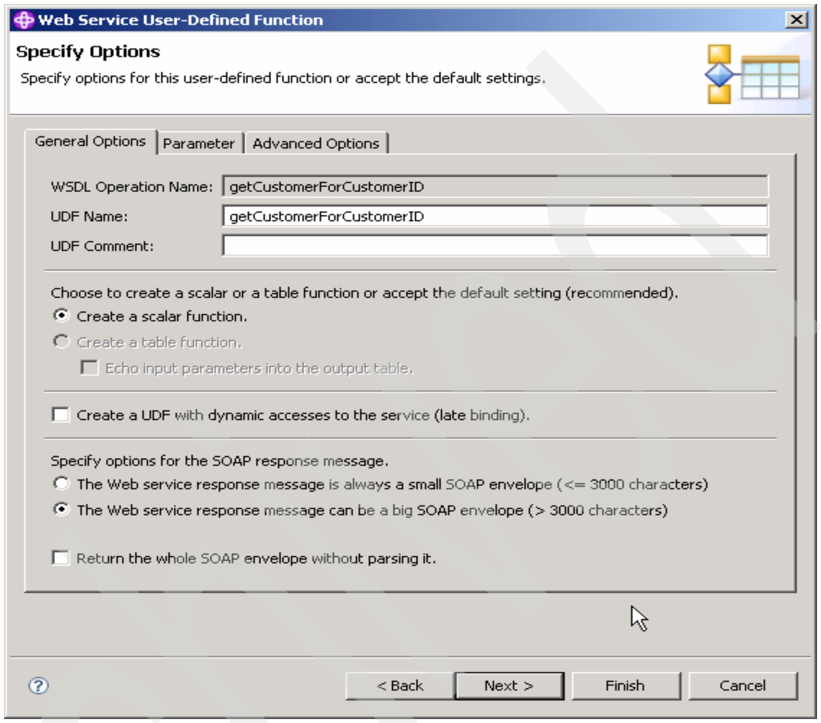
6. Select options for the UDF. For each operation selected in the previous step, you can define options for those UDFs, such as changing the function name, or providing comments on the function. For our example, we chose the selection shown in Figure 3-43 on page 134.

   – You can choose to build a scalar or a table function. Switching from a table function to a scalar function makes sense when the wizard should not automatically map the returned types. In this case, the wizard should return them as an XML fragment. Being able to switch from a scalar to a table function allows you to use the UDF in a FROM clause.

     - The wizard generates a scalar function when the Web Service returns a simple XML type.
     - The wizard generates a table function when it returns a complex XML type. The table function automatically maps the complex XML type into multiple columns.

   – You can include the input parameters as columns in the output table by selecting the "Echo the input parameters into the output table" check box.

   – You can choose to generate a UDF with dynamic access to the Web Service. When you do not specify the service location (the location attribute of the soap:address element) in the WSDL document, you generate a dynamic function. You can select the "Create a UDF with dynamic accesses to the service" check box even when the service location is specified to make use of late binding. When you generate a dynamic function, specify the service location at runtime as a parameter of the UDF.


Web Service User-Defined Function

Specify Options

Specify options for this user-defined function or accept the default settings.

WSDL Operation Name: getCustomerForCustomerID

UDF Name: getCustomerForCustomerID

UDF Comment:

Choose to create a scalar or a table function or accept the default setting (recommended).

- [✓ Create a scalar function.]
- Create a table function.
  - Echo input parameters into the output table.

- Create a UDF with dynamic accesses to the service (late binding).

Specify options for the SOAP response message.

- The Web service response message is always a small SOAP envelope (<= 3000 characters)
- [✓ The Web service response message can be a big SOAP envelope (> 3000 characters)]

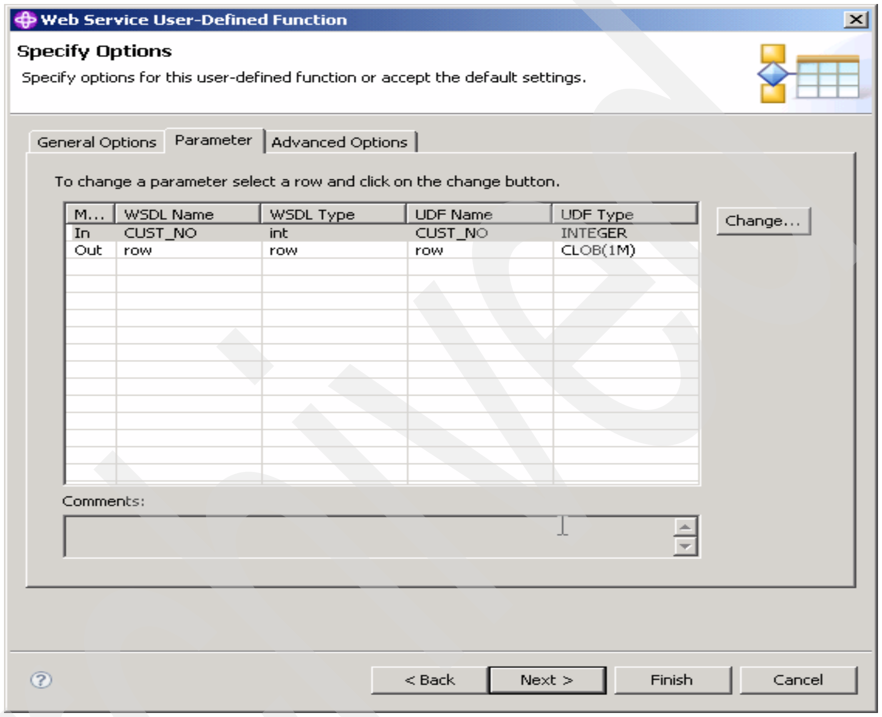
- Return the whole SOAP envelope without parsing it.

Figure 3-43 General Options

– When using a Web Service that can return responses of more than 3000 characters, check the “The Web service response message can be a big SOAP envelope” radio button. By default, the “The Web service response message is always a small SOAP envelope” radio button is specified because this results in better performance for most Web Services. If you specify the small SOAP response option and the wizard returns a SOAP envelope with more than 3000 characters, the generated Web Service UDF returns a descriptive error message.

– Select the “Return the whole SOAP envelope without parsing it” check box to help in debugging the Web Services consumer UDFs.

7. From the Parameter page of the Options window, you can review and change the parameter mappings from WSDL types to SQL types. See Figure 3-44.
8. From the Advanced Options page, you can specify the name for the UDF. If you do not specify a name, then a unique name is automatically generated by the database when you deploy the UDF.



Specify Options

Specify options for this user-defined function or accept the default settings.

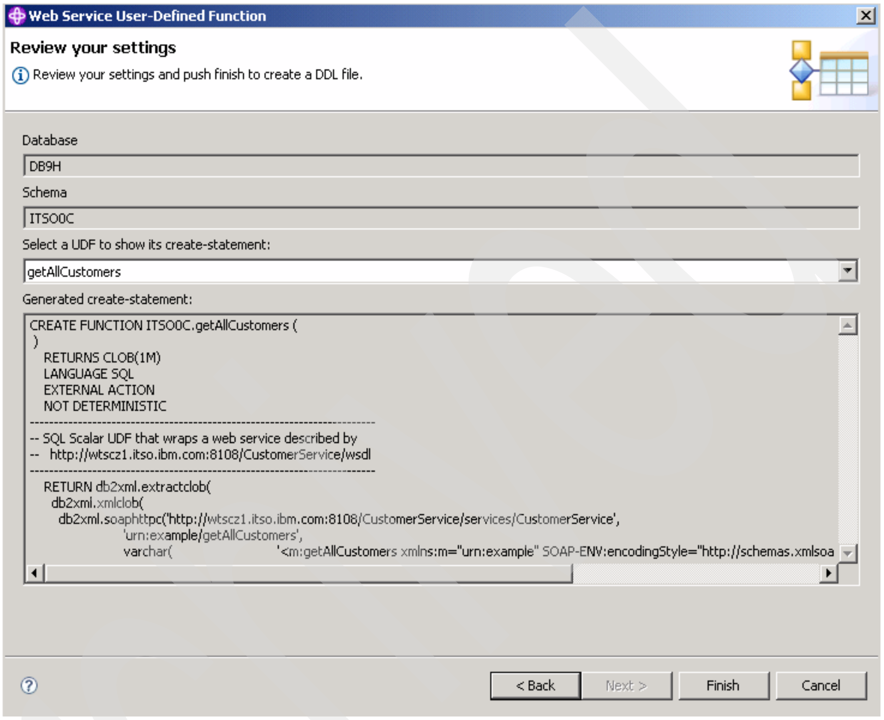
To change a parameter select a row and click on the change button.

M...	WSDL Name	WSDL Type	UDF Name	UDF Type
In	CUST_NO	int	CUST_NO	INTEGER
Out	row	row	row	CLOB(1M)

Figure 3-44 Specify the parameters

9. Review the settings for generating the UDFs. Examine the database and schema, and the UDF statement that is shown. See Figure 3-45.

10. Click **Finish**. This generates the UDF but does not deploy it to the database, because of the earlier selections to generate only and not deploy[13].



Review your settings

Review your settings and push finish to create a DDL file.

Database: DB9H

Schema: ITSO0C

Select a UDF to show its create-statement: getAllCustomers

Generated create-statement:

```
CREATE FUNCTION ITSO0C.getAllCustomers (
 )
   RETURNS CLOB(1M)
   LANGUAGE SQL
   EXTERNAL ACTION
   NOT DETERMINISTIC
-----------------------------------------------------------------
-- SQL Scalar UDF that wraps a web service described by
--   http://wtscz1.itso.ibm.com:8108/CustomerService/wsdl
-----------------------------------------------------------------
   RETURN db2xml.extractclob(
     db2xml.xmlclob(
       db2xml.soaphttpc('http://wtscz1.itso.ibm.com:8108/CustomerService/services/CustomerService',
                       'urn:example/getAllCustomers',
                       varchar(                       '<m:getAllCustomers xmlns:m="urn:example" SOAP-ENV:encodingStyle="http://schemas.xmlsoa
```

Figure 3-45 Showing all the generated UDFs

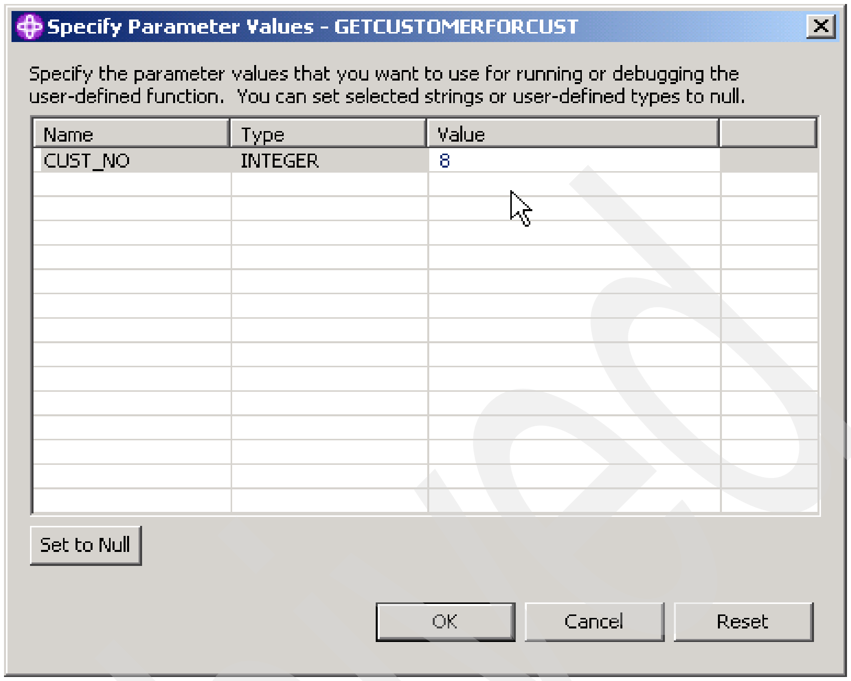
11. You can run the Web Service consumer UDF directly from the workspace. To run the deployed UDF:

    a. Right-click the UDF.

    b. Select **Run**. Specify the parameter values, as shown in Figure 3-46 on page 137.

    c. Click **OK** to see the results of your test. See Figure 3-47 on page 137.

[13] It will help to deploy at a later step as it helps to isolate issues specific to deployment.

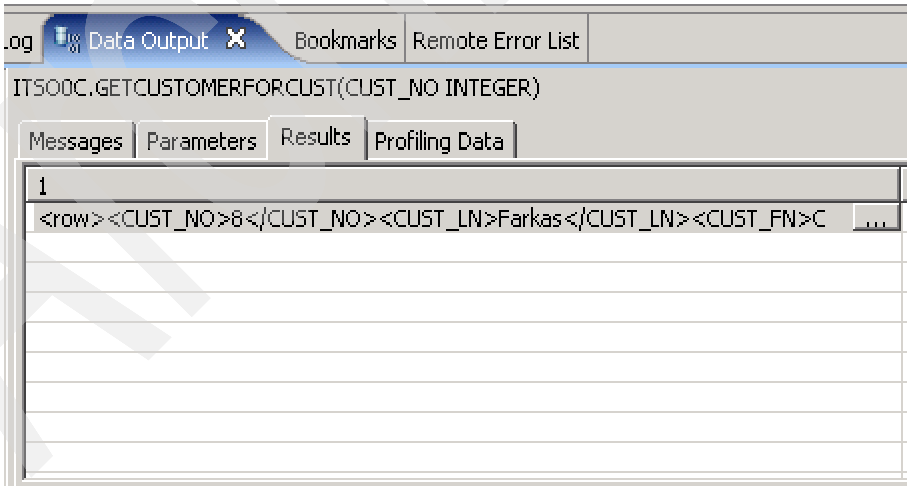


Specify Parameter Values - GETCUSTOMERFORCUST

Specify the parameter values that you want to use for running or debugging the user-defined function. You can set selected strings or user-defined types to null.

Name	Type	Value
CUST_NO	INTEGER	8

Figure 3-46 Specify the input parameter



```
ITSO0C.GETCUSTOMERFORCUST(CUST_NO INTEGER)
1
<row><CUST_NO>8</CUST_NO><CUST_LN>Farkas</CUST_LN><CUST_FN>C
```

Figure 3-47 Results of the UDF invoke

3.7 Additional material

Table 3-3 describes the additional material supplied with this book. Refer to Appendix B, “Additional material” on page 397 for more details.

Table 3-3 Supplied additional material

Name	Purpose
customer_table_create.sql	DDL for creating the Customer table
update_customer_submit_form.html	Submit form referenced in Figure 3-23 on page 113
AllCustomers.xsl	Stylesheet to transform the results of the allCustomers operation invoke, referenced in Figure on page 117
xmlhttp_ajax.html	The HTML form to illustrate usage of XMLHTTPRequest to invoke Web Service.Example 3-11 on page 126
CustomerServiceCustomerService.war	The WAR file as output by the tooling. When deploying, specify “CustomerService” as context root and make sure you map the resource reference for the database with your JNDI name.
CustomerServicePI.zip	The project interchange file with the Data Project and the generated Web project

4

Accessing IMS transactions as Web Services

In this chapter we take you through the steps of using the Enterprise Service Tools (EST) feature of Rational Developer for System z Version 7 (RDz) to enable an existing IMS COBOL program to be accessed as a Web service through the IMS SOAP Gateway.

This chapter is organized as follows:

- IMS SOAP Gateway overview
- An overview of the tasks required
- Exposing the sample IMS PhoneBook lookup transaction as a Web Service
- Deploying the new Web Service using the IMS SOAP Gateway deployment utility
- Testing the deployed Web Service

4.1 Enterprise Service Tools

Enterprise Service Tools (EST) is a set of features that aid you in the transformation of COBOL- and PL/I-based business applications, exposing them as Web Services.

Enterprise Service Tools perspective

The Enterprise Service Tools perspective provides views and editors that assist you in developing service flow projects and EST single-service projects (including Web Services for CICS projects, SOAP for CICS projects, IBM SOAP Gateway projects, and Batch, TSO, USS projects).

4.1.1 XML Services for the Enterprise

XML Services for the Enterprise (XSE) tools is part of the Enterprise Service Tools (EST) along with other tools such as Service Flow Feature.

The XSE capability of Rational Developer for System z Version 7 (RDz) includes tools that let you easily adapt existing COBOL-based applications to both consume and produce XML messages. This adaptation is accomplished without modifying the existing COBOL application and more importantly without modifying the other applications that call the existing COBOL application. A COBOL application must be able to process XML messages to be a Web Service, since XML is the transport used to describe the Web Service request and the parameters that are passed to the application.

4.2 IMS SOAP Gateway overview

Note: For the latest information about the IMS SOAP Gateway and downloading the software, visit:

http://www.ibm.com/software/data/ims/soap/

IBM IMS SOAP Gateway is a Web Services solution that enables IMS applications to interoperate outside of the IMS environment through *Simple Object Access Protocol (SOAP)* to provide and request services independent of platform, environment, application language, or programming model.

In this chapter we expose an IMS COBOL application as a Web Service by using a wizard from the Enterprise Service Tools in RDz to generate Web Service files for the IMS COBOL application. You can then deploy these Web Services files to the IMS SOAP Gateway to make an IMS application available as a Web Service.

Different types of client applications, such as Microsoft .NET, Java, and third-party applications, can submit SOAP requests into IMS to drive the business logic of the IMS applications.

IMS SOAP Gateway is compliant with the industry standards for Web Services, including SOAP/HTTP 1.1 and WSDL 1.1. This allows your IMS assets to interoperate openly with various types of applications.

When IMS applications are deployed as Web Service providers, the IMS SOAP Gateway server receives the SOAP message from the client application, converts it to an IMS input message, and sends it to IMS. It then receives the output message from IMS and converts it to a SOAP message to send back to that client.

We go through the sample message flow as illustrated in Figure 4-1 on page 142. The numbers in this figure correspond to the description that follows.

1. The Web Service client application sends a SOAP message to IMS SOAP Gateway, which contains input to the IMS application in an XML format.
2. IMS SOAP Gateway processes the SOAP header (XML) and retrieves the appropriate correlation and connection information for the input request.
3. IMS SOAP Gateway sends the input XML data to IMS Connect using TCP/IP after adding the appropriate IMS Connect header.
4. IMS Connect calls the XML Adapter, which in turn calls the XML Converter to perform the XML to IMS application format transformation.
5. IMS Connect then sends the message for further processing. From this point on, the processing is the same as a normal transaction flow. The transaction gets executed and the output is queued.
6. On receipt of the response message from IMS, IMS Connect calls the XML Adapter in order to perform the transformation of the IMS application format to XML.
7. IMS Connect sends the output XML message back to IMS SOAP Gateway using TCP/IP.
8. IMS SOAP Gateway wraps a SOAP header on the output message and sends it back to the client application.

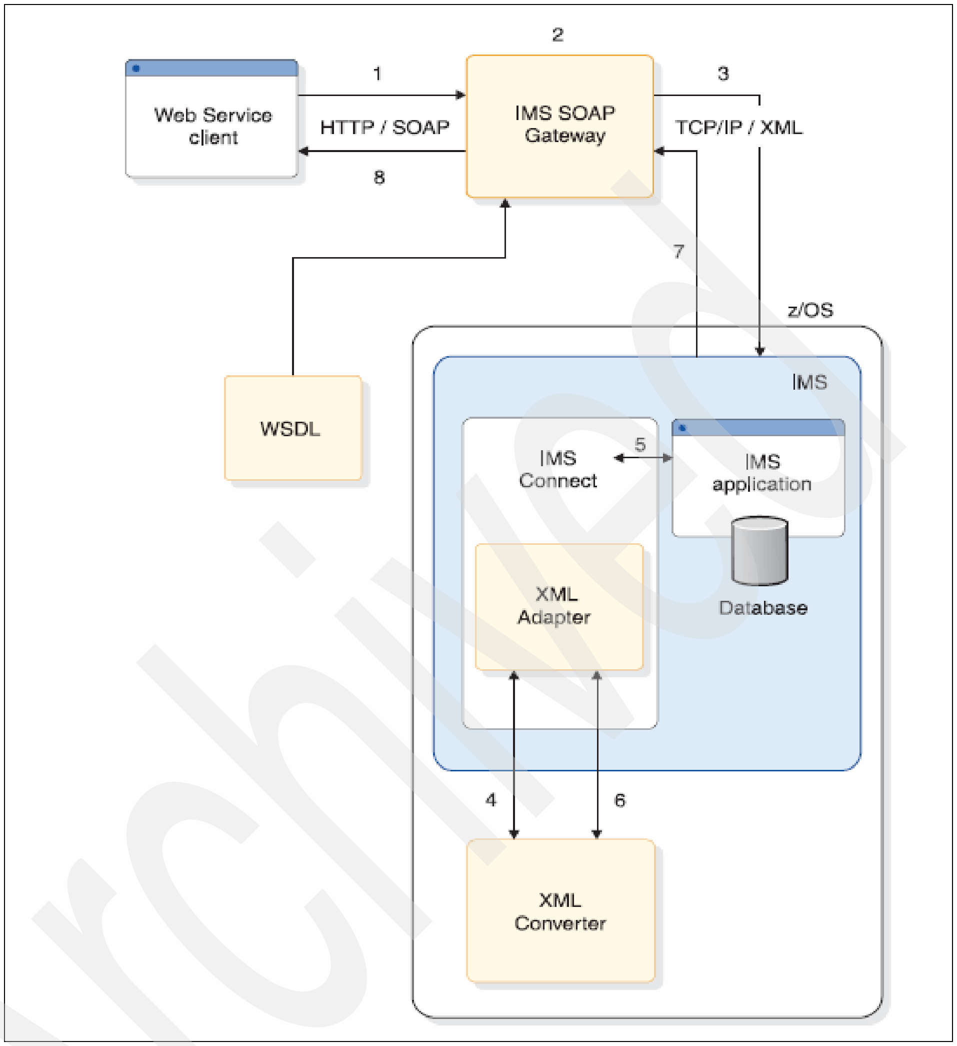
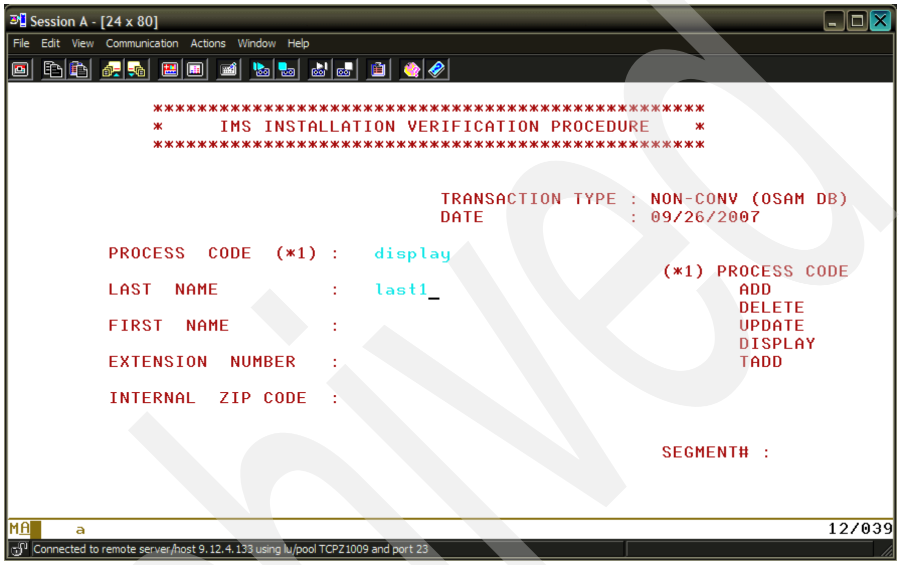


Figure 4-1 IMS SOAP Gateway runtime environment when IMS applications enabled as Web Service providers receive inbound requests from external client applications

4.3 Exposing the Phone Book transaction as a Web Service

In this example, the application interacts with the IVTNO PhoneBook application program in IMS. To be able to set up this scenario on your system, you need to obtain information such as the host name and port number of your IMS Connect and the name of the IMS datastore where the transaction will run. In addition, you need to have set up the phonebook transaction in your IMS system in order to access the phonebook record from the IMS server. Contact your IMS system administrator for this setup.

You can invoke the PhoneBook lookup transaction on your IMS system by typing in `/FOR IVTNO`. You will get a screen that is almost like Figure 4-2.[1]

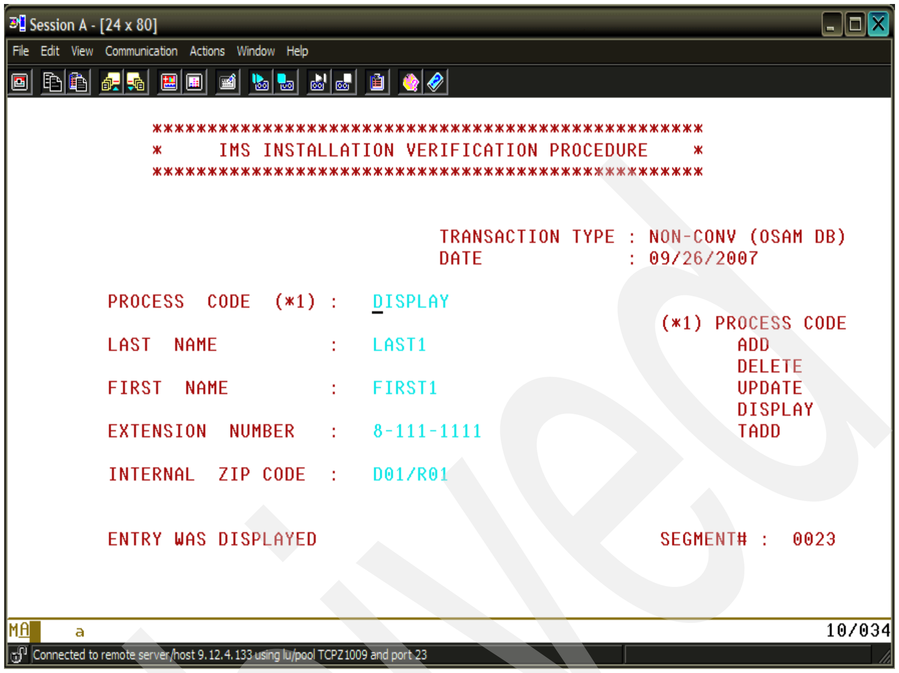


```
        **************************************************
        *     IMS INSTALLATION VERIFICATION PROCEDURE    *
        **************************************************

                                     TRANSACTION TYPE : NON-CONV (OSAM DB)
                                     DATE             : 09/26/2007

    PROCESS  CODE  (*1) :   display
                                                  (*1) PROCESS CODE
    LAST  NAME          :   last1_                       ADD
                                                         DELETE
    FIRST  NAME         :                                UPDATE
                                                         DISPLAY
    EXTENSION  NUMBER   :                                TADD

    INTERNAL  ZIP CODE  :


                                                  SEGMENT# :

MA      a                                                            12/039
Connected to remote server/host 9.12.4.133 using lu/pool TCPZ1009 and port 23
```

Figure 4-2 IVTNO transaction initial screen invoked

Enter the value `display` for PROCESS CODE and the value `last1` for LAST NAME as shown in Figure 4-2 and press enter. You will see a panel like Figure 4-3 on page 144. Our goal in this chapter is to expose these same results as a Web Service that any client can call into.

[1] You will not see the value "display" and "last1" in there. You will be typing it in the next step



```
*************************************************
*     IMS INSTALLATION VERIFICATION PROCEDURE    *
*************************************************


                         TRANSACTION TYPE : NON-CONV (OSAM DB)
                         DATE             : 09/26/2007

PROCESS  CODE  (*1) :   DISPLAY
                                                 (*1) PROCESS CODE
LAST  NAME          :   LAST1                           ADD
                                                        DELETE
FIRST  NAME         :   FIRST1                          UPDATE
                                                        DISPLAY
EXTENSION  NUMBER   :   8-111-1111                      TADD

INTERNAL  ZIP CODE  :   D01/R01


ENTRY WAS DISPLAYED                              SEGMENT# :  0023
```

```
MA     a                                                        10/034
Connected to remote server/host 9.12.4.133 using lu/pool TCPZ1009 and port 23
```

Figure 4-3 Result of the search

There are two steps to enabling the IMS application as a Web Service running in the IMS SOAP Gateway environment:

1. Use the Create New Service Interface (bottom-up) wizard from the Enterprise Service Tools (EST) in RDz to generate Web Service files for the IMS application.
2. Use the *IMS SOAP Gateway Deployment* utility to deploy the Web Service interface to IMS SOAP Gateway. This utility enables you to define connection and correlation information for the Web service.

In the first step, you use the Create New Service Interface (bottom-up) wizard from the Enterprise Service Tools to generate the Web Service files from an existing COBOL source file in your IMS application. The COBOL source file must contain the inbound and outbound interface data structures for the IMS application.

The wizard generates the following components:

- ► A WSDL file, which describes the Web Service interface of the IMS application so that the client can communicate with the Web Service.

- A *correlator* file, which contains information that enables the IMS SOAP Gateway to set the IMS properties and call the IMS application.
- A COBOL source file that contains a Web Service driver and runtime XML conversion programs. These conversion programs convert inbound and outbound data in XML format, which is the format used by SOAP, to and from the inbound and outbound COBOL interface data structures in your IMS application.

4.4 Developing A Web Service using Rational Developer for z

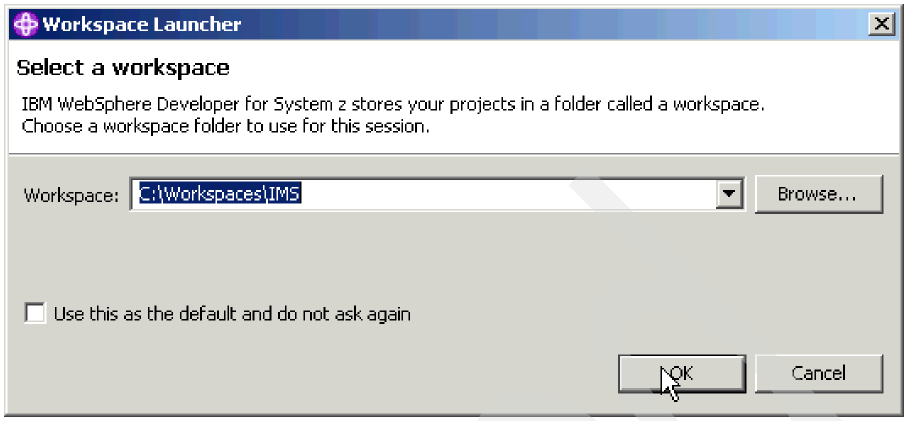
We will use the Enterprise Service Tools inside RDz to generate the artifacts that are needed to enable our existing IMS COBOL application Phonebook lookup to run as a Web Service in the IMS SOAP Gateway environment.

Attention: The GA version of IMS SOAP Gateway on z/OS is only compatible with Rational Developer for System z Version 7.1.1. With earlier versions of RDz or WDz the generated XML files are incompatible with the IMS SOAP Gateway runtime on z/OS.

To generate the artifacts that are needed to enable an existing IMS COBOL application for the IMS SOAP Gateway, you must have a COBOL copybook or COBOL source program that describes the format of the input and output messages for the IMS application.

Because artifacts generated by the Enterprise Service Tools wizard (the WSDL file, the correlator file, and the COBOL source file containing the Web Service driver and the runtime XML conversion programs) should be transferred to a z/OS system, you can use the z/OS Projects perspective and a Local Project in Rational Developer for System z Version 7 (RDz) to assist with this task.

1. Start the RDz workbench, and specify in the Workspace: field the IMS workspace that you will be using, for example: **c:\Workspaces\IMS**, as shown in Figure 4-4. Click **OK**.


Workspace Launcher

Select a workspace

IBM WebSphere Developer for System z stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Workspaces\IMS

Browse...

Use this as the default and do not ask again

OK Cancel

Figure 4-4 Opening RDz with the IMS workspace

2. Open the z/OS Projects perspective, as shown in Figure 4-5.

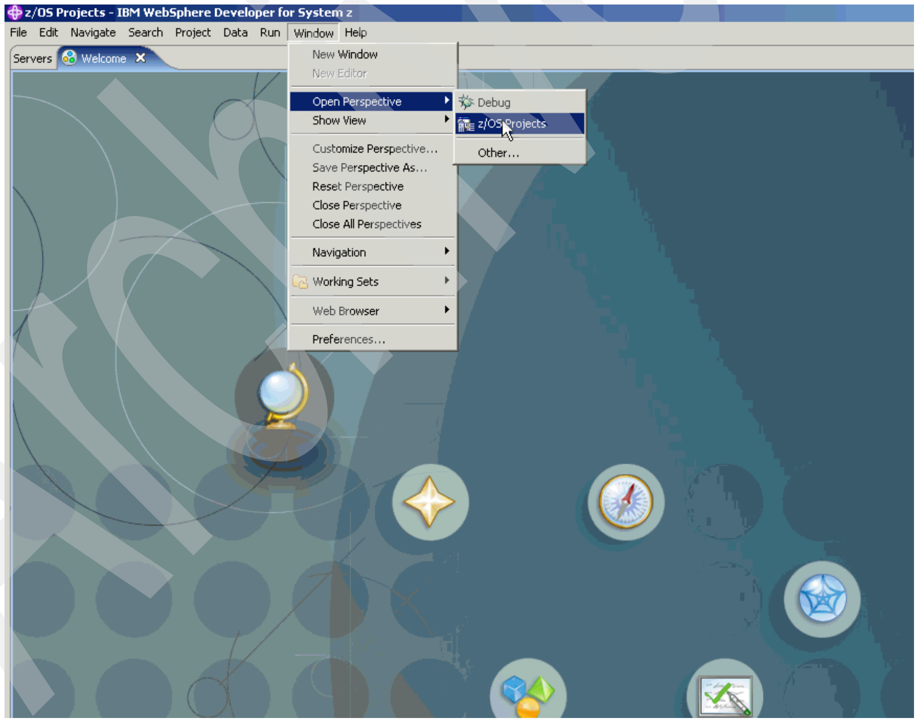


Figure 4-5 Opening the z/OS Projects perspective

3. If you do not get the z/OS Projects perspective, select the Outline view, as shown in Figure 4-6. You will then see the z/OS Projects view on the left-hand side.

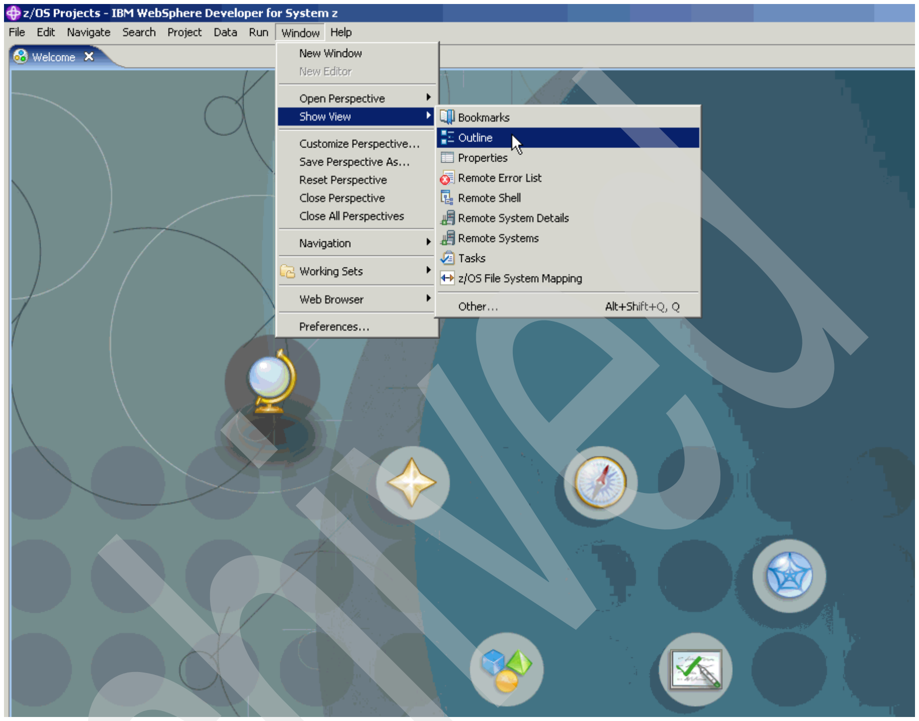
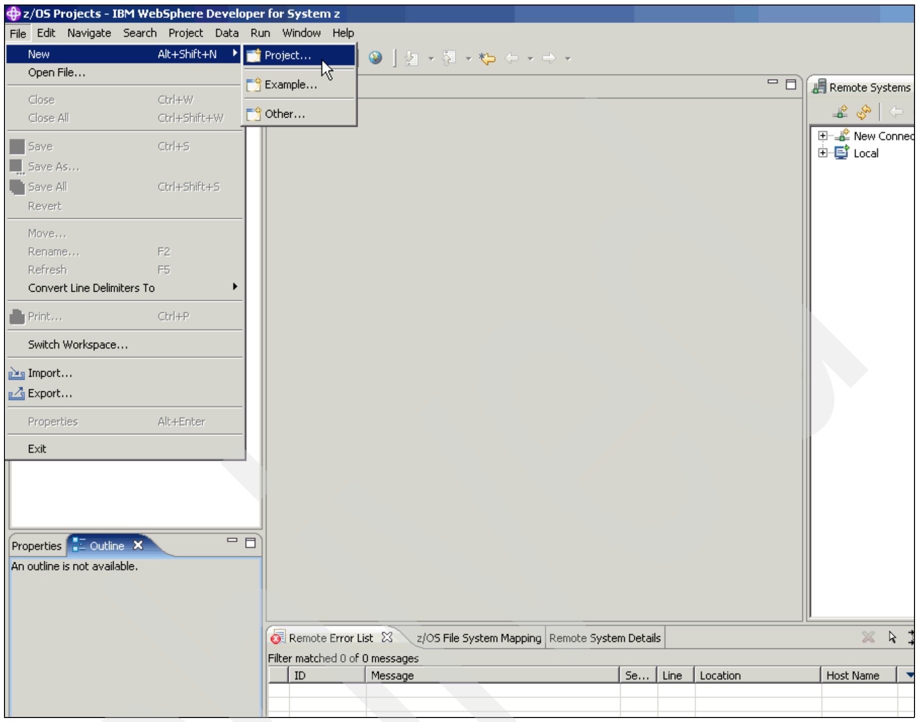


Figure 4-6 Selecting the Outline view

4. Close the Welcome panes in order to make the z/OS Projects perspectives visible.

5. Create a new project by selecting **File** → **New** → **Project**, as shown in Figure 4-7.



An outline is not available.

Filter matched 0 of 0 messages

Figure 4-7 Creating a new project

6. You will now see the Select a wizard dialog. Select **Workstation COBOL or PL/1** → **Local Project** and click **Next**. See Figure 4-8.

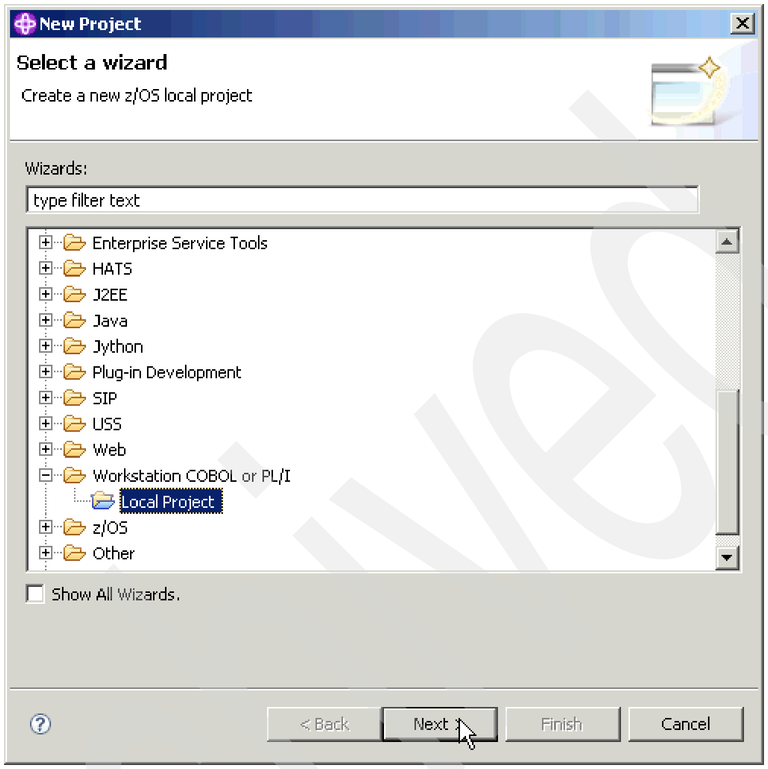
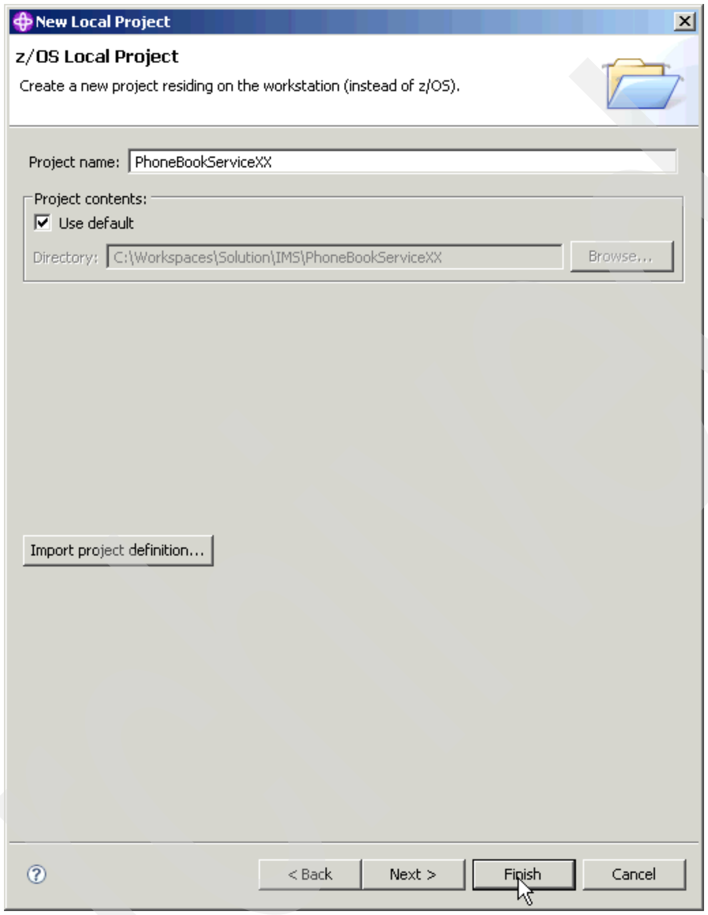


Figure 4-8 Select a wizard

7. You will now see the z/OS Local Project wizard. Specify a name for the project in the Project name field. We will be using PhoneBookServiceXX in our example. See Figure 4-9.



z/OS Local Project

Create a new project residing on the workstation (instead of z/OS).

Project name: PhoneBookServiceXX

Project contents:

☑ Use default

Directory: C:\Workspaces\Solution\IMS\PhoneBookServiceXX

Figure 4-9 z/OS Local Project wizard

8. Click **Finish**. You will now have a new empty project defined, as shown in the Outline view in the z/OS Projects perspective.

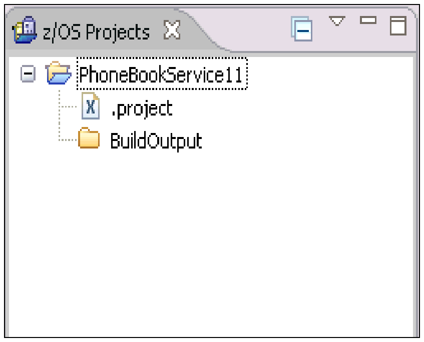


Figure 4-10 New project defined in the Outline view

9. In this step we make sure that the Navigator view is open in the workbench. In the Navigator view it is easy to see all the relevant components and to perform things like file transfers later on. Select **Window** → **Show View** → **Other**, as shown in Figure 4-11 on page 151.

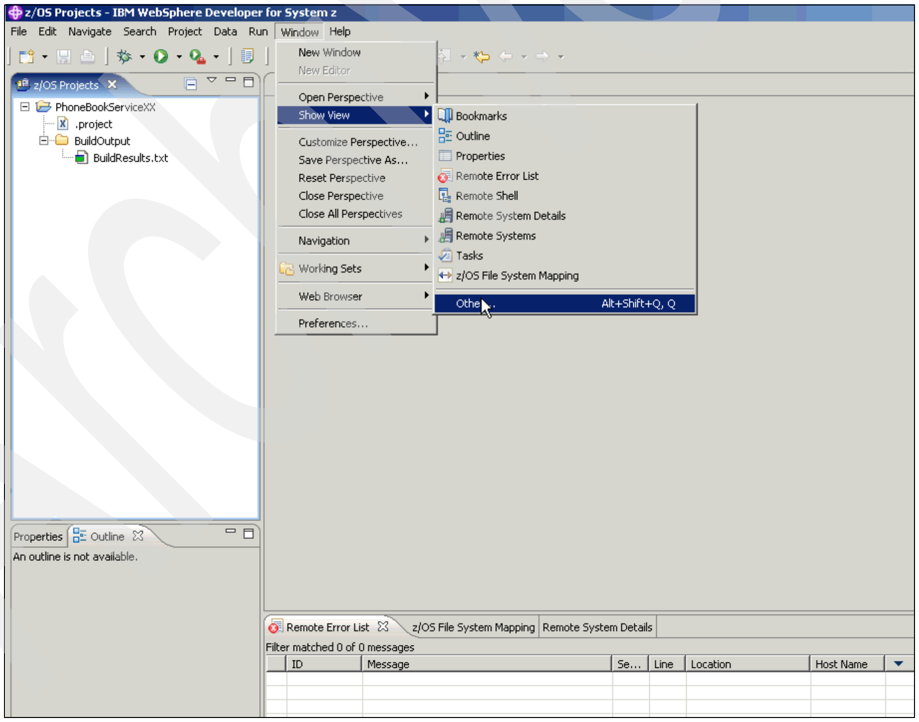


Figure 4-11 Select another view

10. In the Show View dialog, expand **General** and select **Navigator**, as shown in Figure 4-12. Click **OK**.

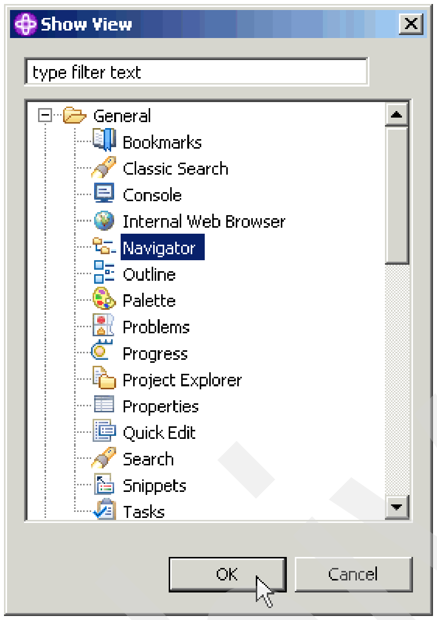


Figure 4-12 Selecting the Navigator view

You should now import into your just created local project the COBOL copybook that contains the data structures describing the input and output of the IMS application. The imported file should also be visible in the Navigator view.

11. Select the **PhoneBookServiceXX** project in the z/OS Projects Outline view. Select **File** → **Import** to open the Import wizard. In the Import wizard, expand **General**, select **File System** and click **Next**, as shown in Figure 4-13.

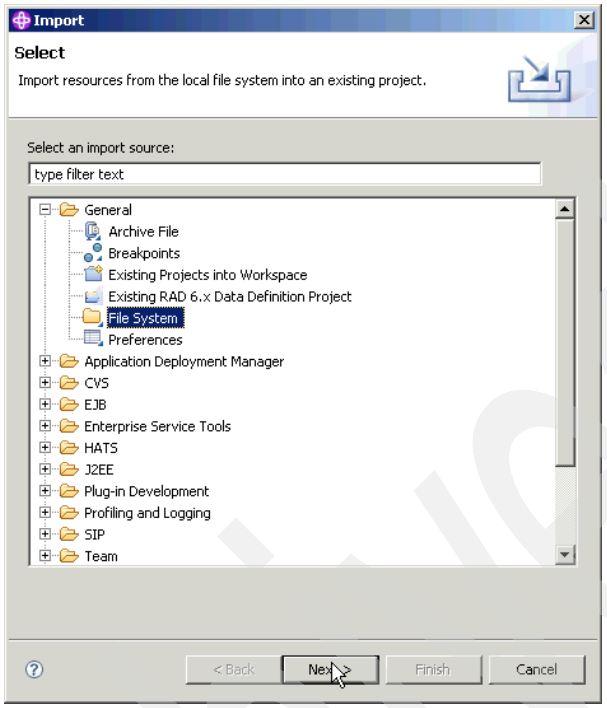


Figure 4-13 Importing from the file system

12. In the Import from File System wizard, click **Browse**. This will bring up the “Import from directory” pane. Navigate through your directory until you locate the COPYBOOK source file. In our example this file is called PHBKXX.cpy and can be found in the additional material for this book (Appendix B, “Additional material” on page 397). See Figure 4-14 on page 154.

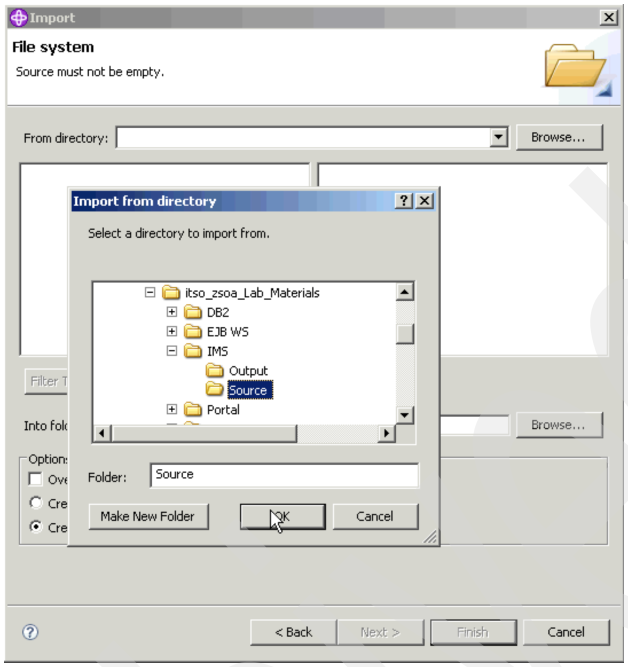
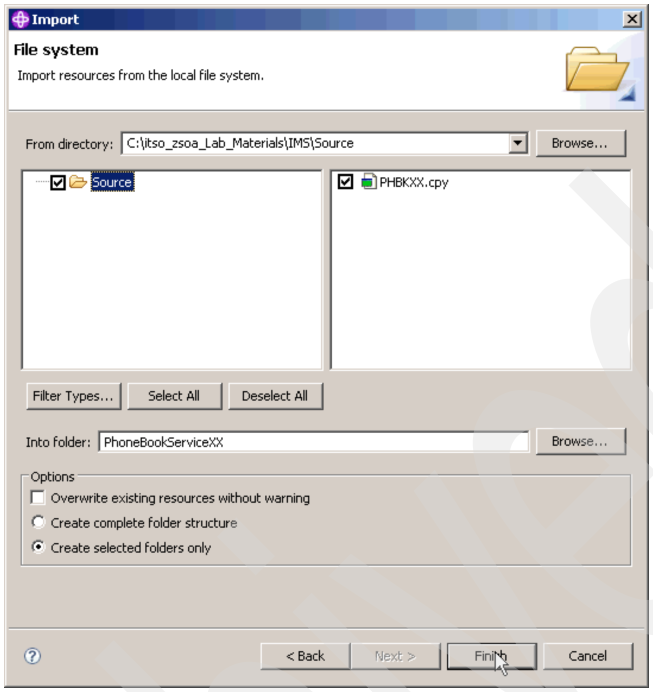


Figure 4-14 Browsing the File System

13. In the next pane, check the Source folder and make sure the PHBKXX.cpy file ONLY is checked too, as shown in Figure 4-15 on page 155. If there are any other files appearing on the right of this pane, uncheck them. Click **Finish**.

Attention: It is important to use the correct file type while importing to avoid errors later on the generation of the Web Service. When the file type is .cpy, which is our case, the importer only expects a record structure and *not* a full COBOL program.


Import

File system

Import resources from the local file system.

From directory: C:\itso_zsoa_Lab_Materials\IMS\Source

[✓] Source

[✓] PHBKXX.cpy

Into folder: PhoneBookServiceXX

Options

[ ] Overwrite existing resources without warning

( ) Create complete folder structure

(●) Create selected folders only

Figure 4-15   Importing the PHBKXX.cpy file

The file we just copied contains a COBOL-style declaration of the input and output message, as shown in Example 4-1.

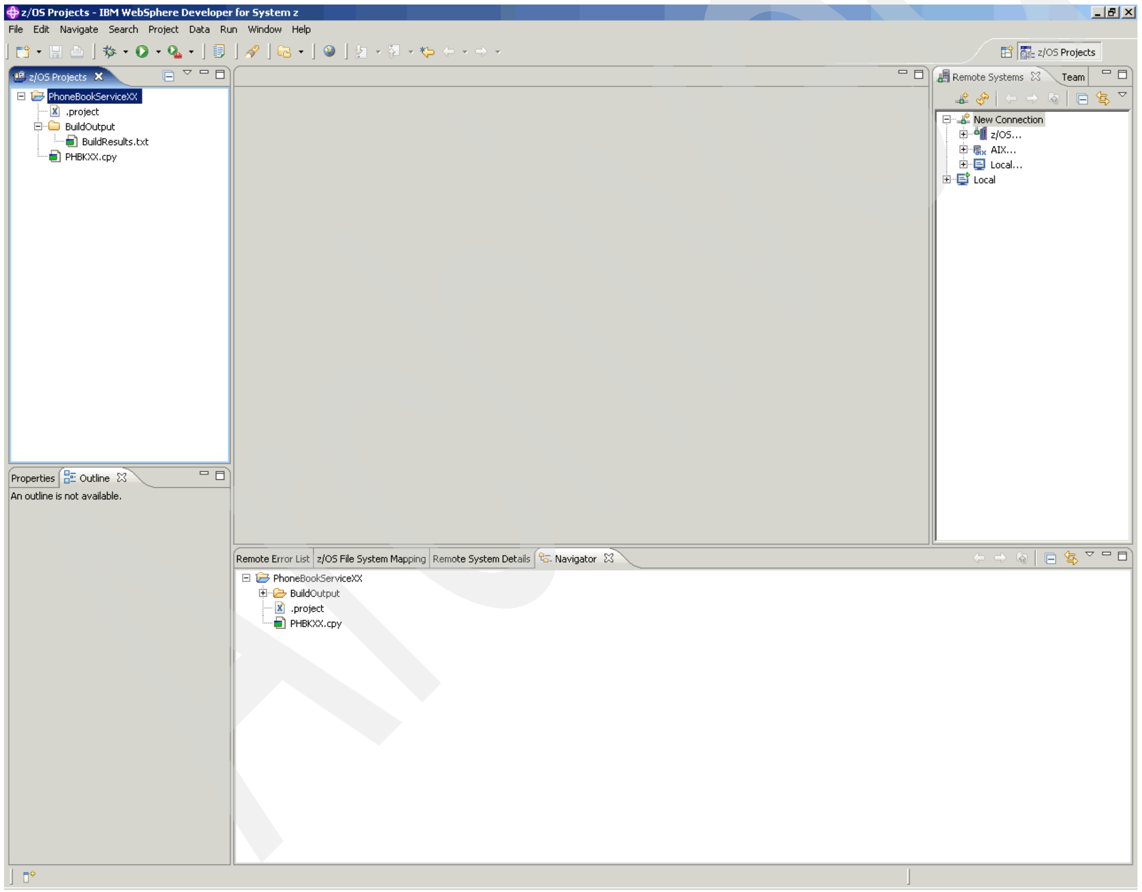
Example 4-1   IMS PhoneBook message structure

```
01 INPUT-MSG.
      02 IN-LL PICTURE S9(3) COMP.
      02 IN-ZZ PICTURE S9(3) COMP.
      02 IN-TRCD PICTURE X(10).
      02 IN-CMD PICTURE X(8).
      02 IN-NAME1 PICTURE X(10).
      02 IN-NAME2 PICTURE X(10).
      02 IN-EXTN PICTURE X(10).
      02 IN-ZIP PICTURE X(7).

01 OUTPUT-MSG.
      02 OUT-LL PICTURE S9(3) COMP VALUE +0.
      02 OUT-ZZ PICTURE S9(3) COMP VALUE +0.
      02 OUT-MSG PICTURE X(40) VALUE SPACES.
      02 OUT-CMD PICTURE X(8) VALUE SPACES.
      02 OUT-NAME1 PICTURE X(10) VALUE SPACES.
      02 OUT-NAME2 PICTURE X(10) VALUE SPACES.
```

```
02 OUT-EXTN PICTURE X(10) VALUE SPACES.
02 OUT-ZIP PICTURE X(7) VALUE SPACES.
02 OUT-SEGNO PICTURE X(4) VALUE SPACES.
```

Important: It is important that the COBOL structure follows the rules. 01 level declarations should start in column 8 of the imported file. You can double-click the PHBKXX.cpy file to open an ISPF-style editor. Any syntax errors will be shown and explained in the editor. Before you continue, make sure the COBOL copybook file does not contain any syntax errors!

14. Your workspace should now be roughly as shown in Figure 4-16.



An outline is not available.

Figure 4-16 RDz workspace after importing COBOL copybook

The next steps are to generate a Web Service definition based on the artifacts in the file you just imported.

15. In the Navigator view, right-click the **PHBKXX.cpy** file and select **Enable Enterprise Web Service** from the pop-up menu. See Figure 4-17.

Important: In this step, make sure you are in the Navigator pane. This option may not be available in any other pane.

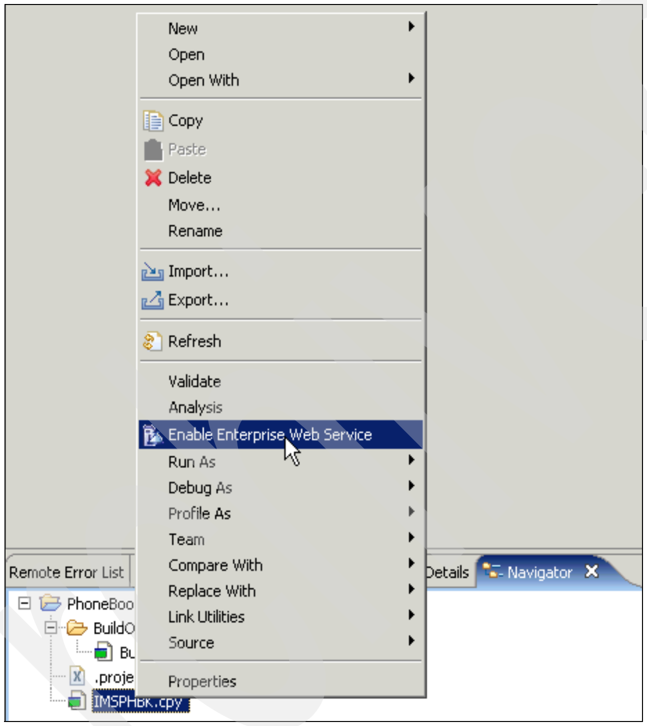


Figure 4-17 Enable Enterprise Web Service

16. This will bring up the Enterprise Service Tools Wizard Launchpad. Specify:

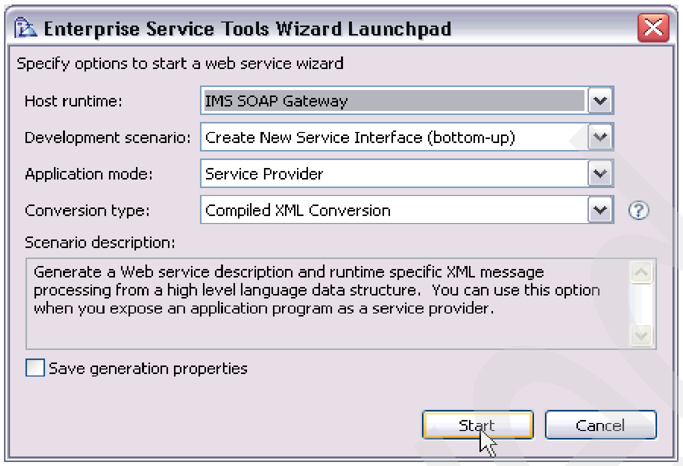
– `IMS SOAP Gateway` in the Host runtime: field

– `Create New Service Interface (bottom-up)` in the Development scenario: field

– `Compiled XML Conversion` in the Conversion type: field

This will lead to the generation of a service to be deployed to the IMS SOAP Gateway using the bottom-up approach, meaning you create WSDL based on an existing asset. See Figure 4-18 on page 158.

Click **Start**.


Enterprise Service Tools Wizard Launchpad

Specify options to start a web service wizard

Host runtime: IMS SOAP Gateway

Development scenario: Create New Service Interface (bottom-up)

Application mode: Service Provider

Conversion type: Compiled XML Conversion

Scenario description:

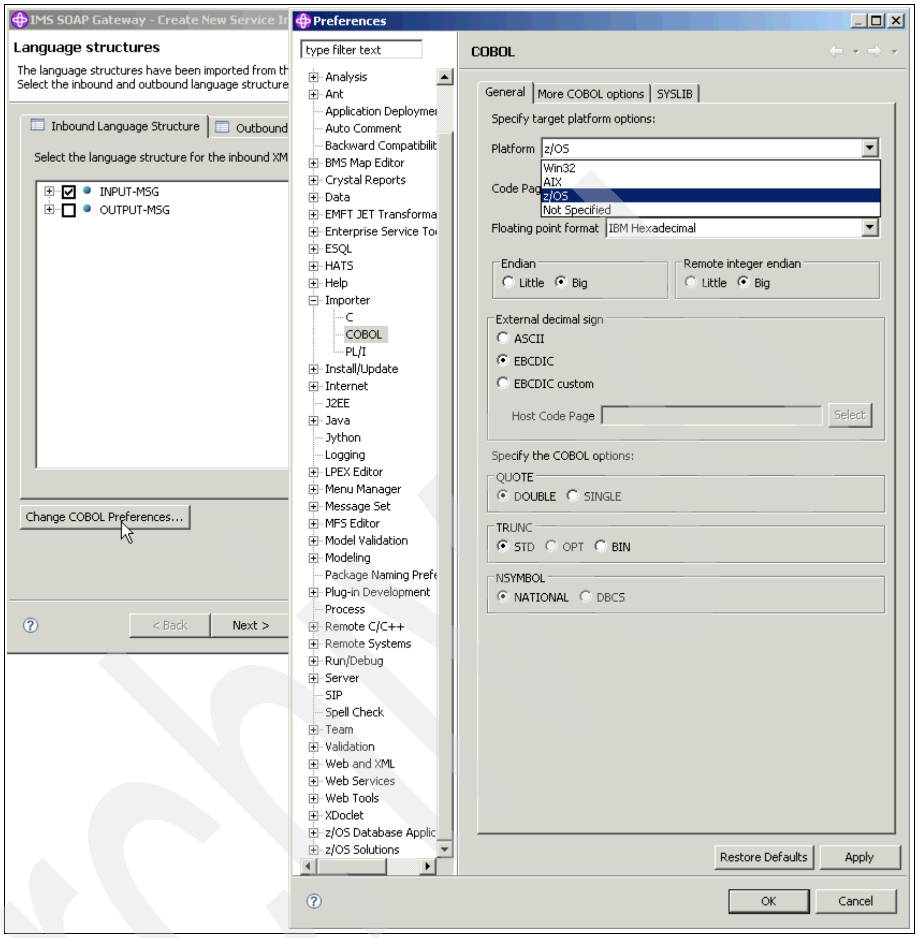
Generate a Web service description and runtime specific XML message processing from a high level language data structure. You can use this option when you expose an application program as a service provider.

☐ Save generation properties

Figure 4-18 Generation of an IMS Web Service bottom-up

17. The next pane shown is “Language structures”. We have to make sure now that we select z/OS as the runtime platform for our Web Service. Click **Change COBOL Preferences**. This brings up the Preferences pane, with focus on the **Importer** → **COBOL preferences**[2]. Select **z/OS** in the Platform drop-down list under the General tab. Click **OK**. See Figure 4-19.

[2] This pane can also be brought up at any time from the workbench by selecting Window → Preferences.


IMS SOAP Gateway - Create New Service Ir

Language structures

The language structures have been imported from th
Select the inbound and outbound language structure

Inbound Language Structure | Outbound

Select the language structure for the inbound XM

- [✓] INPUT-MSG
- [ ] OUTPUT-MSG

Change COBOL Preferences...

< Back | Next >

Preferences

type filter text

Analysis, Ant, Application Deploymer, Auto Comment, Backward Compatibilit, BMS Map Editor, Crystal Reports, Data, EMFT JET Transforma, Enterprise Service To, ESQL, HATS, Help, Importer (C, COBOL, PL/I), Install/Update, Internet, J2EE, Java, Jython, Logging, LPEX Editor, Menu Manager, Message Set, MFS Editor, Model Validation, Modeling, Package Naming Prefe, Plug-in Development, Process, Remote C/C++, Remote Systems, Run/Debug, Server, SIP, Spell Check, Team, Validation, Web and XML, Web Services, Web Tools, XDoclet, z/OS Database Applic, z/OS Solutions

COBOL

General | More COBOL options | SYSLIB

Specify target platform options:

Platform: z/OS (Win32, AIX, z/OS, Not Specified)

Code Pag

Floating point format: IBM Hexadecimal

Endian: Little [✓ Big]

Remote integer endian: Little [✓ Big]

External decimal sign: ASCII [✓ EBCDIC] EBCDIC custom

Host Code Page | Select

Specify the COBOL options:

QUOTE: [✓ DOUBLE] SINGLE

TRUNC: [✓ STD] OPT BIN

NSYMBOL: [✓ NATIONAL] DBCS

Restore Defaults | Apply

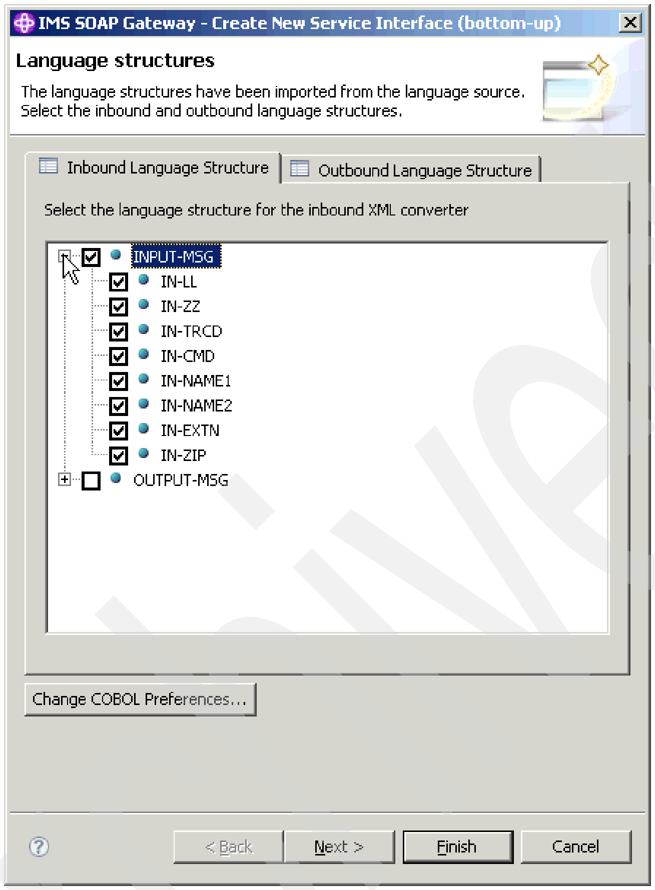
OK | Cancel

Figure 4-19 COBOL preferences

18. We are now back in the Language structures pane. We now need to specify which part of the imported data structure is input and which is output.

    Under the Inbound Language Structure tab, we select the COBOL structure that is the input structure for our IMS application. (By default the first structure defined in the program source file (PHBKxx.cpy) is selected.)

It should look like Figure 4-20 when you expand the input message.


IMS SOAP Gateway - Create New Service Interface (bottom-up)

Language structures

The language structures have been imported from the language source.
Select the inbound and outbound language structures.

Inbound Language Structure
Outbound Language Structure

Select the language structure for the inbound XML converter

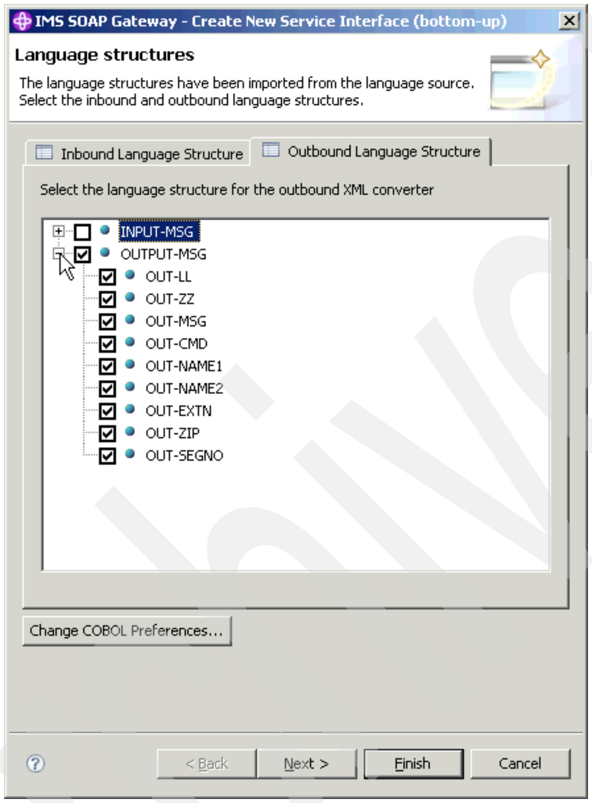
```
[✓] INPUT-MSG
    [✓] IN-LL
    [✓] IN-ZZ
    [✓] IN-TRCD
    [✓] IN-CMD
    [✓] IN-NAME1
    [✓] IN-NAME2
    [✓] IN-EXTN
    [✓] IN-ZIP
[ ] OUTPUT-MSG
```

Change COBOL Preferences...
< Back
Next >
Finish
Cancel

Figure 4-20 Input message

In the Outbound Language Structure tab, we select the COBOL structure that is the output structure for our IMS application. You need to explicitly uncheck the input message and check the output message!

It should look like Figure 4-21 when you expand the output message.


IMS SOAP Gateway - Create New Service Interface (bottom-up)

Language structures

The language structures have been imported from the language source.
Select the inbound and outbound language structures.

Inbound Language Structure | Outbound Language Structure

Select the language structure for the outbound XML converter

- [ ] INPUT-MSG
- [x] OUTPUT-MSG
  - [x] OUT-LL
  - [x] OUT-ZZ
  - [x] OUT-MSG
  - [x] OUT-CMD
  - [x] OUT-NAME1
  - [x] OUT-NAME2
  - [x] OUT-EXTN
  - [x] OUT-ZIP
  - [x] OUT-SEGNO

Figure 4-21 Output message

19. Click **Next**. This brings you to the Generation Options pane.

In the XML Converters tab, we specify the following:

– Converter program name prefix: `PHBKXX`

– Author name: `WD4Z` or another name you like to use

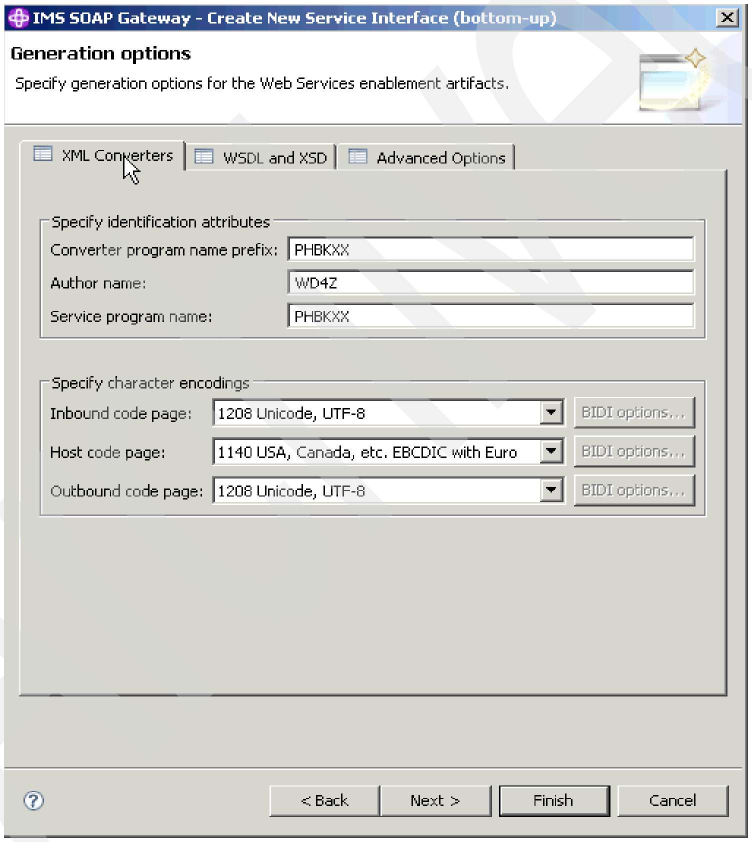
– Service program name: `PHBKXX`

– Inbound code page: select **1208 Unicode, UTF-8**

– Host code page: select the *code page used by the host* (in our case z/OS)

– Outbound code page: select **1208 Unicode, UTF-8**

See Figure 4-22.



Generation options

Specify generation options for the Web Services enablement artifacts.

Specify identification attributes

Converter program name prefix: PHBKXX

Author name: WD4Z

Service program name: PHBKXX

Specify character encodings

Inbound code page: 1208 Unicode, UTF-8

Host code page: 1140 USA, Canada, etc. EBCDIC with Euro

Outbound code page: 1208 Unicode, UTF-8

Figure 4-22 XML Converters tab in Generation options

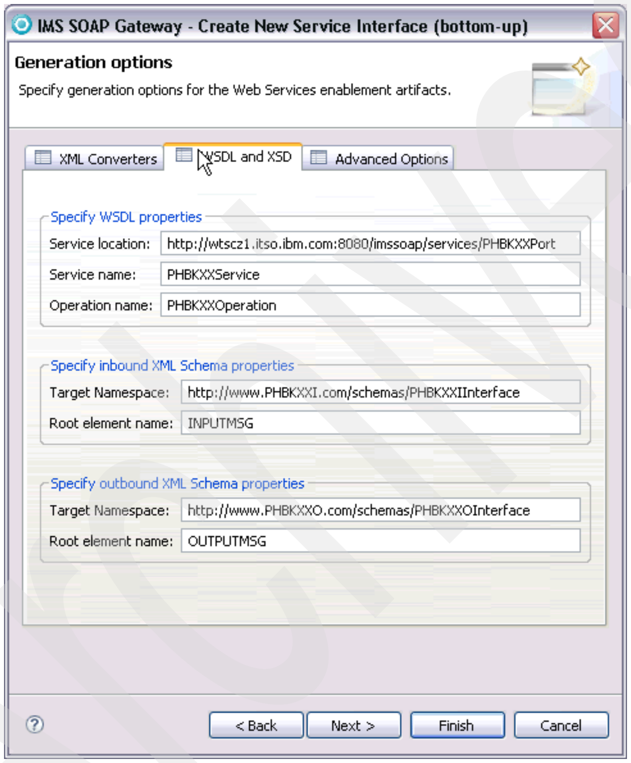
In the WSDL and XSD tab, you should:

– Change the value of the hostname and portnumber in the Service location field to the hostname and portnumber where you are running the IMS SOAP Gateway.

– Make sure that the part after the last / in the URI has the value PHBKXXPort.

– Leave the defaults for the other fields under this tab.

See Figure 4-23 for the final result.


IMS SOAP Gateway - Create New Service Interface (bottom-up)

Generation options

Specify generation options for the Web Services enablement artifacts.

Specify WSDL properties

Service location: http://wtsc21.itso.ibm.com:8080/imssoap/services/PHBKXXPort

Service name: PHBKXXService

Operation name: PHBKXXOperation

Specify inbound XML Schema properties

Target Namespace: http://www.PHBKXXI.com/schemas/PHBKXXIInterface

Root element name: INPUTMSG

Specify outbound XML Schema properties

Target Namespace: http://www.PHBKXXO.com/schemas/PHBKXXOInterface

Root element name: OUTPUTMSG

Figure 4-23 WSDL and XSD tab in Generation options

It is not necessary at this point to change anything in the “Advanced options” tab, so you can click **Next** now.

This brings you to the IMS SOAP Gateway Web Service Provider pane.

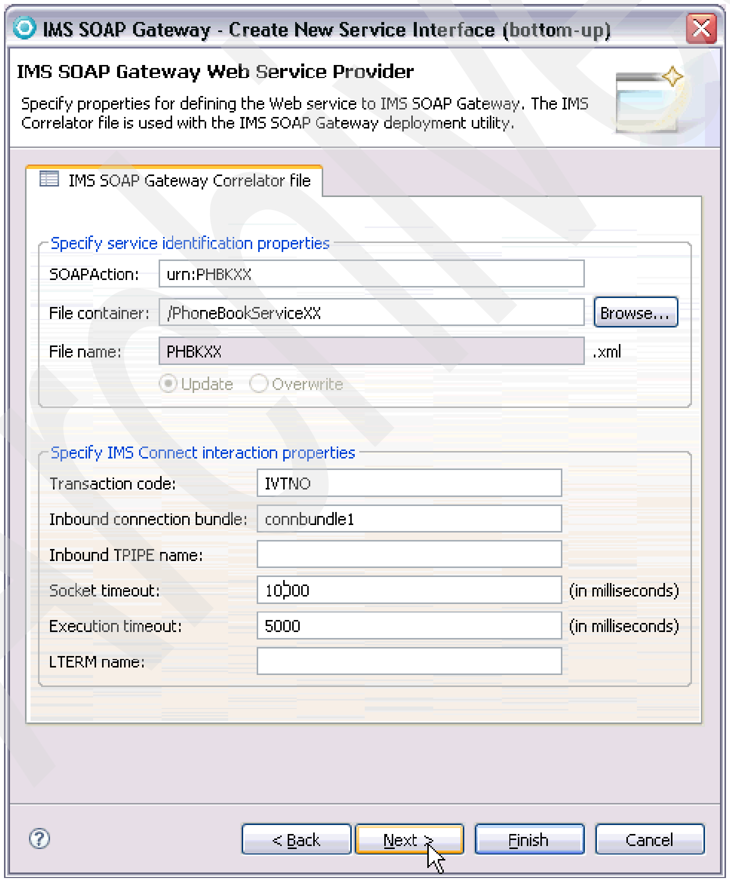
20. Use Figure 4-24 to guide you in making sure the fields have all the right values.

Notes:

- Socket Timeout: 10000 (value specified in milliseconds; a value must be entered or a socket will remain connected to IMS Connect until Connect is stopped. The value can be changed later using the IMS SOAP Gateway Deployment utility).

  Execution Timeout: 10000 (same as above).
- Notice the Connection bundle name. We will be referring to this name (connbundle1) in the deployment step.

Click **Next**.


IMS SOAP Gateway - Create New Service Interface (bottom-up)
IMS SOAP Gateway Correlator file
Browse...
< Back
Next >
Finish
Cancel

IMS SOAP Gateway Web Service Provider

Specify properties for defining the Web service to IMS SOAP Gateway. The IMS Correlator file is used with the IMS SOAP Gateway deployment utility.

Specify service identification properties

SOAPAction:	urn:PHBKXX	
File container:	/PhoneBookServiceXX	
File name:	PHBKXX	.xml

(● Update ○ Overwrite)

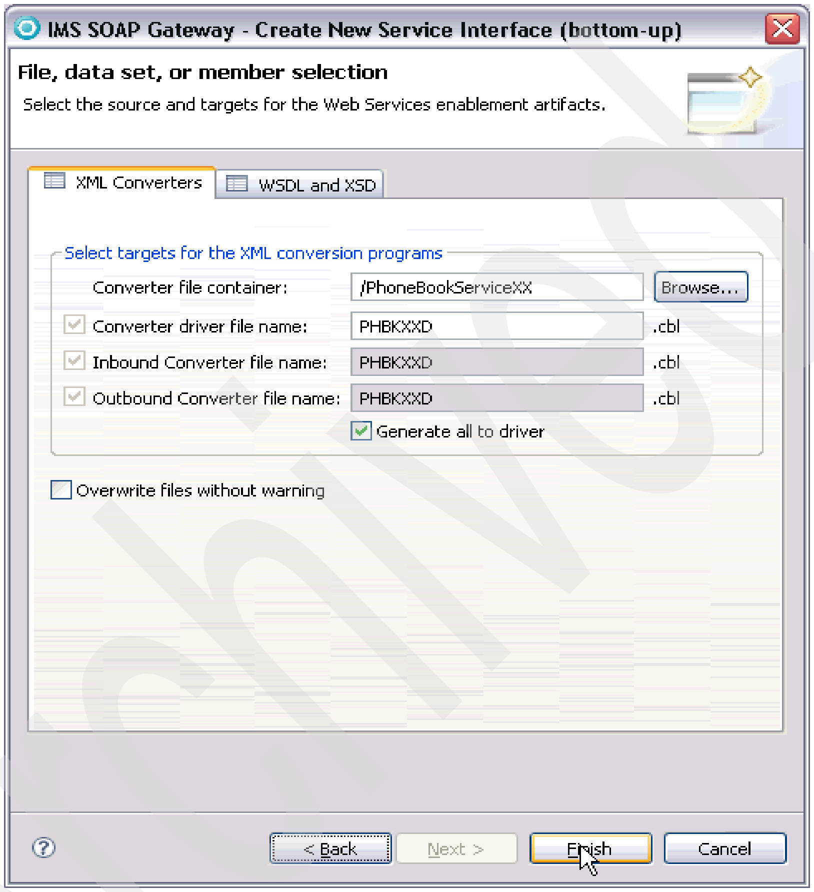
Specify IMS Connect interaction properties

Transaction code:	IVTNO	
Inbound connection bundle:	connbundle1	
Inbound TPIPE name:		
Socket timeout:	10000	(in milliseconds)
Execution timeout:	5000	(in milliseconds)
LTERM name:		

Figure 4-24 IMS SOAP Gateway generation properties

21. You will now arrive at the “File, data set or member selection” pane. Make sure that the following are specified:

   – In the XML Converters tab, use the defaults, as shown in Figure 4-25.



IMS SOAP Gateway - Create New Service Interface (bottom-up)

File, data set, or member selection

Select the source and targets for the Web Services enablement artifacts.

Select targets for the XML conversion programs

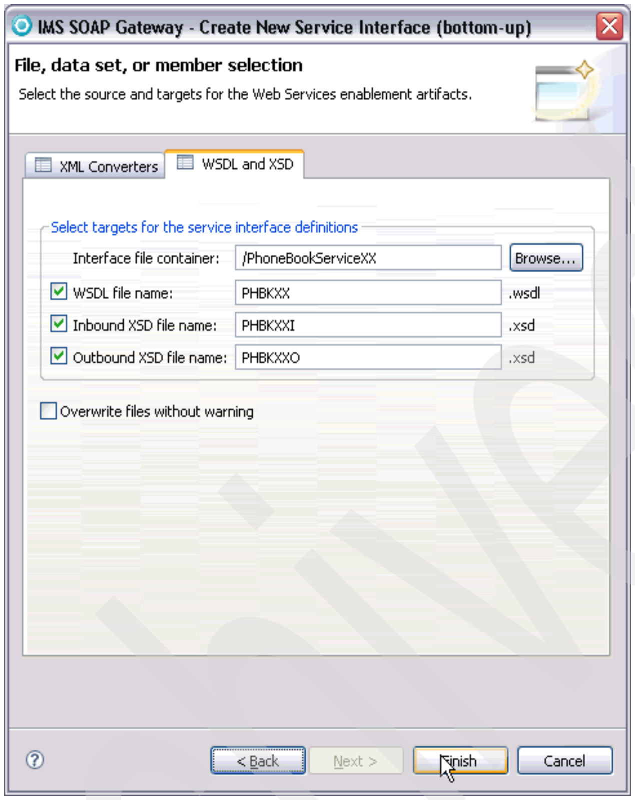
Converter file container:	/PhoneBookServiceXX	Browse...
✓ Converter driver file name:	PHBKXXD	.cbl
✓ Inbound Converter file name:	PHBKXXD	.cbl
✓ Outbound Converter file name:	PHBKXXD	.cbl

✓ Generate all to driver

☐ Overwrite files without warning

Figure 4-25   XML Converters tab

– In the WSDL and XSD tab, use the defaults also, as shown in Figure 4-26.


IMS SOAP Gateway - Create New Service Interface (bottom-up)

File, data set, or member selection

Select the source and targets for the Web Services enablement artifacts.

Select targets for the service interface definitions

Interface file container: /PhoneBookServiceXX

[✓] WSDL file name: PHBKXX .wsdl

[✓] Inbound XSD file name: PHBKXXI .xsd

[✓] Outbound XSD file name: PHBKXXO .xsd

[ ] Overwrite files without warning

Figure 4-26 WSDL and XSD tab

Click **Finish**. We now have various .xml, .wsdl and .xsd files generated in the workspace. The components of the Web Service have started to exist now:

– The PHBKXX.wsdl file contains the bindings and other artifacts of the Web Service. You can take a look at it by just right-clicking it.

– The PHBKXX.xml file is the *correlator* file. It contains the codepage definitions and other artifacts that you entered when generating the Web Service.

– The PHBKXXD.cbl file is the driver and runtime XML conversion program. It is a full-size COBOL program generated by the tooling.

– The two .xsd files represent the input and output message structures.

4.5 Deploying the service using the IMS SOAP Gateway Deployment utility

Now that all artifacts have been generated, we are ready to deploy the Web Service to the IMS SOAP Gateway on z/OS. To be able to use the IMS SOAP Gateway on z/OS, certain configuration steps are required on z/OS, in the IMS SOAP gateway and IMS Connect. We will not discuss those here and refer you to the IMS documentation at:

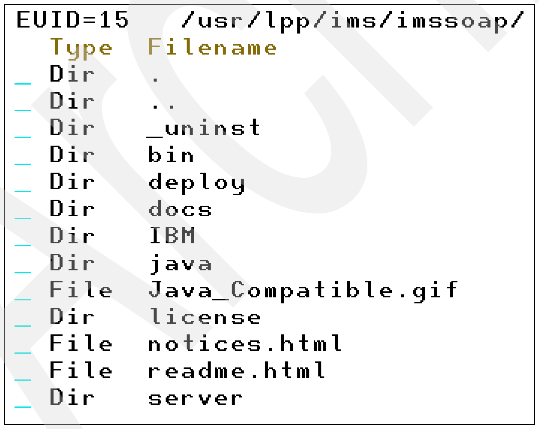
`http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.etools.soap10.doc/soapoverview.htm`

This is an outline of the steps to be performed to deploy the Web Service:

1. Store the .wsdl file and .xml files on z/OS in the required subdirectories.
2. Invoke the shell script of the deployment utility on z/OS.
3. Use the IMS SOAP Gateway admin console to verify the successful deployment of the Web Service.

4.5.1 Uploading generated artifacts to z/OS

In the next step, the IMS SOAP Gateway required artifacts are made available on z/OS. The IMS SOAP Gateway on z/OS has a UNIX directory structure, as shown in Figure 4-27.



```
EUID=15   /usr/lpp/ims/imssoap/
   Type   Filename
_  Dir    .
_  Dir    ..
_  Dir    _uninst
_  Dir    bin
_  Dir    deploy
_  Dir    docs
_  Dir    IBM
_  Dir    java
_  File   Java_Compatible.gif
_  Dir    license
_  File   notices.html
_  File   readme.html
_  Dir    server
```

Figure 4-27 IMS SOAP Gateway on z/OS directory structure

- The `deploy` directory contains the deployment utility.
- The `server/webapps/imssoap/wsdl` directory contains the .wsdl files of the deployed Web Services.

- The `server/webapps/imssoap/xml` directory contains the .xml (correlator) files of the deployed Web Services.

Proceed as follows:

1. Create a new connection in RDz for the z/OS system that runs the IMS SOAP Gateway. In the Remote Systems view, right-click **Connection** → **z/OS** and select **New Connection** from the pop-up menu, as shown in Figure 4-28.

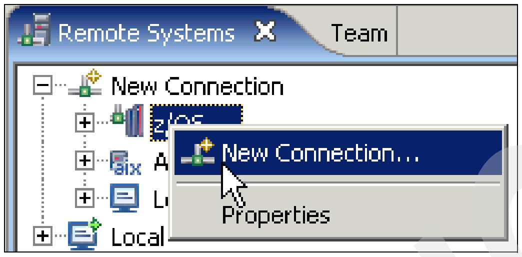
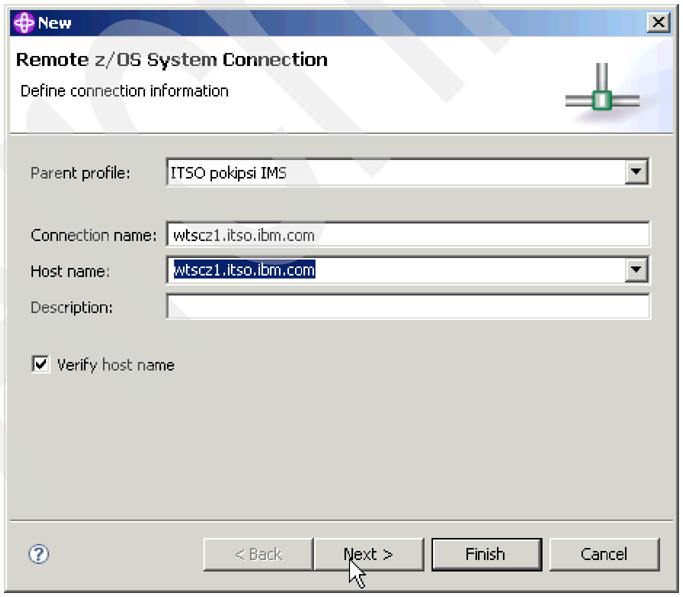


Figure 4-28 Creating a new connection to z/OS

2. In the next pane, fill in a connection profile name, for example WTSCZ1. Click **Next**.
3. In the Remote z/OS System Connection pane, fill in the fields as shown in Figure 4-29. Make sure you use your own hostname. Click **Next**.



Remote z/OS System Connection
Define connection information

Parent profile: ITSO pokipsi IMS
Connection name: wtscz1.itso.ibm.com
Host name: wtscz1.itso.ibm.com
Description:
[✓] Verify host name

Figure 4-29 Filling in the connection details

4. In the JES pane, for the JES Job Monitor Port field, ensure that the port field is set to the value as specified by your systems programmer and click **Next**.
5. In the MVS Files pane, ensure that the **Daemon Port** field is set to the value specified by your systems programmer and click **Finish**.
6. In the USS files pane, leave the details and click **Finish**.
7. Right-click the new connection you just defined and select **Connect** from the pop-up menu, as shown in Figure 4-30.

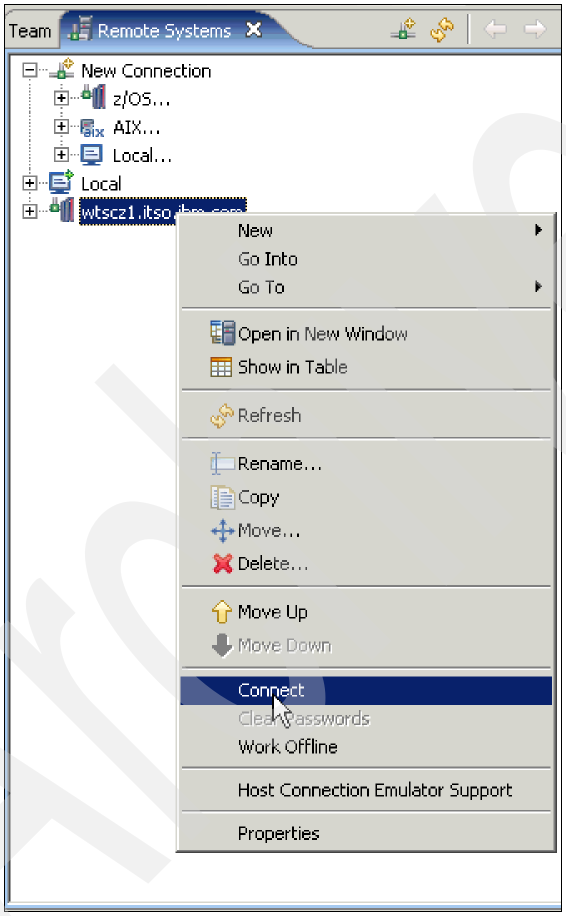
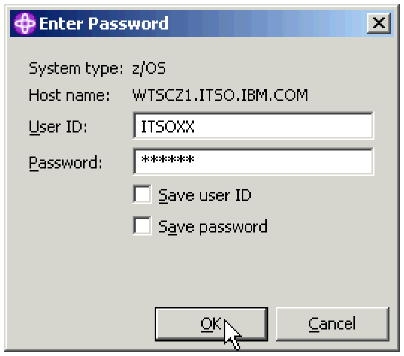


Figure 4-30 Connect to the z/OS host

8. In the next pane, specify your z/OS user ID and password, as shown in Figure 4-31.



System type: z/OS
Host name: WTSCZ1.ITSO.IBM.COM
User ID: ITSOXX
Password: ******
Save user ID
Save password

Figure 4-31 Entering user ID and password

Once you are connected you can navigate through the z/OS assets in a "graphical" way, by expanding or collapsing folders in the tree in the Remote Systems view. See Figure 4-32.

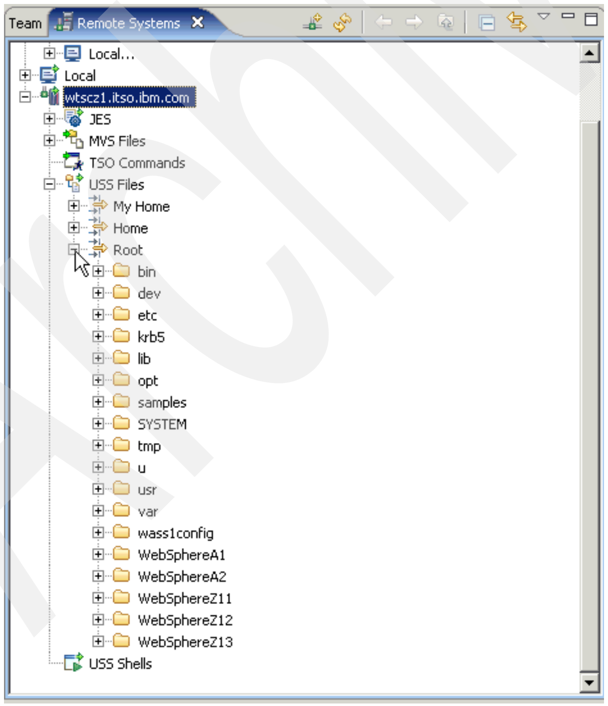


Figure 4-32 Navigating z/OS

The next step is to upload the required artifacts from the RDz workbench to the z/OS host. We will do this by dragging and dropping files from the z/OS Projects Outline view to the Remote Systems view. The first one we will do is the .wsdl file.

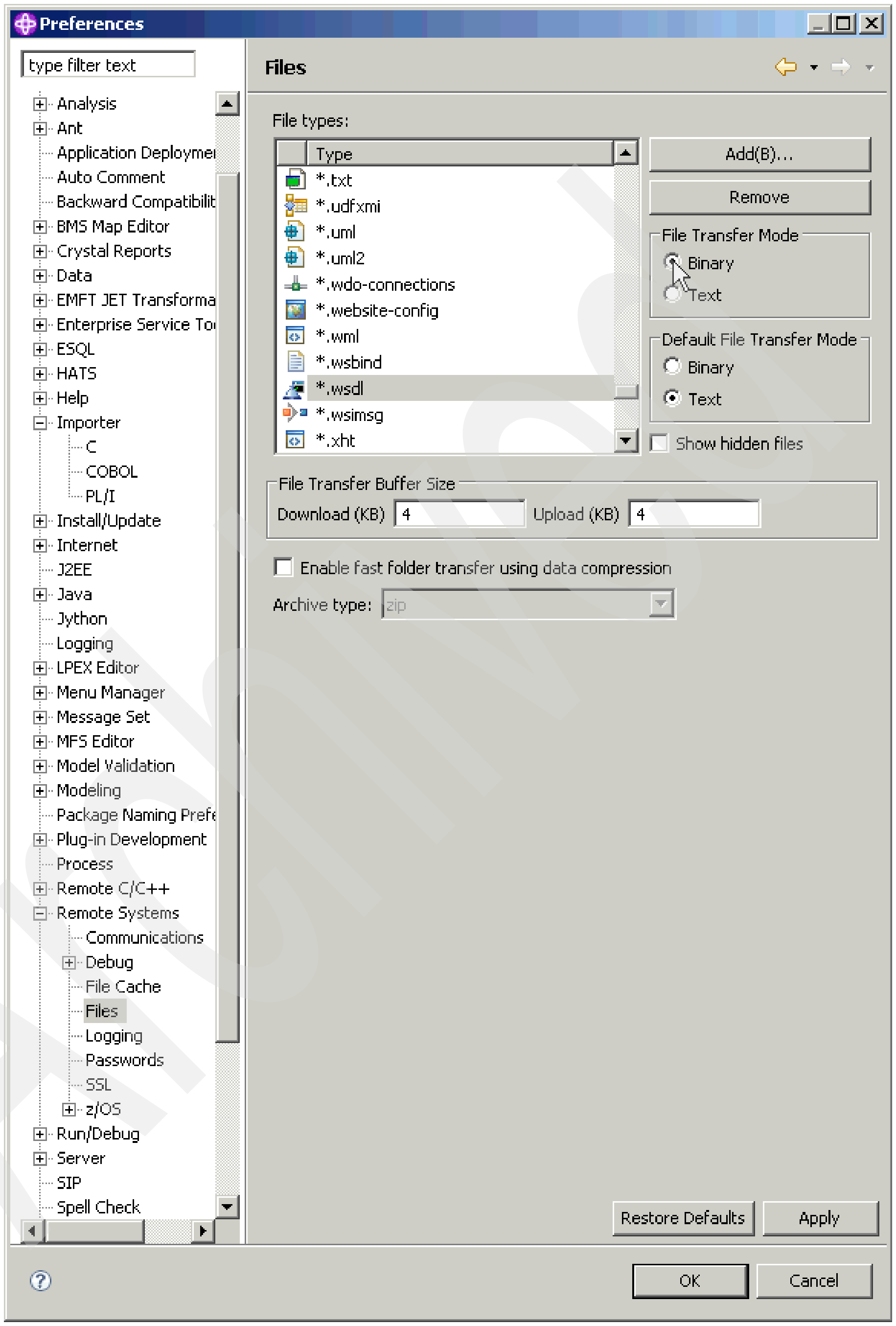
In the next steps we will make sure that the files are transferred in the correct mode:

- The .wsdl file needs to be transferred in binary mode (while the default in RDz is text).
- The .xml file also needs to be transferred in binary mode.

9. Select **Window** → **Preferences** to open the RDz preferences pane.
10. Expand **Remote systems** and click **Files**.
11. In the File types, scroll down and select **.wsdl**. Click the radio button in front of Binary in the File Transfer Mode section. Click **Apply**.

    Perform the same for files with file type .xml, because we need to transfer an .xml file later, too.

    When you have finished, click **OK**. See Figure 4-33 on page 172.



Preferences

type filter text

Analysis
Ant
Application Deployme
Auto Comment
Backward Compatibilit
BMS Map Editor
Crystal Reports
Data
EMFT JET Transforma
Enterprise Service To
ESQL
HATS
Help
Importer
- C
- COBOL
- PL/I
Install/Update
Internet
J2EE
Java
Jython
Logging
LPEX Editor
Menu Manager
Message Set
MFS Editor
Model Validation
Modeling
Package Naming Prefe
Plug-in Development
Process
Remote C/C++
Remote Systems
- Communications
- Debug
- File Cache
- Files
- Logging
- Passwords
- SSL
- z/OS
Run/Debug
Server
SIP
Spell Check

Files

File types:

Type
*.txt
*.udfxmi
*.uml
*.uml2
*.wdo-connections
*.website-config
*.wml
*.wsbind
*.wsdl
*.wsimsg
*.xht

Add(B)... Remove

File Transfer Mode: [✓ Binary] Text

Default File Transfer Mode: Binary [✓ Text]

Show hidden files

File Transfer Buffer Size
Download (KB) 4 Upload (KB) 4

Enable fast folder transfer using data compression

Archive type: zip

Restore Defaults Apply

OK Cancel

Figure 4-33 Changing the file transfer mode of the .wsdl file

12. In the Remote Systems view, expand **wtscz1.itso.ibm.com** → **USS Files** → **Root** → **usr** → **lpp** → **ims** → **imssoap** → **server** → **webapps** → **imssoap**. In the z/OS Projects view, highlight the **PHBKXX.wsdl** file. See Figure 4-34.

13. Drag and drop the PHBKXX.wsdl file from the z/OS Projects view onto the wsdl directory in the expanded subdirectory in the Remote Systems view. This will cause the .wsdl file to be uploaded from the workbench to the physical UNIX directory on z/OS.

14. In the z/OS Projects view, highlight the **PHBKXX.xml** file. Drag and drop the PHBKXX.xml file from the z/OS Projects view onto the xml directory in the expanded subdirectory in the Remote Systems view. This will cause the .xml file to be uploaded from the workbench to the physical UNIX directory on z/OS.

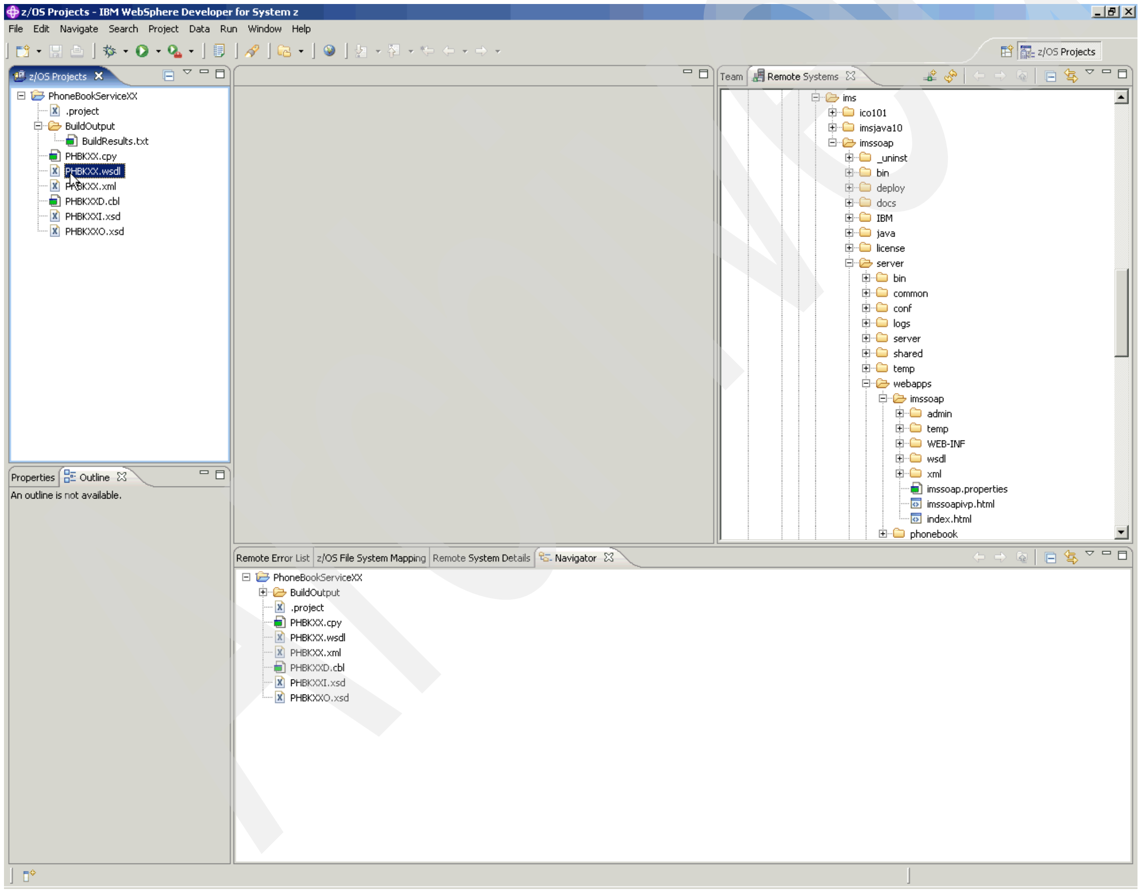


Figure 4-34 Workbench just before dragging and dropping the .wsdl file

4.5.2 Deploying the wsdl to the IMS SOAP Gateway on z/OS

Now, you will need to log on to the z/OS system to start the IMS SOAP Gateway dialog under UNIX System Services. We will execute those tasks using the host emulator support in RDz.

1. Start a USS shell. Right-click **USS Files** under the wtscz1.itso.ibm.com connection in the Remote systems view and select **Launch Shell** from the pop-up window, as shown in Figure 4-35.

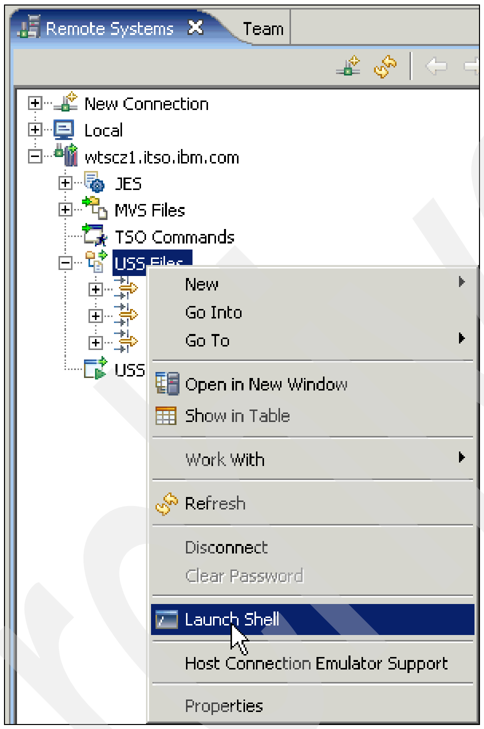


Figure 4-35 Starting a USS Shell

You now see a new window at the bottom of the workbench with two sections:

– Command Shell - Running

– Command

Commands are typed in the Command section.

2. To be able to better view the shell and the menu of the IMS SOAP Gateway deployment utility, resize the shell window or just maximize it by clicking the maximize icon in the upper-left corner, as shown in Figure 4-36.

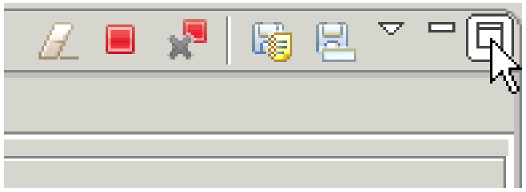
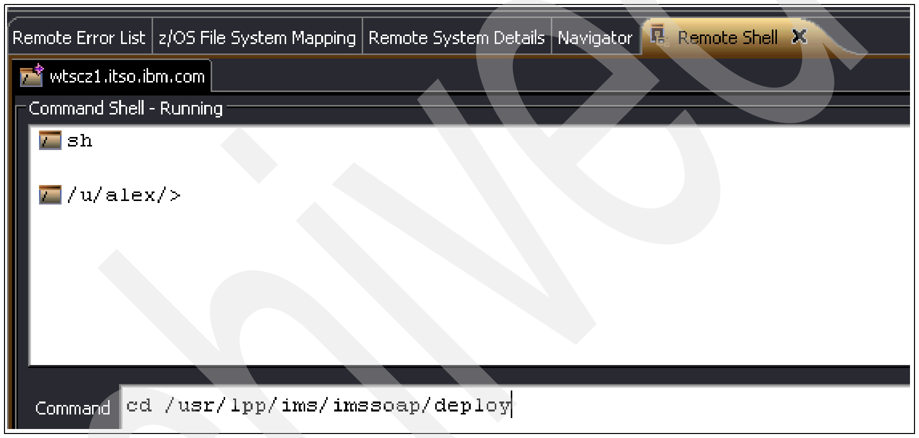


Figure 4-36 Maximizing a window in RDz

3. Switch the directory to `/usr/lpp/ims/imssoap/deploy`, by entering the command `cd /usr/lpp/ims/imssoap/deploy` in the Command section. Press Enter. See Figure 4-37.

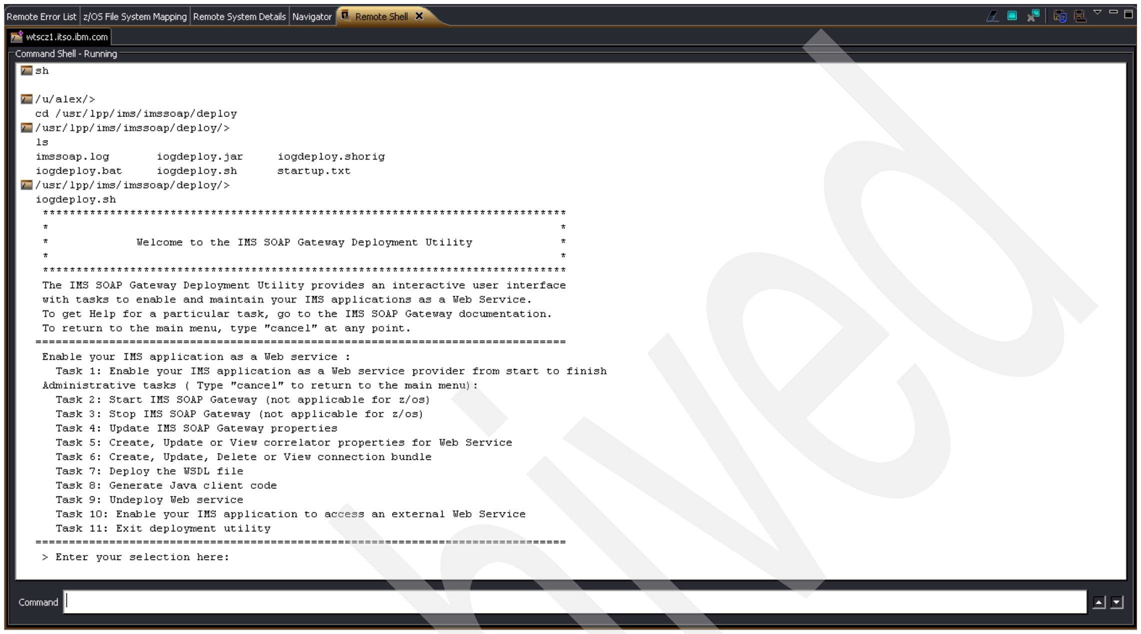


```
sh

/u/alex/>
```

```
cd /usr/lpp/ims/imssoap/deploy
```

Figure 4-37 Changing directory

4. Once switched to the directory, type `iogdeploy.sh` and press Enter. This command will start up the IMS SOAP Gateway deployment utility. You will now be presented with the menu of the IMS SOAP Gateway deployment utility, as shown in Figure 4-38.



```
sh

/u/alex/>
 cd /usr/lpp/ims/imssoap/deploy
/usr/lpp/ims/imssoap/deploy/>
 ls
 imssoap.log       iogdeploy.jar     iogdeploy.shorig
 iogdeploy.bat     iogdeploy.sh      startup.txt
/usr/lpp/ims/imssoap/deploy/>
 iogdeploy.sh
  ******************************************************************************
  *                                                                            *
  *              Welcome to the IMS SOAP Gateway Deployment Utility            *
  *                                                                            *
  ******************************************************************************
  The IMS SOAP Gateway Deployment Utility provides an interactive user interface
  with tasks to enable and maintain your IMS applications as a Web Service.
  To get Help for a particular task, go to the IMS SOAP Gateway documentation.
  To return to the main menu, type "cancel" at any point.
 ===============================================================================
  Enable your IMS application as a Web service :
    Task 1: Enable your IMS application as a Web service provider from start to finish
  Administrative tasks ( Type "cancel" to return to the main menu):
    Task 2: Start IMS SOAP Gateway (not applicable for z/os)
    Task 3: Stop IMS SOAP Gateway (not applicable for z/os)
    Task 4: Update IMS SOAP Gateway properties
    Task 5: Create, Update or View correlator properties for Web Service
    Task 6: Create, Update, Delete or View connection bundle
    Task 7: Deploy the WSDL file
    Task 8: Generate Java client code
    Task 9: Undeploy Web service
    Task 10: Enable your IMS application to access an external Web Service
    Task 11: Exit deployment utility
 ===============================================================================
  > Enter your selection here:
```

Figure 4-38 iogdeploy.sh

5. Choose option 1 by typing 1 in the Command section and press Enter. Option 1 will guide you through all the necessary steps to deploy the Web Service. When you become more familiar with the deployment utility later on, you may skip certain options and only execute the options you really need. For this example, however, we show you the entire sequence of possible options.
6. After having entered option 1, you are asked to supply the name of the .wsdl file. If you have copied the .wsdl file to the default wsdl directory of the IMS SOAP Gateway, you only need to supply the name of the file without the full directory path. Remember that we just did this in step 13 on page 173, so you only type the name of your .wsdl file (PHBKXX.wsdl) and press Enter.
7. You are now asked whether you want to see all the connection bundles (Y/N). Answer Y and press Enter. The system will display all available connection bundles. If you see the connection bundle you want to reuse, you do not need to create a new one. In our example we are reusing a connection bundle called connbundle1.

Note: The connection bundle contains the information about IMS Connect and the IMS datastore. So, this is an essential piece of information for the IMS SOAP Gateway.

8. You are now asked whether you want to create a new connection bundle. We assume you already have a connection bundle defined, so you can answer N and press Enter.
9. You are now asked to provide the name of the connection bundle to use. Type the name of the one you want to use and press Enter. In our example, the connection bundle is called connbundle1.
10. You are now asked whether you want to use an existing correlator file. You have just copied the .xml correlator file to the default xml directory of the IMS SOAP Gateway in step 14 on page 173, so you can answer Y to this question.
11. You are now asked to supply the name of the file without the full directory path. Type the name of your .xml correlator file (PHBKXX.xml) and press Enter.
12. You are now asked whether you want to view the correlator file. Answer Y and press Enter. You will now see some of the values that you specified earlier in the wizards in RDz.
13. The next question is whether you want to deploy the .wsdl file to the IMS SOAP Gateway. Answer Y and press Enter. If you see the message `WSDL successfully deployed`, you can assume everything went OK so far.
14. You are now asked whether you want to generate Java client code to invoke the Web Service. Answer Y and press Enter. We will use this client code later to test the Web Service. Also, you can incorporate this client code in a WebSphere Web application. The Web Service calls are built using the Apache AXIS framework.
15. You are now asked to enter the location of the client code. Enter a directory of choice, for example `/u/itsoxx`, and press Enter. If you see the message `Generation of client code complete`, it should be OK.

    The client code is generated using the Apache AXIS framework. This framework provides classes to create a Web Service call.
16. Check your generated client code by browsing the /com subdirectory in the directory you provided earlier, for example:

    ```
    /u/itsoxx/com
    ```

    There should be two new subdirectories now, called PHBKXXI and PHBKXXO. The PHBKXXI directory contains the class for the input message and the PHBKXXO directory contains the class for the output message.

17. Also, check the /files/target subdirectory, for example:

```
/u/itsoxx/files/target
```

There should be four new Java classes in this directory. These classes are required to make a Web Service call, too.

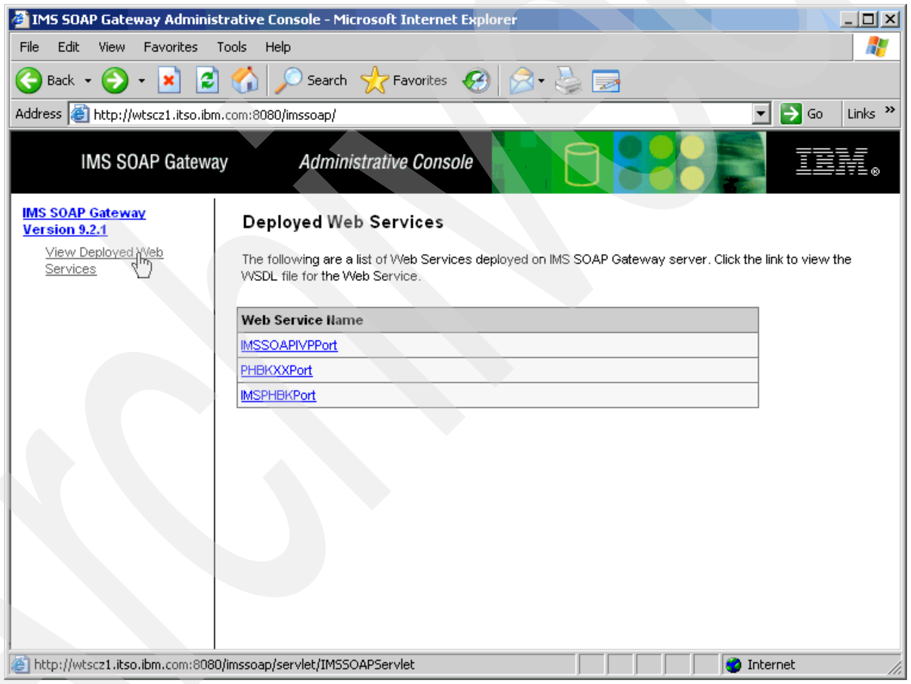
We will use these classes in 4.6, “Testing the Web Service” on page 183.

18. Finally, check successful deployment of your WSDL. Open a browser window and type in the URL of the IMS SOAP Gateway, for example:

```
http://<IMS SOAP Gateway address>/imssoap
```

Click in the left pane on View Deployed Web Services.

Your window should look like Figure 4-39.


IMS SOAP Gateway Administrative Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://wtscz1.itso.ibm.com:8080/imssoap/
Go Links
IMS SOAP Gateway Administrative Console
IBM
IMS SOAP Gateway Version 9.2.1
View Deployed Web Services
http://wtscz1.itso.ibm.com:8080/imssoap/servlet/IMSSOAPServlet
Internet

Deployed Web Services

The following are a list of Web Services deployed on IMS SOAP Gateway server. Click the link to view the WSDL file for the Web Service.

Web Service Name
IMSSOAPIVPPort
PHBKXXPort
IMSPHBKPort

Figure 4-39 IMS SOAP Gateway admin console

19. If you click a Web Service, the WSDL will be displayed.

4.5.3 Deploying and compiling the COBOL XML converter driver

In the following steps we show how to deploy the driver and runtime XML conversion program (PHBKXXD.cbl) to z/OS. There are two steps to perform:

1. Copying the COBOL source program to z/OS.
2. Compiling the program on z/OS, resulting in a COBOL load module.
   This load module will be placed in a load library that is concatenated in the STEPLIB of the IMS Connect startup PROC. This data set has to be APF-authorized.

   Execute the following steps:f you are not connected or lost your connection, reconnect, as explained in 7 on page 169.
3. Type in your user ID and password, when prompted, as explained in 8 on page 170.
4. Expand **wtscz1.itso.ibm.com** → **MVS Files** → **My Data Sets (ITSOXX.*)**. In your case you would use your own hostname and you would see your own HLQ in the My Data Sets folder.
5. Highlight the COBOL XML conversion program (in our case PHBKXXD.cbl) in the z/OS Projects View and drag/drop it onto the IMS COBOL source data set you will be using on z/OS in the Remote Systems View, as shown in Figure 4-40 on page 180.

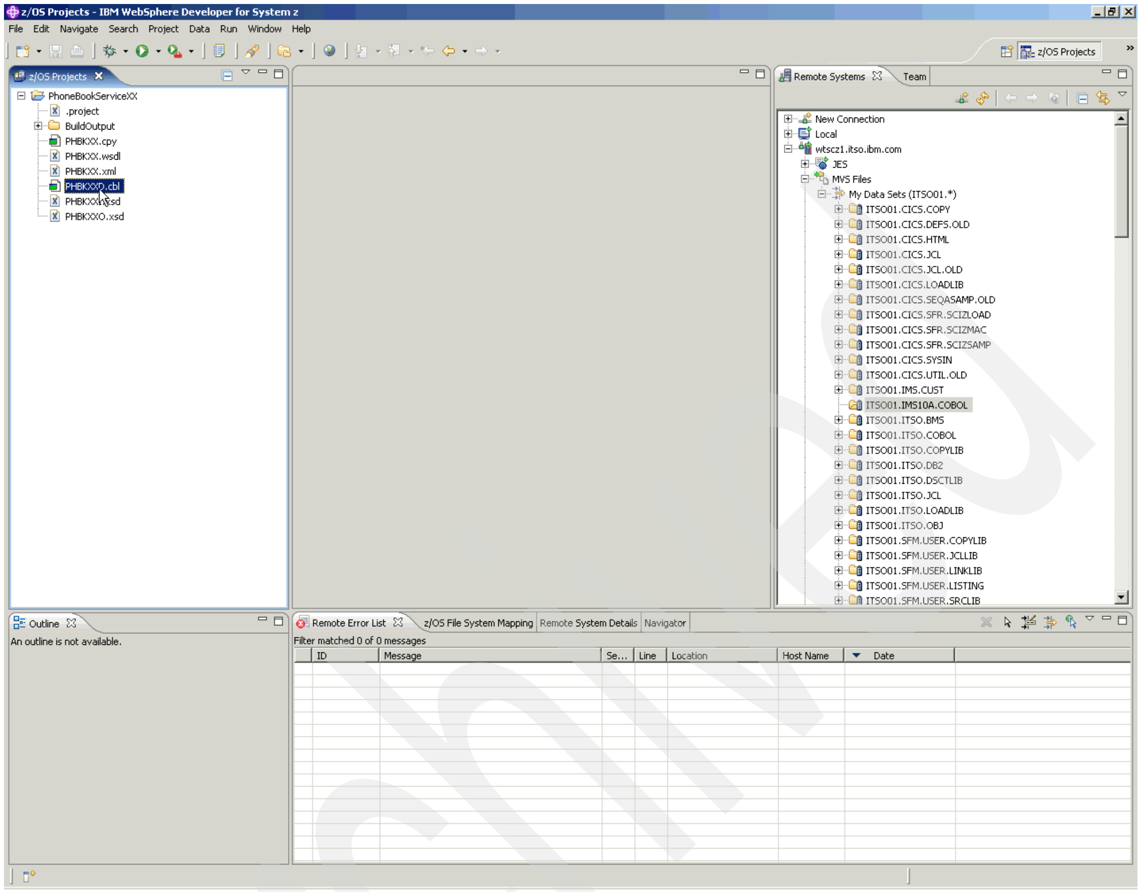


Figure 4-40 RDz workspace before dragging and dropping the PHBKXX.cbl file

At this point the COBOL source is in the IMS COBOL data set and we will now compile the COBOL source to a load module. A traditional way of doing this is by logging on to TSO and submitting the compile JCL from TSO. However, you can also submit the JCL from RDz and view the JES output in RDz.

6. In the Remote Systems View, expand your IMS JCL data set under the My Data Sets (ITSOXX.*) folder. Right-click the compile JCL member you have set up before and from the pop-up window select **Submit**, as shown in Figure 4-41. An example of the compile JCL is included in the additional material of this book.

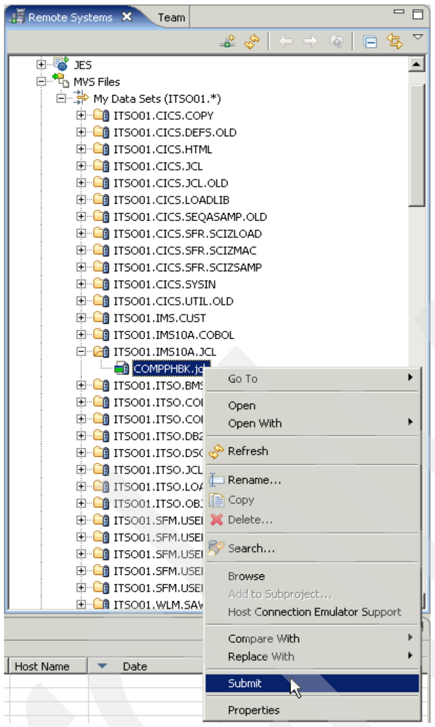
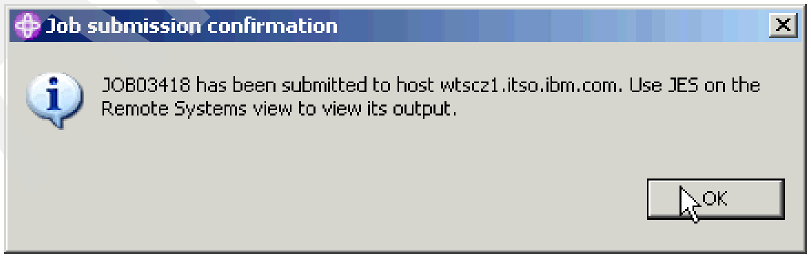


Figure 4-41 Submitting the IMS SOAP compile job

7. You will now see a message window, telling that the job is submitted. Click **OK** in this window. See Figure 4-42.



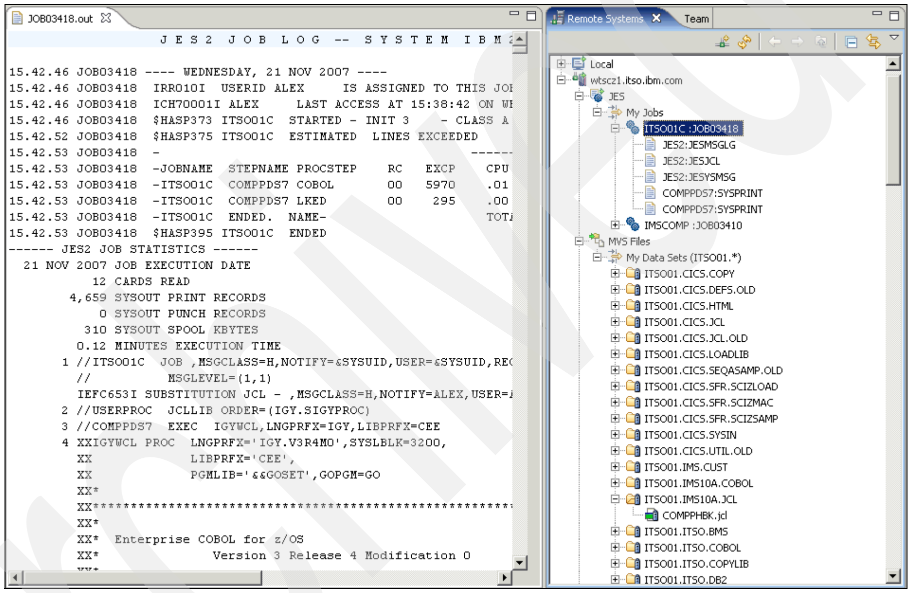
JOB03418 has been submitted to host wtscz1.itso.ibm.com. Use JES on the Remote Systems view to view its output.

Figure 4-42 Job submitted

8. After one or two minutes, your job output should be ready. In the Remote System View, expand **wtscz1.itso.ibm.com** → **JES** → **My Jobs** and check whether you already see the job output.

   Note that the window does not refresh by itself; you need to refresh the folder contents by right-clicking **JES** and selecting **Refresh** from the pop-up window.

9. If you see your job output, double-click it. A text editor window now opens showing you the job output. The return codes of both steps (compile and link-edit) should both be 0. See Figure 4-43 on page 182 for a snapshot of your workbench.



```
                    J E S 2   J O B   L O G  --  S Y S T E M   I B M 2

15.42.46 JOB03418 ---- WEDNESDAY, 21 NOV 2007 ----
15.42.46 JOB03418  IRR010I  USERID ALEX     IS ASSIGNED TO THIS JO
15.42.46 JOB03418  ICH70001I ALEX     LAST ACCESS AT 15:38:42 ON W
15.42.46 JOB03418  $HASP373 ITSO01C  STARTED - INIT 3    - CLASS A
15.42.52 JOB03418  $HASP375 ITSO01C  ESTIMATED  LINES EXCEEDED
15.42.53 JOB03418  -                                          ------
15.42.53 JOB03418  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU
15.42.53 JOB03418  -ITSO01C  COMPPDS7 COBOL       00   5970    .01
15.42.53 JOB03418  -ITSO01C  COMPPDS7 LKED        00    295    .00
15.42.53 JOB03418  -ITSO01C  ENDED.  NAME-                     TOT
15.42.53 JOB03418  $HASP395 ITSO01C  ENDED
------ JES2 JOB STATISTICS ------
  21 NOV 2007 JOB EXECUTION DATE
           12 CARDS READ
        4,659 SYSOUT PRINT RECORDS
            0 SYSOUT PUNCH RECORDS
          310 SYSOUT SPOOL KBYTES
         0.12 MINUTES EXECUTION TIME
       1 //ITSO01C  JOB ,MSGCLASS=H,NOTIFY=&SYSUID,USER=&SYSUID,RE
         //            MSGLEVEL=(1,1)
         IEFC653I SUBSTITUTION JCL - ,MSGCLASS=H,NOTIFY=ALEX,USER=
       2 //USERPROC  JCLLIB ORDER=(IGY.SIGYPROC)
       3 //COMPPDS7  EXEC  IGYWCL,LNGPRFX=IGY,LIBPRFX=CEE
       4 XXIGYWCL PROC  LNGPRFX='IGY.V3R4M0',SYSLBLK=3200,
         XX             LIBPRFX='CEE',
         XX             PGMLIB='&&GOSET',GOPGM=GO
         XX*
         XX***************************************************
         XX*
         XX*  Enterprise COBOL for z/OS
         XX*              Version 3 Release 4 Modification 0
```

Figure 4-43 Viewing the job output of the IMS SOAP compile job

What did we accomplish so far?

► We have copied the .wsdl, .xml correlator file and COBOL XML converter driver program to z/OS.

► We deployed the Web Service to the IMS SOAP Gateway using the IMS SOAP Gateway deployment utility.

► We compiled the COBOL XML converter driver program and created a load module residing in the STEPLIB of the IMS Connect address space.

In the next section we will create a Java client program driving a Web Service request to the newly deployed Web Service.

4.6 Testing the Web Service

In this last step we assemble a small Java client program to invoke the Web Service we just deployed to z/OS. We will use the client code generated in step 14 on page 177 and 15 on page 177. Also, you can use this client code to include in a WebSphere application making the call to the IMS Web Service.

The Java client classes on z/OS have been generated using the Apache AXIS framework. The steps for creating the Java client program are:

1. Copy the generated Java client classes from z/OS to the Java Project in the RDz workbench.
2. Create a Java Project in RDz.
3. Add the Apache AXIS framework to the Java Project.
4. Create the Java main program to call the Web Service.
5. Execute the Java main program.

Proceed as follows:

1. Open the Java perspective by selecting **Window** → **Open Perspective** → **Other** → **Java**.
2. Create a Java Project by selecting **File** → **New** → **Project**. In the Select a wizard pane, select **Java** → **Java Project** and click **Next**. See Figure 4-44 on page 184.

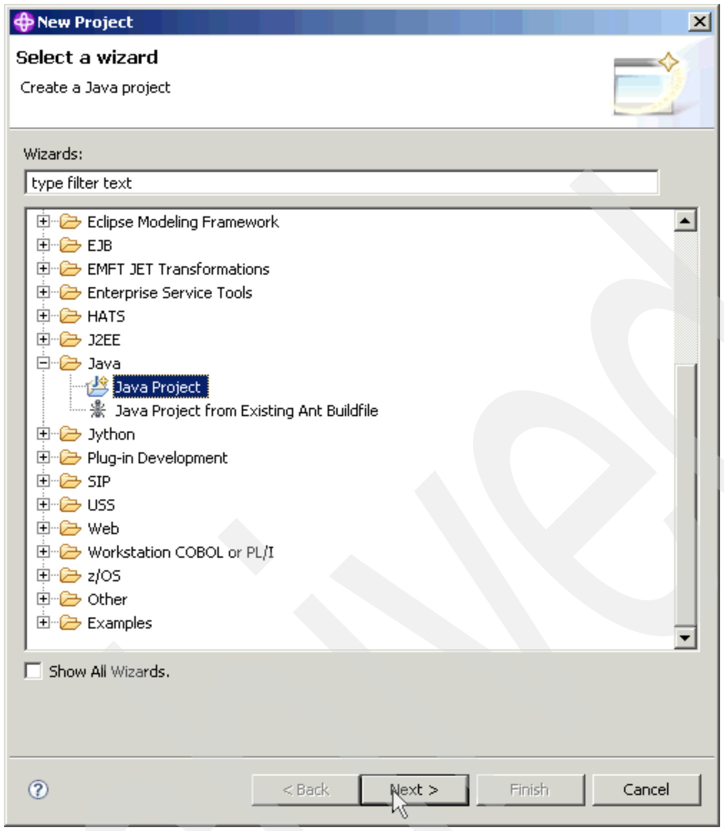
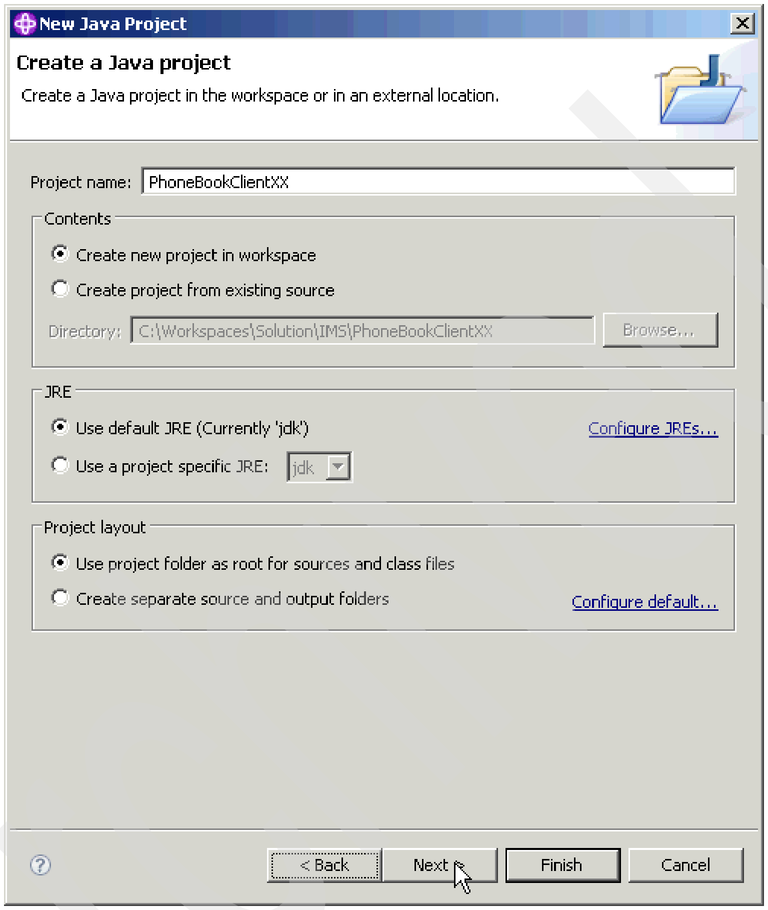


Figure 4-44 Create a new Java Project wizard

3. In the next pane, enter the name of the project and leave the defaults for all other fields. See Figure 4-45.



Create a Java project

Create a Java project in the workspace or in an external location.

Project name: PhoneBookClientXX

Contents

- [✓ Create new project in workspace]
- Create project from existing source

Directory: C:\Workspaces\Solution\IMS\PhoneBookClientXX

JRE

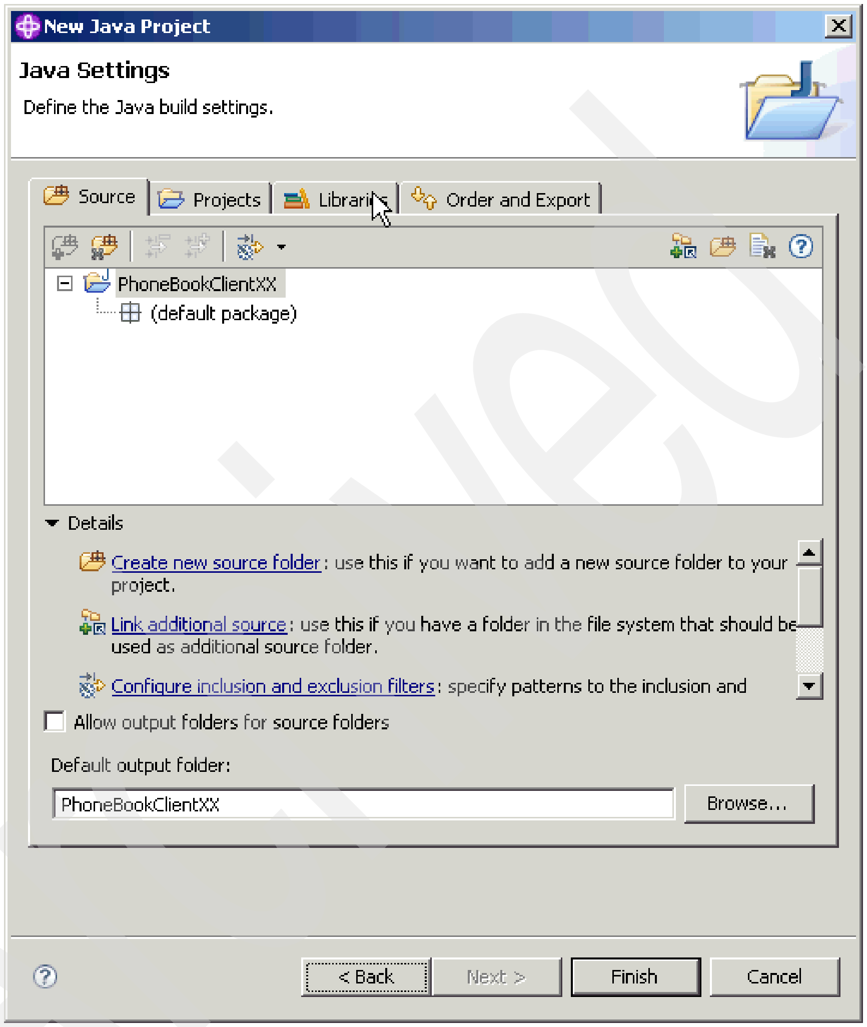
- [✓ Use default JRE (Currently 'jdk')]
- Use a project specific JRE: jdk

Project layout

- [✓ Use project folder as root for sources and class files]
- Create separate source and output folders

Figure 4-45 Enter the name of the new Java project

4. In the Java settings pane, click on the tab Java libraries, as shown in Figure 4-46.


New Java Project

Java Settings

Define the Java build settings.

PhoneBookClientXX
(default package)

Create new source folder: use this if you want to add a new source folder to your project.

Link additional source: use this if you have a folder in the file system that should be used as additional source folder.

Configure inclusion and exclusion filters: specify patterns to the inclusion and

☐ Allow output folders for source folders

Default output folder:

PhoneBookClientXX

Figure 4-46 Java settings - libraries

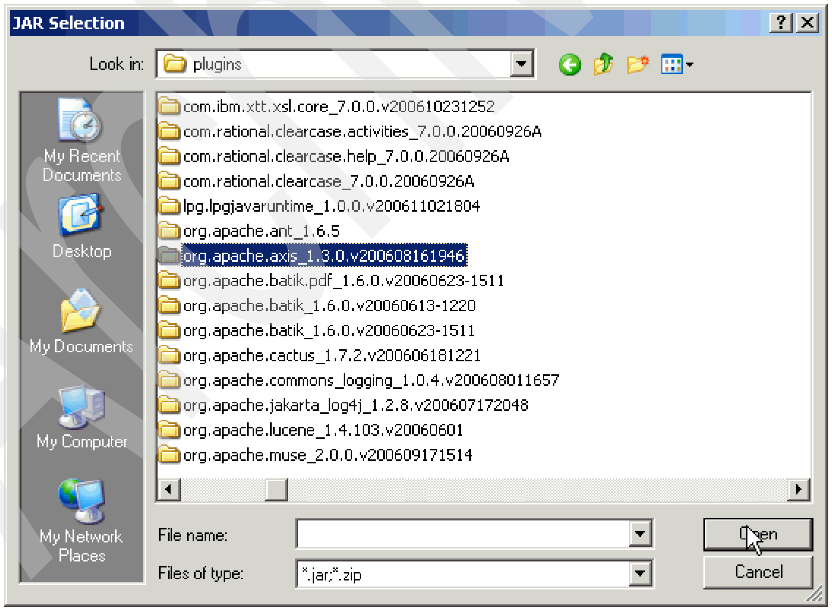
As we have explained earlier, the client stubs generated by the IMS SOAP Gateway deployment utility use Apache AXIS classes to execute a Web Services call. Because we will be creating a stand-alone Java client program in RDz that will use those stubs, you need to add the Apache AXIS libraries to the build path of this project in order to compile the Java client program without errors. The libraries needed are shown in Table 4-1 on page 187.

Table 4-1 List of jars to successfully compile

Jar name
axis.jar
jaxrpc.jar
commons-discovery-0.2.jar
commons-logging-1.0.4.jar
saaj.jar
webserviceutils.jar
wsdl4j-1.5.1.jar

These jars are available in the RDz install folder plugin for AXIS.

5. Click **Add External JARs**. This brings up a JAR selection window in which you can select the JARs to add. In our case, the JARs that we are looking for all live in the C:\Program Files\IBM\SDP70Shared\plugins directory. Select **org.apache.axis_1.3.0.v200608161946**, as shown in Figure 4-47. Click **Open**.



com.ibm.xtt.xsl.core_7.0.0.v200610231252
com.rational.clearcase.activities_7.0.0.20060926A
com.rational.clearcase.help_7.0.0.20060926A
com.rational.clearcase_7.0.0.20060926A
lpg.lpgjavaruntime_1.0.0.v200611021804
org.apache.ant_1.6.5
org.apache.axis_1.3.0.v200608161946
org.apache.batik.pdf_1.6.0.v20060623-1511
org.apache.batik_1.6.0.v20060613-1220
org.apache.batik_1.6.0.v20060623-1511
org.apache.cactus_1.7.2.v200606181221
org.apache.commons_logging_1.0.4.v200608011657
org.apache.jakarta_log4j_1.2.8.v200607172048
org.apache.lucene_1.4.103.v20060601
org.apache.muse_2.0.0.v200609171514

Figure 4-47 Selecting the Apache AXIS libraries

6. In the next pane, select **lib**.
7. In the next pane select the files as highlighted in Figure 4-48. Click **Open**. Five of the seven required jar files have now been added to the Java build path.

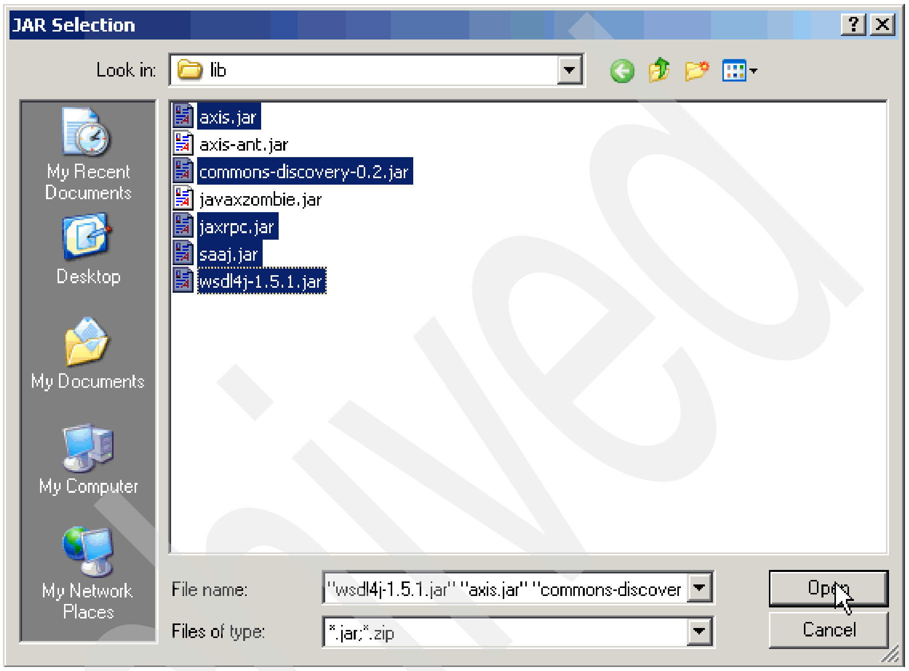
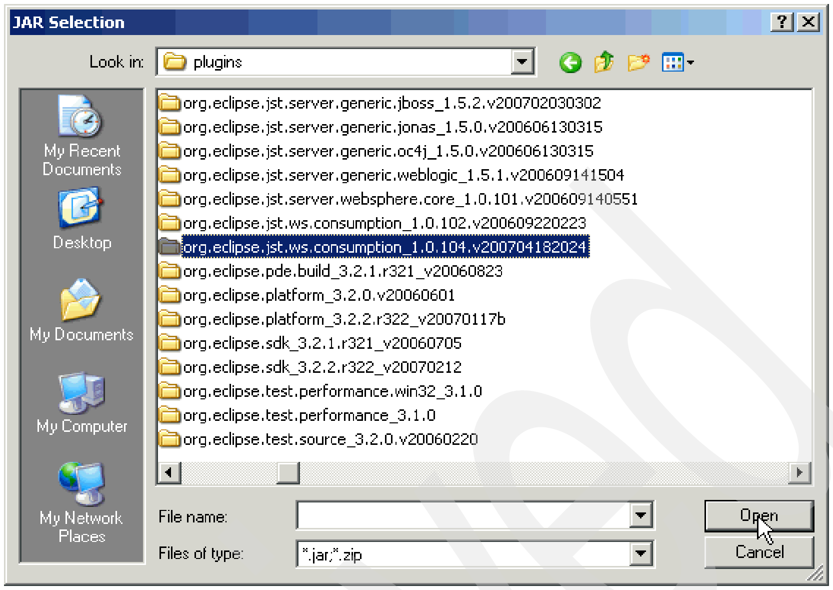


Figure 4-48 Selecting Apache AXIS libraries (2)

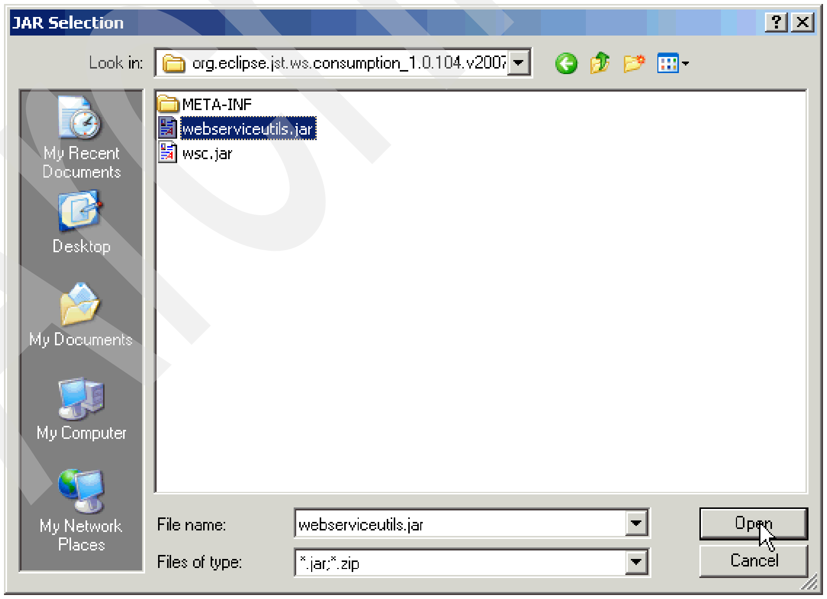
8. Now you need to add the two remaining jar files to the Java build path. In the Java settings pane, click **Add External JARs** again. This brings up the JAR selection window again in which you can select the JARs to add. In our case the webserviceutils.jar lives in the C:\Program Files\IBM\SDP70Shared\plugins directory. Select **org.eclipse.jst.ws.consumption_1.0.104.v200704182024**, as shown in Figure 4-49. Click **Open**.



```
plugins

org.eclipse.jst.server.generic.jboss_1.5.2.v200702030302
org.eclipse.jst.server.generic.jonas_1.5.0.v200606130315
org.eclipse.jst.server.generic.oc4j_1.5.0.v200606130315
org.eclipse.jst.server.generic.weblogic_1.5.1.v200609141504
org.eclipse.jst.server.websphere.core_1.0.101.v200609140551
org.eclipse.jst.ws.consumption_1.0.102.v200609220223
org.eclipse.jst.ws.consumption_1.0.104.v200704182024
org.eclipse.pde.build_3.2.1.r321_v20060823
org.eclipse.platform_3.2.0.v20060601
org.eclipse.platform_3.2.2.r322_v20070117b
org.eclipse.sdk_3.2.1.r321_v20060705
org.eclipse.sdk_3.2.2.r322_v20070212
org.eclipse.test.performance.win32_3.1.0
org.eclipse.test.performance_3.1.0
org.eclipse.test.source_3.2.0.v20060220

File name:
Files of type: *.jar;*.zip
```

Figure 4-49 Selecting webserviceutils.jar file

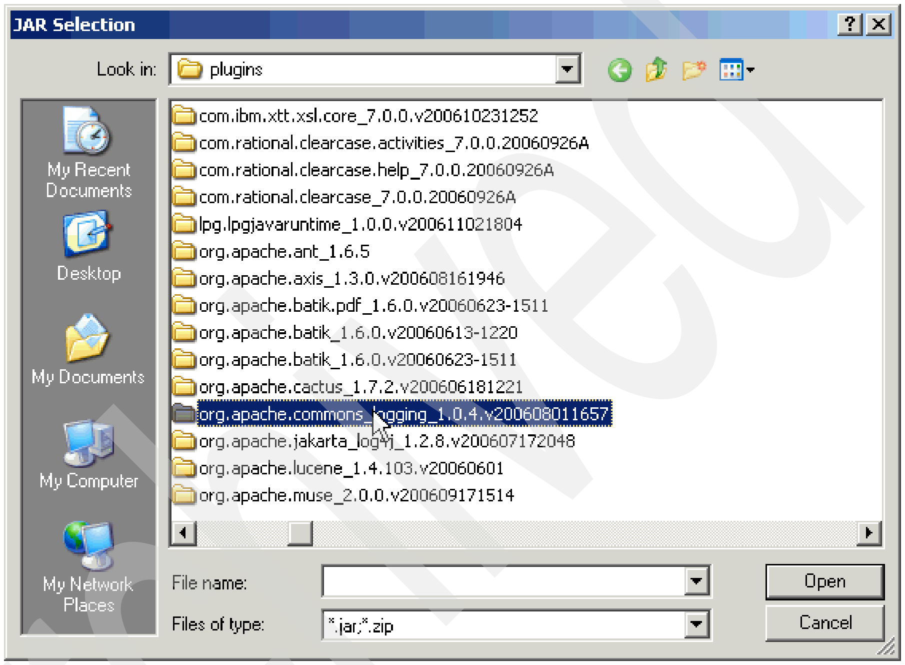
9. In the next pane select the file as highlighted in Figure 4-50. Click **Open**. Six of the seven required jar files have now been added to the Java build path.



```
org.eclipse.jst.ws.consumption_1.0.104.v2007

META-INF
webserviceutils.jar
wsc.jar

File name: webserviceutils.jar
Files of type: *.jar;*.zip
```

Figure 4-50 Selecting webserviceutils.jar file (2)

10. The last jar file to be added is commons-logging-1.0.4.jar. In the Java settings pane, click **Add External JARs** again. This brings up the JAR selection window again in which you can select the JARs to add. In our case, commons-logging-1.0.4.jar lives in the C:\Program Files\IBM\SDP70Shared\plugins directory. Select **org.apache.commons_logging_1.0.4.v200608011657**, as shown in Figure 4-51. Click **Open**.



com.ibm.xtt.xsl.core_7.0.0.v200610231252
com.rational.clearcase.activities_7.0.0.20060926A
com.rational.clearcase.help_7.0.0.20060926A
com.rational.clearcase_7.0.0.20060926A
lpg.lpgjavaruntime_1.0.0.v200611021804
org.apache.ant_1.6.5
org.apache.axis_1.3.0.v200608161946
org.apache.batik.pdf_1.6.0.v20060623-1511
org.apache.batik_1.6.0.v20060613-1220
org.apache.batik_1.6.0.v20060623-1511
org.apache.cactus_1.7.2.v200606181221
org.apache.commons_logging_1.0.4.v200608011657
org.apache.jakarta_log4j_1.2.8.v200607172048
org.apache.lucene_1.4.103.v20060601
org.apache.muse_2.0.0.v200609171514

Figure 4-51 Selecting the commons-logging.jar file

11. In the next pane, select **lib**.

12. In the next pane select the file as highlighted in Figure 4-52. Click **Open**. All required jar files have now been added to the Java build path.

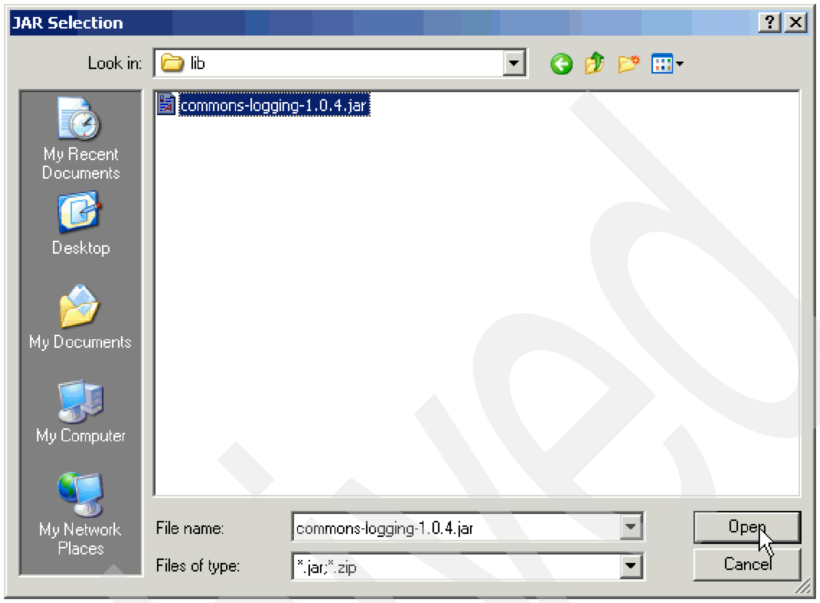
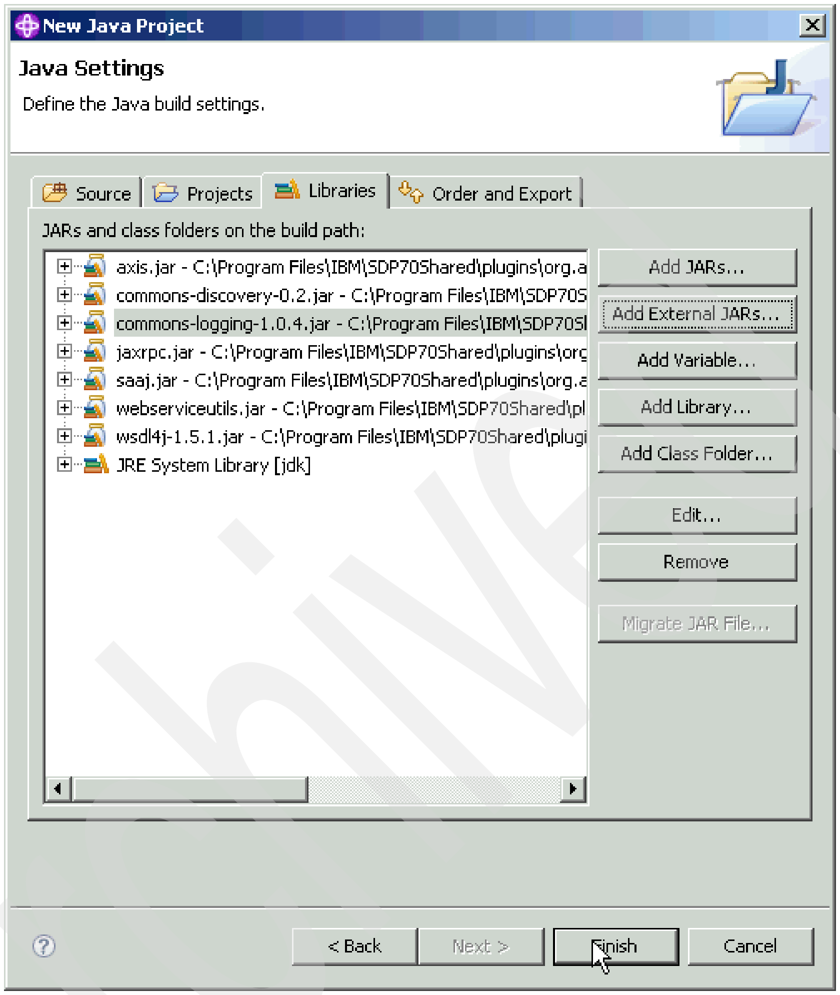


Figure 4-52 Selecting the commons-logging.jar file (2)

13. Back in the Java settings pane, you should now see the complete build path for the client program that we will create in the next steps. See Figure 4-53. Click **Finish**. If you see a message box asking you to open an associated perspective, click **Yes**.



Java Settings

Define the Java build settings.

JARs and class folders on the build path:

axis.jar - C:\Program Files\IBM\SDP70Shared\plugins\org.a
commons-discovery-0.2.jar - C:\Program Files\IBM\SDP70S
commons-logging-1.0.4.jar - C:\Program Files\IBM\SDP70Sl
jaxrpc.jar - C:\Program Files\IBM\SDP70Shared\plugins\org
saaj.jar - C:\Program Files\IBM\SDP70Shared\plugins\org.a
webserviceutils.jar - C:\Program Files\IBM\SDP70Shared\pl
wsdl4j-1.5.1.jar - C:\Program Files\IBM\SDP70Shared\plugi
JRE System Library [jdk]

Figure 4-53 Java settings with complete Java build path

We now import all required client classes that were generated previously on z/OS by the IMS SOAP Gateway deployment utility. You will use the Import function in RDz to directly import them from the z/OS USS file system into your new Java project.

14. Create a source folder in the PhoneBookClientXX project to hold the imported Java client classes (which are in source format). Right-click **PhoneBookClientXX** and select **New** → **Source Folder** from the pop-up window, as shown in Figure 4-54.

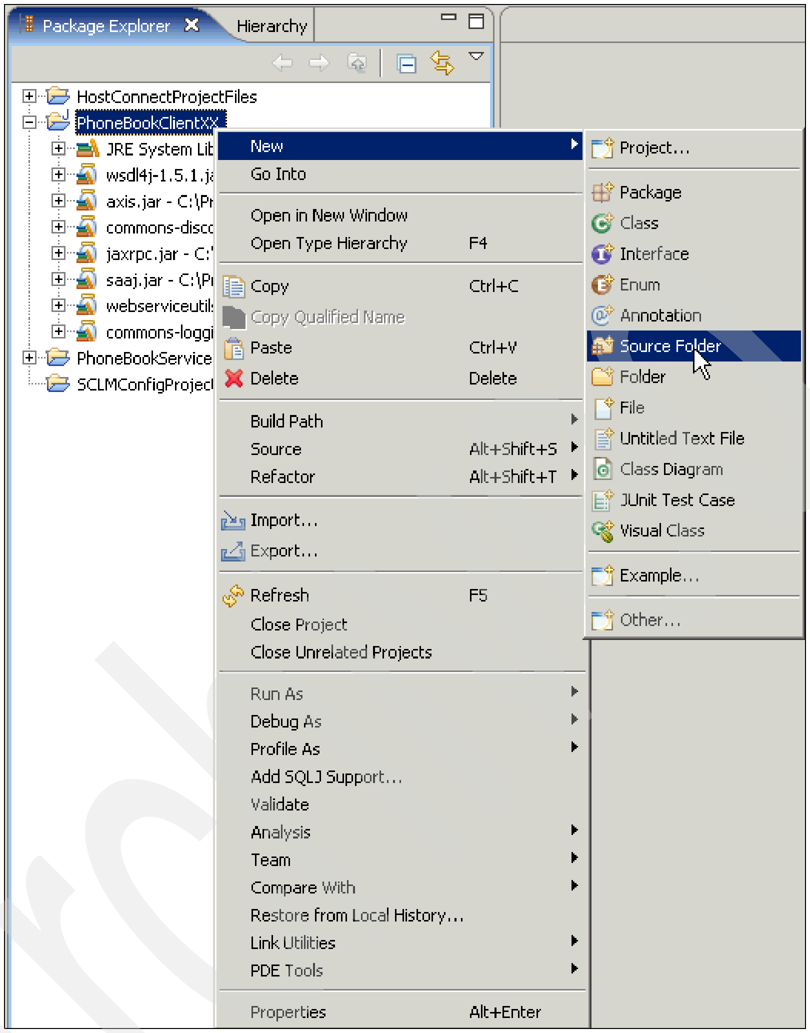
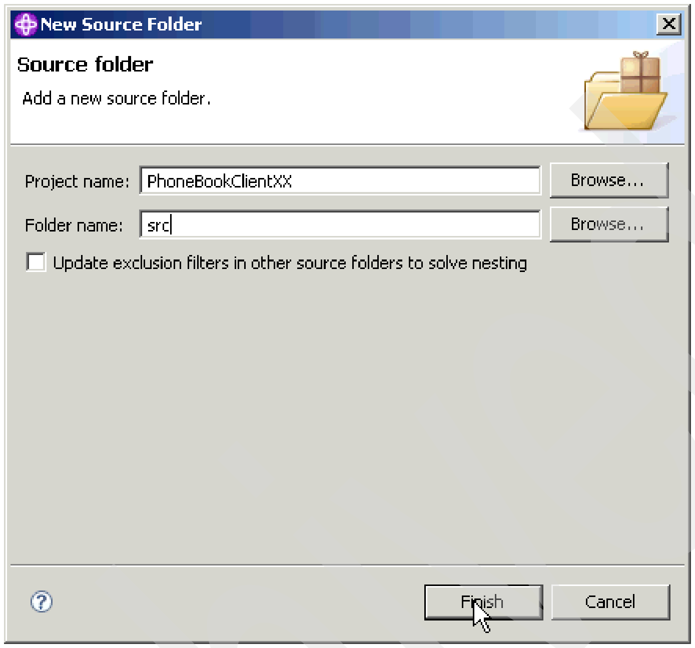


Figure 4-54 Creating a new source folder

15. In the next pane, only enter a name (**src**) in the Folder name: and click **Finish**. See Figure 4-55.



Project name: PhoneBookClientXX

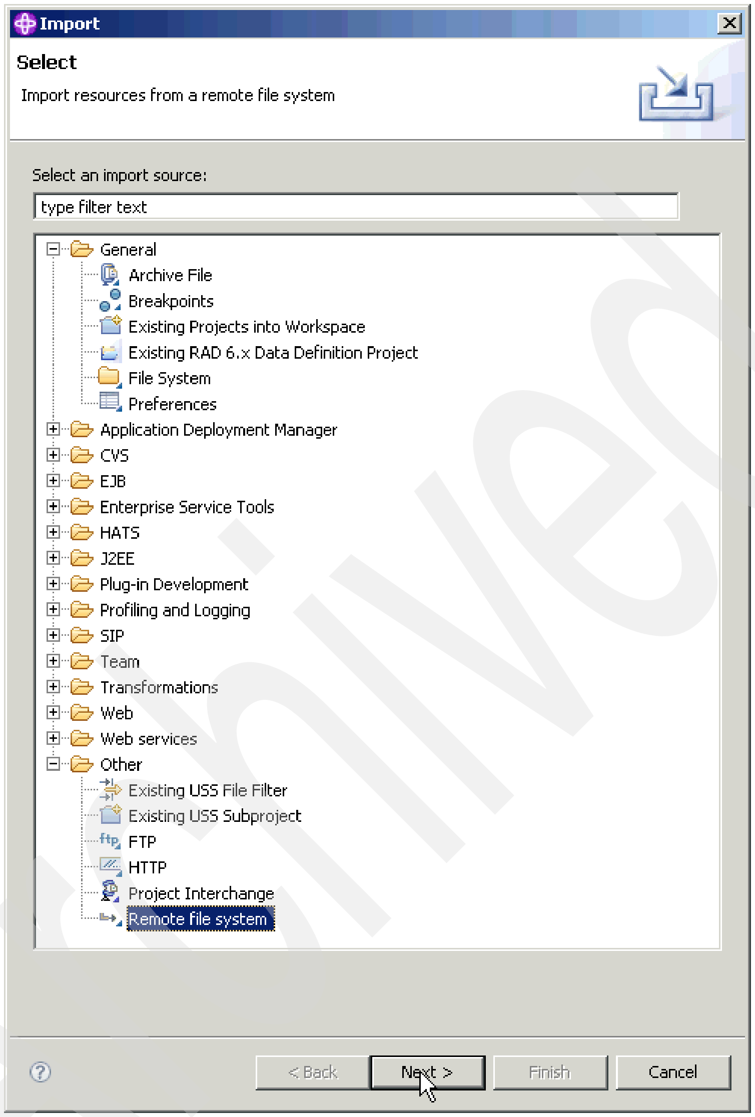
Folder name: src

☐ Update exclusion filters in other source folders to solve nesting

Figure 4-55 Entering a name for the new source folder

16. If not open yet, open the z/OS Projects view by selecting **Windows** → **Show View** → **Other**. Expand **Remote Systems** and select **Remote Systems** from the list in the pop-up window. This brings up the Remote Systems view again with the connection to the wtscz1.itso.ibm.com system.
17. In the Remote Systems views, right-click **wtscz1.itso.ibm.com** and select **Connect** from the pop-up window[3]. Use your user ID and password to log in again.
18. In the Package Explorer view, expand the **PhoneBookClientXX** project, highlight the **src** folder and right-click it. Select **Import** from the pop-up window. In the Select pane scroll down and expand **Other**. Select **Remote file system**, as shown in Figure 4-56.

[3] RDz remembers which was the last system you were logged into. So, usually RDz will automatically try to reconnect to the wtscz1.itso.ibm.com system after you open the z/OS Projects perspective and you will get the login window directly.


Import

Select

Import resources from a remote file system

Select an import source:

type filter text

- General
  - Archive File
  - Breakpoints
  - Existing Projects into Workspace
  - Existing RAD 6.x Data Definition Project
  - File System
  - Preferences
- Application Deployment Manager
- CVS
- EJB
- Enterprise Service Tools
- HATS
- J2EE
- Plug-in Development
- Profiling and Logging
- SIP
- Team
- Transformations
- Web
- Web services
- Other
  - Existing USS File Filter
  - Existing USS Subproject
  - FTP
  - HTTP
  - Project Interchange
  - Remote file system

Figure 4-56   Selecting Remote file system for import

19. In the next pane, the Remote file systems pane, click **Browse**. Expand **wtscz1.itso.ibm.com** → **/** → **u**. Highlight your home directory (**itsoxx**), into which you earlier generated your client classes (refer to step 15 on page 177). Click **OK**. See Figure 4-57.

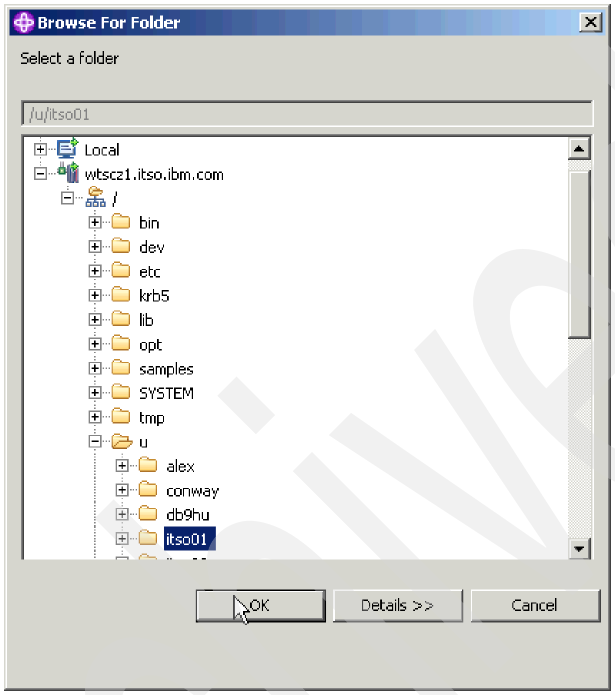
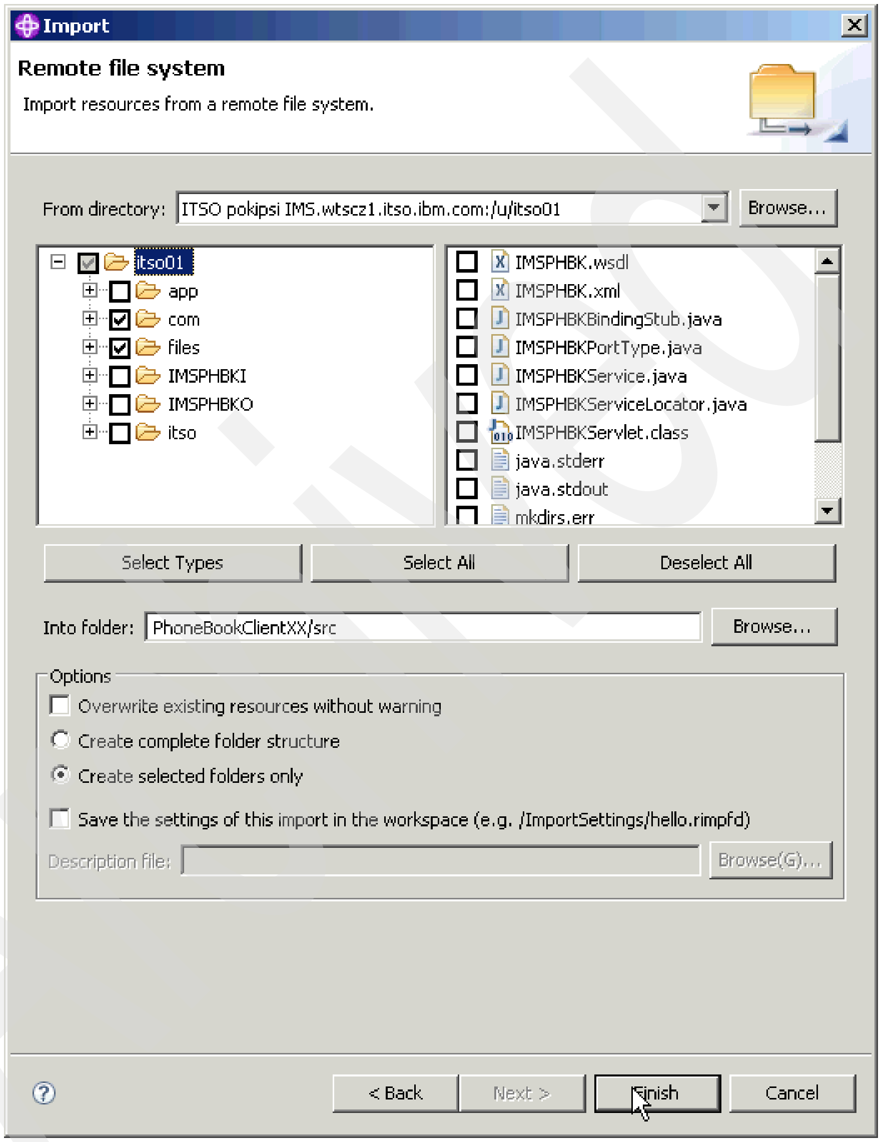


Figure 4-57 Selecting your home directory in the Remote file system

20. In the next pane, on the left, expand your home directory. In the directory tree that opens, check **files** and **com**. Leave the default for all other values in this pane. Click **Finish**. See Figure 4-58.


Import

Remote file system

Import resources from a remote file system.

From directory: ITSO pokipsi IMS.wtscz1.itso.ibm.com:/u/itso01

- itso01
  - [ ] app
  - [✓] com
  - [✓] files
  - [ ] IMSPHBKI
  - [ ] IMSPHBKO
  - [ ] itso

- [ ] IMSPHBK.wsdl
- [ ] IMSPHBK.xml
- [ ] IMSPHBKBindingStub.java
- [ ] IMSPHBKPortType.java
- [ ] IMSPHBKService.java
- [ ] IMSPHBKServiceLocator.java
- [ ] IMSPHBKServlet.class
- [ ] java.stderr
- [ ] java.stdout
- [ ] mkdirs.err

Into folder: PhoneBookClientXX/src

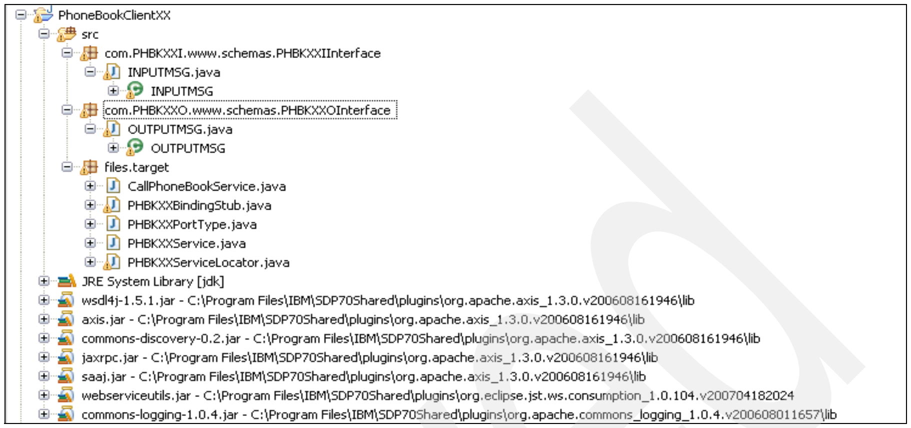
Options

- [ ] Overwrite existing resources without warning
- ( ) Create complete folder structure
- (●) Create selected folders only
- [ ] Save the settings of this import in the workspace (e.g. /ImportSettings/hello.rimpfd)

Description file:

Figure 4-58 Selecting packages to import

21. You now have three new packages in the src folder of your PhoneBookClientXX project, as shown in Figure 4-59.



```
PhoneBookClientXX
  src
    com.PHBKXXI.www.schemas.PHBKXXIInterface
      INPUTMSG.java
        INPUTMSG
    com.PHBKXXO.www.schemas.PHBKXXOInterface
      OUTPUTMSG.java
        OUTPUTMSG
    files.target
      CallPhoneBookService.java
      PHBKXXBindingStub.java
      PHBKXXPortType.java
      PHBKXXService.java
      PHBKXXServiceLocator.java
  JRE System Library [jdk]
  wsdl4j-1.5.1.jar - C:\Program Files\IBM\SDP70Shared\plugins\org.apache.axis_1.3.0.v200608161946\lib
  axis.jar - C:\Program Files\IBM\SDP70Shared\plugins\org.apache.axis_1.3.0.v200608161946\lib
  commons-discovery-0.2.jar - C:\Program Files\IBM\SDP70Shared\plugins\org.apache.axis_1.3.0.v200608161946\lib
  jaxrpc.jar - C:\Program Files\IBM\SDP70Shared\plugins\org.apache.axis_1.3.0.v200608161946\lib
  saaj.jar - C:\Program Files\IBM\SDP70Shared\plugins\org.apache.axis_1.3.0.v200608161946\lib
  webserviceutils.jar - C:\Program Files\IBM\SDP70Shared\plugins\org.eclipse.jst.ws.consumption_1.0.104.v200704182024
  commons-logging-1.0.4.jar - C:\Program Files\IBM\SDP70Shared\plugins\org.apache.commons_logging_1.0.4.v200608011657\lib
```

Figure 4-59 PhoneBookClientXX project after import

22. Expand the **PhoneBookClientXX** → **src** folder and highlight the **files.target** package. Right-click it and select **New** → **Class**, as shown in Figure 4-60.

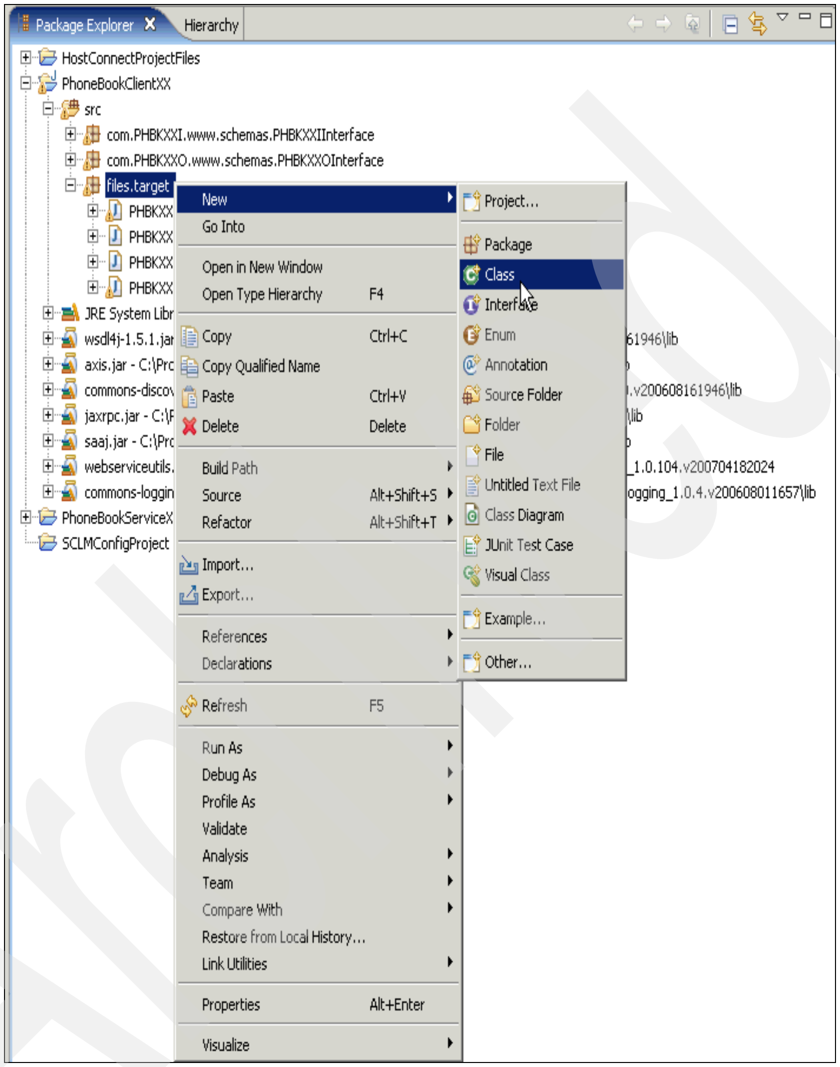


Figure 4-60 Creating a new Java class

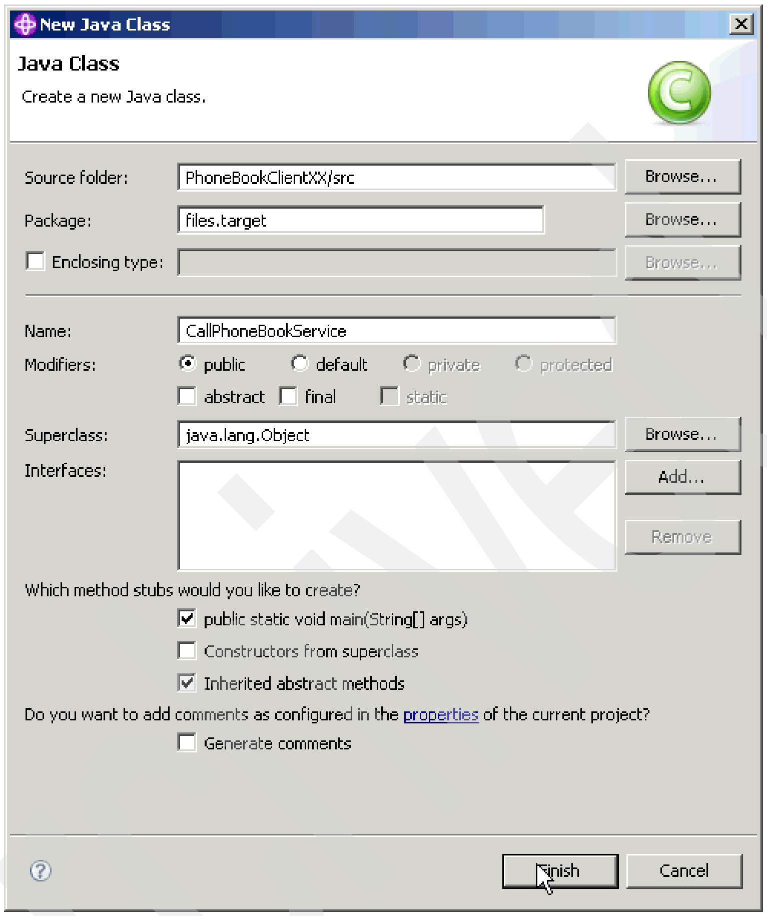
23. In the New Java Class window, specify:

– `CallPhoneBookService` in Name:

– Check **Public static void main**.

– Leave all other values default.

as shown in Figure 4-61. Click **Finish**.


New Java Class

Java Class

Create a new Java class.

Source folder: PhoneBookClientXX/src

Package: files.target

[ ] Enclosing type:

Name: CallPhoneBookService

Modifiers: [✓ public] default private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[✓] public static void main(String[] args)

[ ] Constructors from superclass

[✓] Inherited abstract methods

Do you want to add comments as configured in the properties of the current project?

[ ] Generate comments

Figure 4-61 New Java Class settings

24. An editor window will now open with an empty class. Add code to this class similar to what is shown in Example 4-2.

Example 4-2 CallPhoneBookService Java main program

```
package files.target;

import java.rmi.RemoteException;

import javax.xml.rpc.ServiceException;

import com.PHBKXXI.www.schemas.PHBKXXIInterface.INPUTMSG;
import com.PHBKXXO.www.schemas.PHBKXXOInterface.OUTPUTMSG;
```

```
public class CallPhoneBookService {

   /**
    * @param args
    */
   public static void main(String[] args) {
      // TODO Auto-generated method stub
      try {
         PHBKXXBindingStub stub = (PHBKXXBindingStub) new
PHBKXXServiceLocator()
               .getPHBKXXPort();
         INPUTMSG input = new INPUTMSG();

         try {
            input.setIn_trcd("IVTNO");
            input.setIn_cmd("DISPLAY");
            input.setIn_extn("");
            input.setIn_name1("LAST1");
            input.setIn_name2("");
            input.setIn_zip("");

            OUTPUTMSG out = stub.PHBKXXOperation(input);
            System.out.println("Message:" + out.getOut_msg());
            System.out.println("Name1:" + out.getOut_name1());
            System.out.println("Name2:" + out.getOut_name2());
            System.out.println("ZIP:" + out.getOut_zip());
            System.out.println("Extension:" + out.getOut_extn());

         } catch (RemoteException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
         }

      } catch (ServiceException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }

   }

}
```

25. Now save the Java class and you are ready to run the program. Right-click the **CallPhoneBookService** class and select **Run As** → **2 Java Application** from the pop-up window, as shown in Figure 4-62.

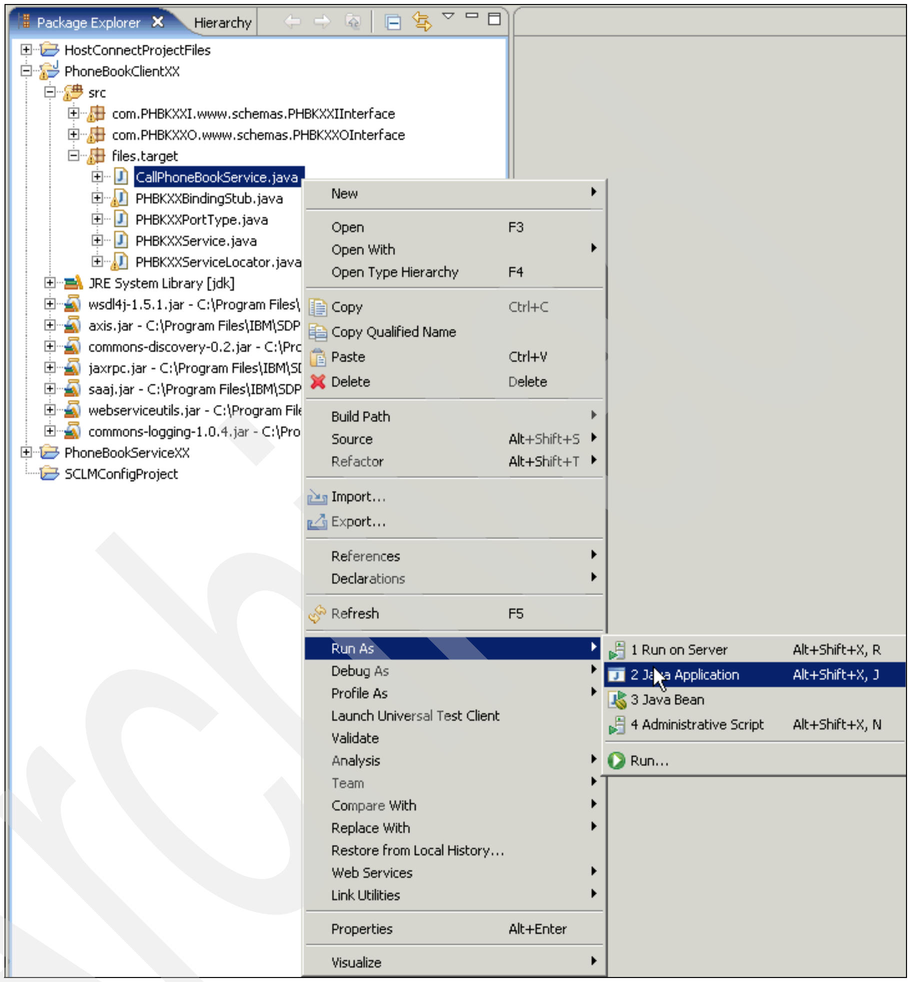

Package Explorer
Hierarchy
HostConnectProjectFiles
PhoneBookClientXX
src
com.PHBKXXI.www.schemas.PHBKXXIInterface
com.PHBKXXO.www.schemas.PHBKXXOInterface
files.target
CallPhoneBookService.java
PHBKXXBindingStub.java
PHBKXXPortType.java
PHBKXXService.java
PHBKXXServiceLocator.java
JRE System Library [jdk]
wsdl4j-1.5.1.jar - C:\Program Files\
axis.jar - C:\Program Files\IBM\SDP
commons-discovery-0.2.jar - C:\Pro
jaxrpc.jar - C:\Program Files\IBM\SI
saaj.jar - C:\Program Files\IBM\SDP
webserviceutils.jar - C:\Program File
commons-logging-1.0.4.jar - C:\Pro
PhoneBookServiceXX
SCLMConfigProject
New
Open F3
Open With
Open Type Hierarchy F4
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
References
Declarations
Refresh F5
Run As
Debug As
Profile As
Launch Universal Test Client
Validate
Analysis
Team
Compare With
Replace With
Restore from Local History...
Web Services
Link Utilities
Properties Alt+Enter
Visualize
1 Run on Server Alt+Shift+X, R
2 Java Application Alt+Shift+X, J
3 Java Bean
4 Administrative Script Alt+Shift+X, N
Run...

Figure 4-62 Running the Java main client program

26. The console window in RDz will now open to show you any errors and/or the output, as shown in Example 4-3.

Example 4-3 Output from the main program

```
Sep 26, 2007 3:48:46 PM org.apache.axis.utils.JavaUtils
isAttachmentSupported
WARNING: Unable to find required classes (javax.activation.DataHandler
and javax.mail.internet.MimeMultipart). Attachment support is disabled.
Message:ENTRY WAS DISPLAYED
Name1:LAST1
Name2:FIRST1
ZIP:D01/R01
Extension:8-111-1111
```

4.7 Additional material

Table 4-2 describes the additional material supplied with this book. Refer to Appendix B, “Additional material” on page 397 for download instructions.

Table 4-2 Supplied material

Name	Purpose
PhoneBookPI.zip	The PIF file consisting of the PhoneBookService z/OS local project and PhoneBookServiceClient java project
IMSPHBK.cbl	The PhoneBook input/output cobol structure
COMPPHBK.jcl	JCL to compile the generated XML adapter

5

Accessing a Web Service in WebSphere Application Server from CICS

This chapter describes a scenario in which CICS is the Web Service requester and WebSphere is Web Service provider. We go through the tasks required to deploy a CICS application as a Web Service requester and make an outbound call to WebSphere. The chapter is organized into the following sections:

- Planning a Web Service requester application
- CICS resource definitions for the Web Service requester
- Creating the requester application from a WSDL
- Testing
- Supplied material

5.1 Planning a Web Service requester application in CICS

When CICS is the service requester, it performs the following operations:

1. Builds a request using data provided by the application program.
2. Sends the request to the service provider.
3. Receives a response from the service provider.
4. Examines the response, and extract the contents that are relevant to the original application program.
5. Returns control to the application program.

For this example we used WebSphere Developer for System z Version 7 (WDz) to generate the mappings between the COBOL data structures and the Web Service provider's SOAP messages, but you could also use the newer version of this tool: Rational Developer for System z Version 7.1. This tool provides a rapid deployment of many applications into a Web Services setting with little or no additional programming. And when additional programming is required, it is usually straightforward, and can be done without changing existing business logic. The capabilities of this tool are described in detail in Chapter 2, "Accessing CICS from WebSphere using Web Services over HTTP" on page 3.

Figure 5-1 shows a typical application partitioned to ensure a separation between communications logic and business logic or presentation logic. Following this practice will help you to deploy new and existing applications as a Web Service requester in a straightforward way. In our example, the WDz tooling will generate the COBOL program named CUSTOMER that corresponds to the Communications logic piece.

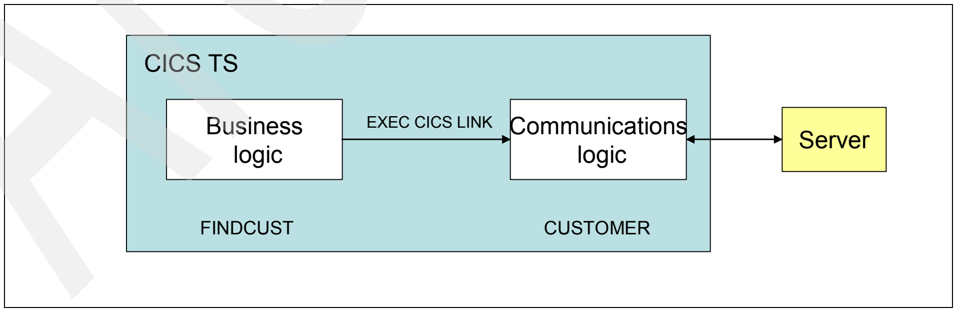


Figure 5-1 CICS application partitioned into communications and business logic

As a client requester, there is no business logic, but there is presentation logic (FINDCUST) to show the output values from the Web Service on the screen. The FINDCUST program sends a 3270 BMS map used to request a customer number. If the customer number is valid, FINDCUST then invokes another COBOL program named CUSTOMER using an EXEC CICS LINK via the CICS COMMAREA. The CUSTOMER program receives the customer number, populates the Web service request with that number, invokes the Web Service using the EXEC CICS INVOKE WEBSERVICE command and sends the Web Service response data results retrieved back to the COMMAREA. When FINDCUST receives control, it displays the data on the 3270 screen.

5.1.1 Error Handling

If your service requester application receives a SOAP fault message from the service provider, you need to decide how your application program should handle the fault message. CICS does not automatically roll back any changes when a SOAP fault message is received.

5.2 CICS resource definitions for Web Services requester

The following CICS resources support a Web services requester in CICS.

5.2.1 PIPELINE resource

A PIPELINE resource definition provides information about the message handler programs that act on a service request and on the response. Typically, a single PIPELINE definition defines an infrastructure that can be used by many applications. The information about the message handlers is supplied indirectly.

The PIPELINE specifies the name of an HFS file that contains an XML description of the handlers and their configurations. PIPELINE resources are defined for Web Service requesters and Web Service providers. A PIPELINE resource that is created for a Web Service requester cannot be used for a Web Service provider, and vice versa.

The two types of PIPELINE are distinguished by the contents of the pipeline configuration file that is specified in the CONFIGFILE attribute: for a service provider, the top level element is <provider_pipeline>; for a service requester it is <requester_pipeline>.

Example 5-1 Example of a configuration file for a service requester pipeline

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<requester_pipeline
xmlns="http://www.ibm.com/software/htp/cics/pipeline"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipelinereques
ter.xsd ">
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler/>
    </service_handler_list>
  </service>
</requester_pipeline>
```

The `<cics_soap_1.1_handler>` indicates that the terminal handler of the pipeline is the CICS-supplied handler program for SOAP 1.1 messages. For a detailed explanation of the pipeline definition and the configuration file structure, refer to *Implementing CICS Web Services*, SG24-7206 at:

http://www.redbooks.ibm.com/redbooks/pdfs/sg247206.pdf

5.2.2 WEBSERVICE resource

A WEBSERVICE resource definition defines aspects of the runtime environment for a CICS application program deployed in a Web Services setting. Although CICS provides the usual resource definition mechanisms for WEBSERVICE resources, they are typically created automatically from a Web Service binding file when the PIPELINE's pickup directory is scanned. This happens when the PIPELINE resource is installed, or as a result of a PERFORM PIPELINE SCAN command.

The attributes applied to the WEBSERVICE resource in this case come from a Web Services binding file, which is created by the WDz tooling. The information in the binding file comes from the Web Service description. A WEBSERVICE resource that is created for a Web Service requester cannot be used for a Web Service provider, and vice versa. The two sorts of WEBSERVICE are distinguished by the PROGRAM attribute: for a service provider, the attribute must be specified; for a service requester it must be omitted.

5.2.3 Web Service binding file

A Web Services description contains abstract representations of the input and output messages used by the service. When a Web Service provider or Web

Service requester application executes, CICS needs information about how the content of the messages maps to the data structures used by the application. This information is held in a Web Service binding file.

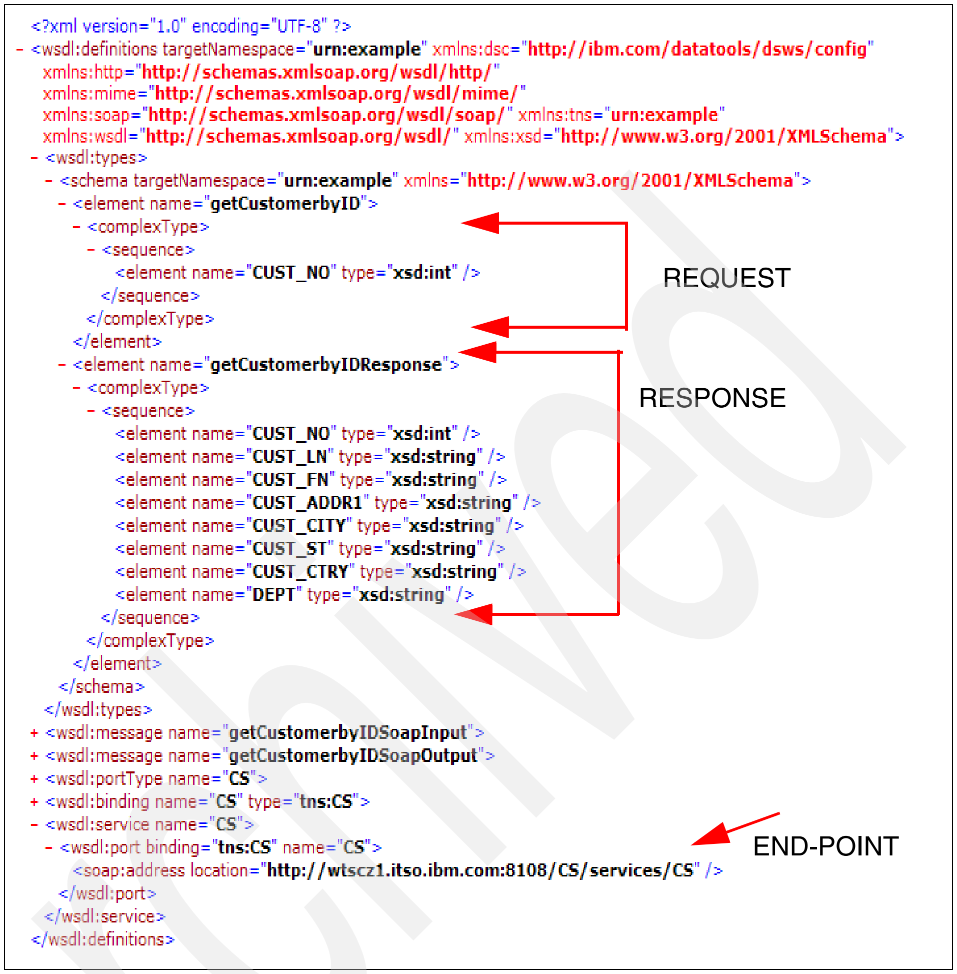
At runtime, CICS uses information in the Web Service binding file to perform the mapping between application data structures and SOAP messages.

5.3 Creating a Web Service requester application from a WSDL

We make use of an existing Web Service already deployed to the WebSphere z/OS environment. In our example, the WSDL endpoint is located at:

```
http://wtscz1.itso.ibm.com:8108/CS/services/CS/wsdl/CS.wsdl
```

Figure 5-2 on page 210 shows the entire WSDL with the request, response and end-point marked. The request takes in a value for customer ID and the response will have all the details of the customer that correspond to the input request customer ID. The next step is to generate the driver template, input and output convertors, and the CICS required resource file (*.WSBind).



```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="urn:example" xmlns:dsc="http://ibm.com/datatools/dsws/config"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="urn:example"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <wsdl:types>
    - <schema targetNamespace="urn:example" xmlns="http://www.w3.org/2001/XMLSchema">
      - <element name="getCustomerbyID">
        - <complexType>
          - <sequence>
              <element name="CUST_NO" type="xsd:int" />
            </sequence>
          </complexType>
        </element>
      - <element name="getCustomerbyIDResponse">
        - <complexType>
          - <sequence>
              <element name="CUST_NO" type="xsd:int" />
              <element name="CUST_LN" type="xsd:string" />
              <element name="CUST_FN" type="xsd:string" />
              <element name="CUST_ADDR1" type="xsd:string" />
              <element name="CUST_CITY" type="xsd:string" />
              <element name="CUST_ST" type="xsd:string" />
              <element name="CUST_CTRY" type="xsd:string" />
              <element name="DEPT" type="xsd:string" />
            </sequence>
          </complexType>
        </element>
      </schema>
    </wsdl:types>
  + <wsdl:message name="getCustomerbyIDSoapInput">
  + <wsdl:message name="getCustomerbyIDSoapOutput">
  + <wsdl:portType name="CS">
  + <wsdl:binding name="CS" type="tns:CS">
  - <wsdl:service name="CS">
    - <wsdl:port binding="tns:CS" name="CS">
        <soap:address location="http://wtscz1.itso.ibm.com:8108/CS/services/CS" />
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

Figure 5-2 The WSDL from the Web Service provider

The tasks that we perform for this example are:

1. Creating a project in WDz to hold all the artefacts ("Creating a WDz Web Services for CICS project" on page 211)
2. Importing the WSDL into a WDz Web Services for CICS project ("Importing the WSDL of Web Service to be called" on page 212)
3. Generating the Web Service artifacts and code ("Generating the Web Service artefacts" on page 214)
4. Creating the CICS requester PIPELINE ("Creating the CICS requester pipeline" on page 220)

5. Creating the CICS requester Web Service ("Creating the CICS requester Web Service" on page 222)
6. Modifying the driver and deploying the generated COBOL program to the CICS transaction server on z/OS ("Modifying the driver and installing the COBOL programs into CICS" on page 224)
7. Testing ("Testing" on page 232)

5.3.1 Creating a WDz Web Services for CICS project

The first thing to do is to create a project in WDz to hold all the artefacts we will be importing, creating and generating:

1. Open the EST perspective by selecting **Window** → **Open perspective** → **Other** and selecting the **Enterprise Service Tools** in the popup window.
2. Select **File** → **New** → **Project.** Select **CICS Web Services Project** in the Enterprise Service Tools section as shown in Figure 5-3. Click **Next**.
3. In the next window, supply a project name. We use Customer for our example. Click **Next**. See Figure 5-4 on page 212.

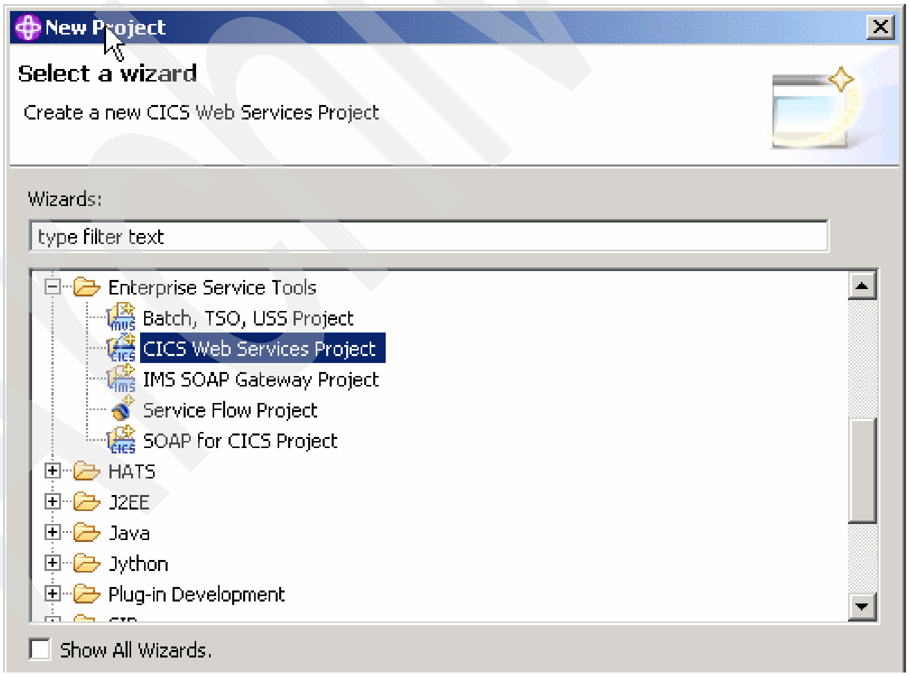
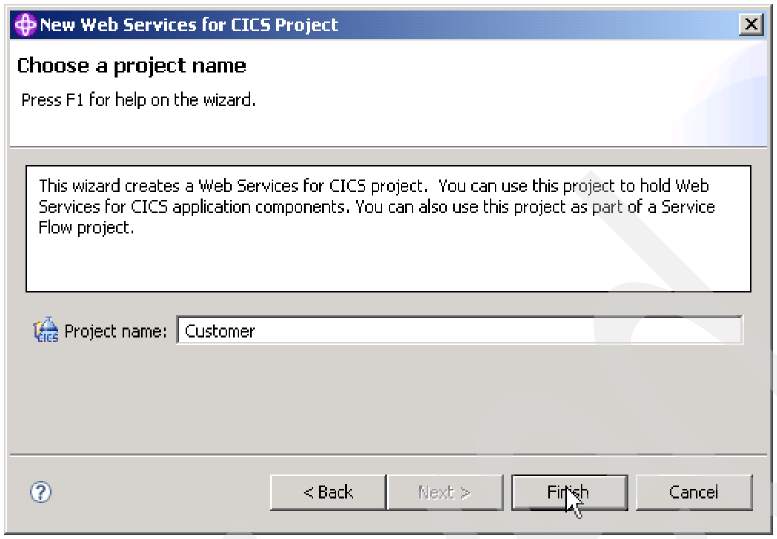


Figure 5-3 New Project wizard


New Web Services for CICS Project
Choose a project name
Press F1 for help on the wizard.

This wizard creates a Web Services for CICS project. You can use this project to hold Web Services for CICS application components. You can also use this project as part of a Service Flow project.

Project name: Customer

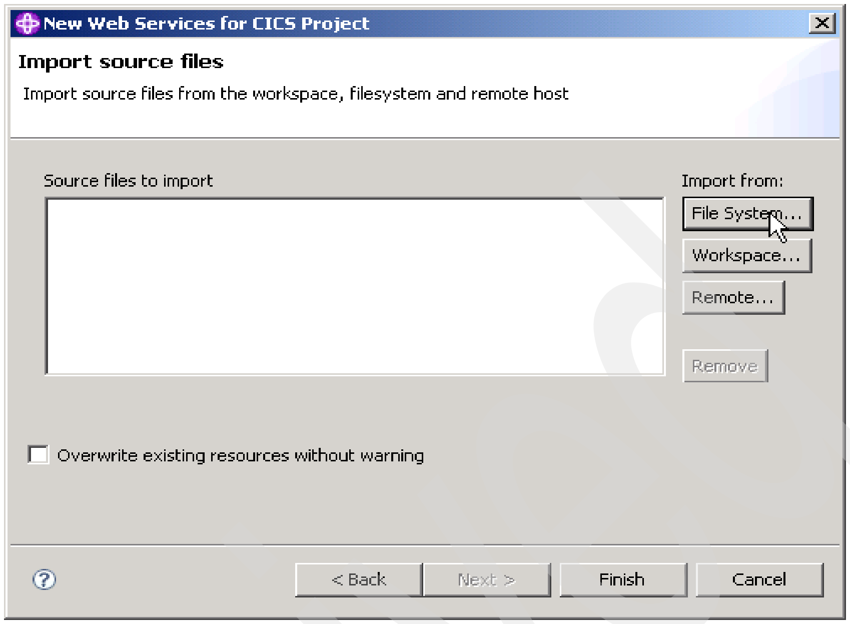
< Back
Next >
Finish
Cancel

Figure 5-4 Project name for the "New Web Services for CICS Project" wizard

5.3.2 Importing the WSDL of Web Service to be called

In this section, we import the WSDL.

1. To import the WSDL, click **File System** and navigate to the location where the WSDL file is saved. You must change the filter in the Browse window to *.wsdl in order to locate the WSDL. Click **Finish**. See Figure 5-5 on page 213 and Figure 5-6 on page 213.


New Web Services for CICS Project

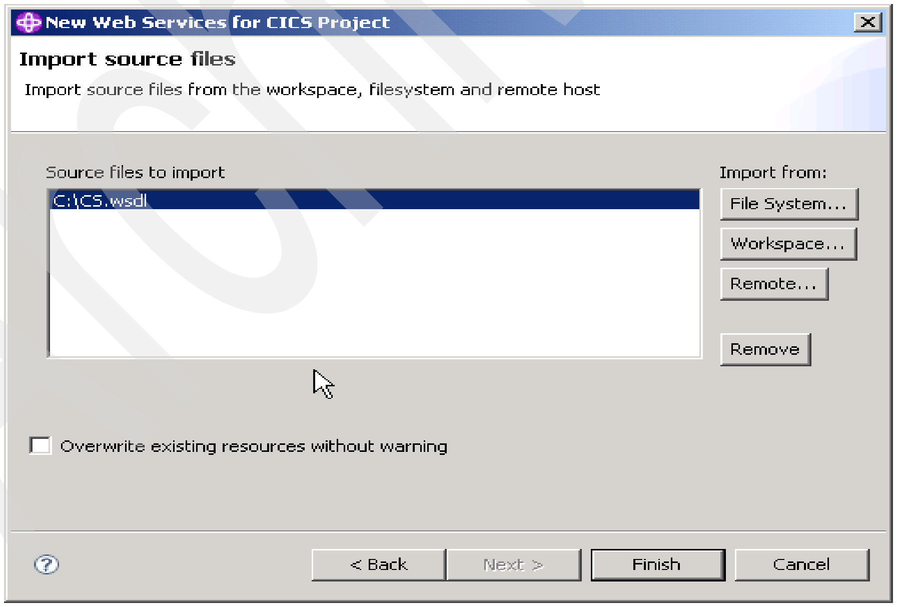
Import source files

Import source files from the workspace, filesystem and remote host

Source files to import

Import from:

Figure 5-5 Importing the WSDL - Step 1


New Web Services for CICS Project

Import source files

Import source files from the workspace, filesystem and remote host

Source files to import

C:\CS.wsdl

Import from:

Figure 5-6 Importing the WSDL - Step 2

5.3.3 Generating the Web Service artefacts

After having created a project and imported the WSDL, it is time to go through the generation steps:

1. Right-click the WSDL file and choose **Generate Web Services for CICS resources** from the pop-up menu, as shown in Figure 5-7.

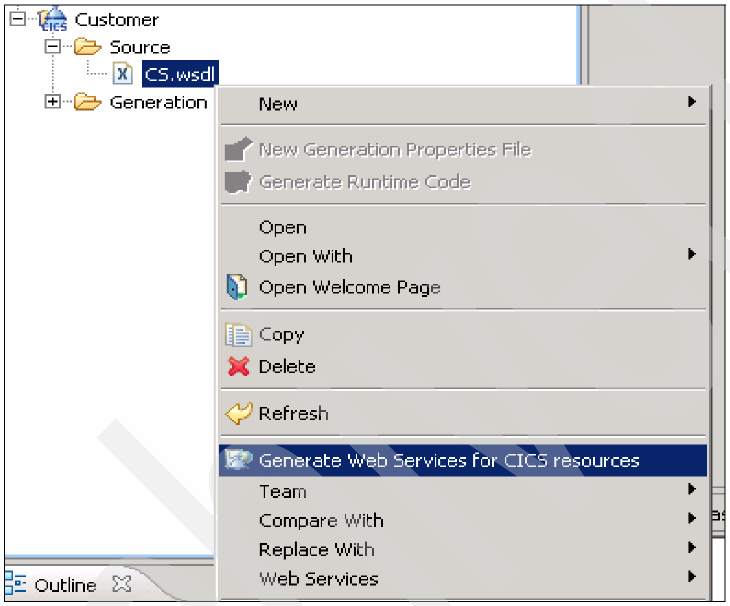
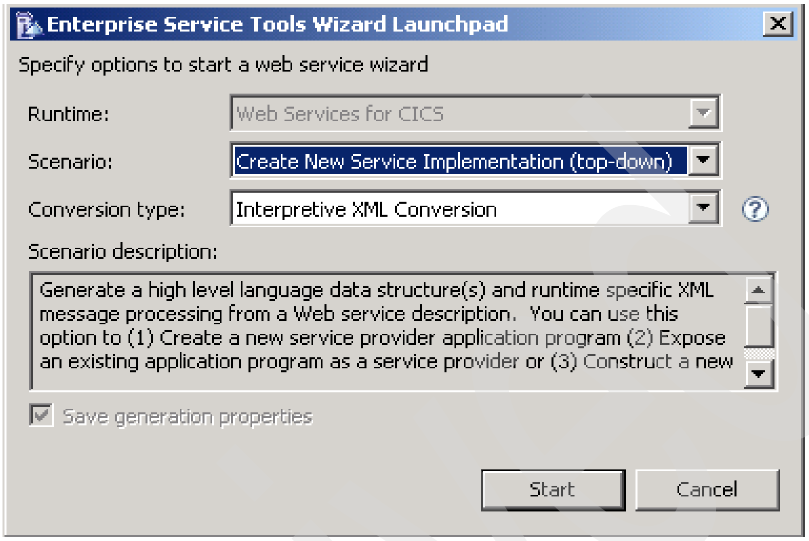


Figure 5-7 Invoking the "Generate Web Services for CICS resources" menu

2. In the Enterprise Service Tools (EST) launchpad window shown in Figure 5-8, take the defaults and click **Start**.


Enterprise Service Tools Wizard Launchpad

Specify options to start a web service wizard

Runtime: Web Services for CICS

Scenario: Create New Service Implementation (top-down)

Conversion type: Interpretive XML Conversion

Scenario description:

Generate a high level language data structure(s) and runtime specific XML message processing from a Web service description. You can use this option to (1) Create a new service provider application program (2) Expose an existing application program as a service provider or (3) Construct a new

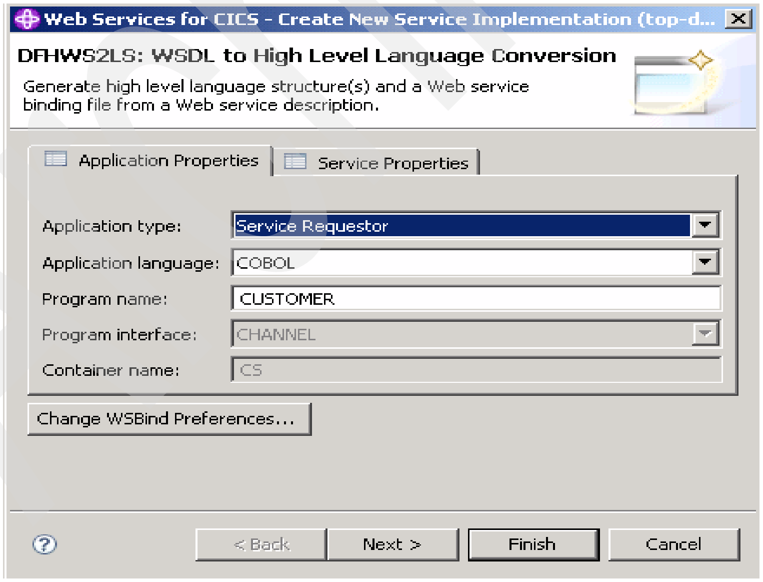
☑ Save generation properties

Figure 5-8 EST launchpad

3. In the “Web Services for CICS - Create New Service Implementation (Top-down)” window, change the Application Type to Service Requestor and the Program name to Customer. The Service Properties tab has the bindings that have been defined in the WSDL. In our example, we have only one binding and we take the defaults as is. See Figure 5-9.

Note: Runtime XML conversion: compiled or interpretive. There are two types of runtime XML conversion that are provided by Web services for CICS: interpretive XML conversion (less flexible) and compiled XML conversion (recommended). When you select either of these two types, the selection applies to both inbound and outbound XML conversion for the resulting Web Service provider or requester.

This includes: for inbound XML conversion (conversion of data from XML format to specific COBOL data structure), the Web Service requester converts the data in an inbound Web Service response from XML format to a high-level language data format. For outbound XML conversion, the Web service requester converts the data in an outbound Web Service request from a specific COBOL data structure format to XML format. For this example we choose “Interpretive XML conversion”. Using the interpretive XML converter has the advantage of freeing CICS Web Service developers from the task of writing their own XML conversion programs.

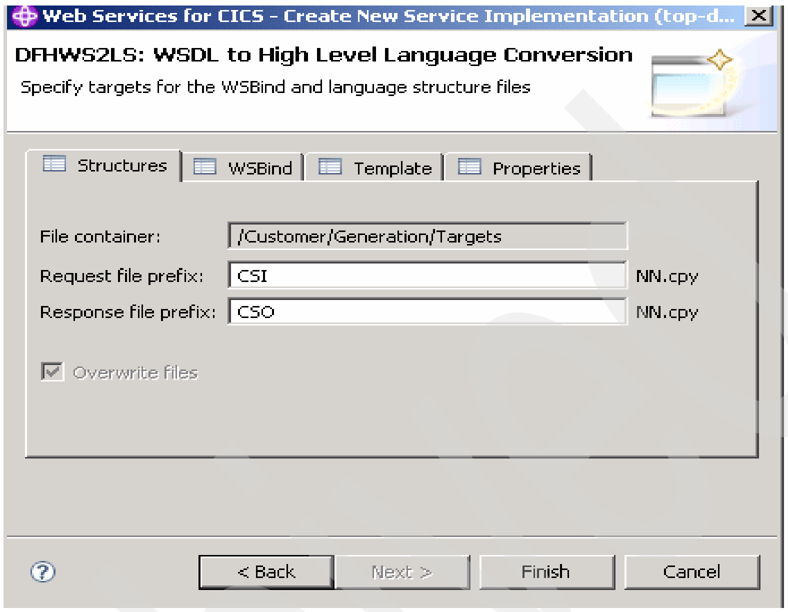

Web Services for CICS - Create New Service Implementation (top-d...
DFHWS2LS: WSDL to High Level Language Conversion
Generate high level language structure(s) and a Web service binding file from a Web service description.
Application Properties
Service Properties

Application type: Service Requestor
Application language: COBOL
Program name: CUSTOMER
Program interface: CHANNEL
Container name: CS

Change WSBind Preferences...
< Back
Next >
Finish
Cancel

Figure 5-9 Choosing the Application type and entering the Program name

4. The next window allows us to specify the Request and Response COBOL file structure prefixes. Take the defaults as shown in Figure 5-10.


Web Services for CICS - Create New Service Implementation (top-d...

DFHWS2LS: WSDL to High Level Language Conversion

Specify targets for the WSBind and language structure files

File container: /Customer/Generation/Targets

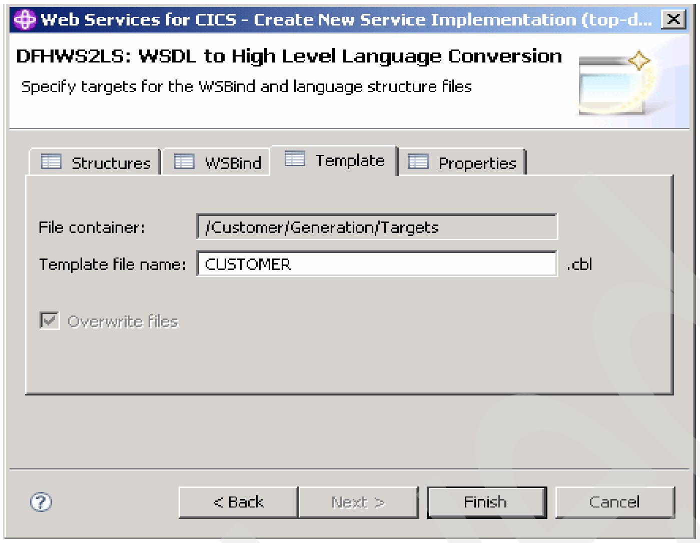
Request file prefix: CSI NN.cpy

Response file prefix: CSO NN.cpy

[✓] Overwrite files

Figure 5-10 Accepting defaults for the Input and Output COBOL structures

5. Before you click **Finish**, switch to the template tab in the same window. This section allows you to specify the name of the COBOL template file that will be generated. This file will represent the skeleton of the COBOL program that will make the Web Service call. For this example, we change this to CUSTOMER as shown in Figure 5-11 on page 218 and then click **Finish**.


Web Services for CICS - Create New Service Implementation (top-d...

DFHWS2LS: WSDL to High Level Language Conversion

Specify targets for the WSBind and language structure files

File container: /Customer/Generation/Targets

Template file name: CUSTOMER .cbl

☑ Overwrite files

Figure 5-11 Specifying the Template file name

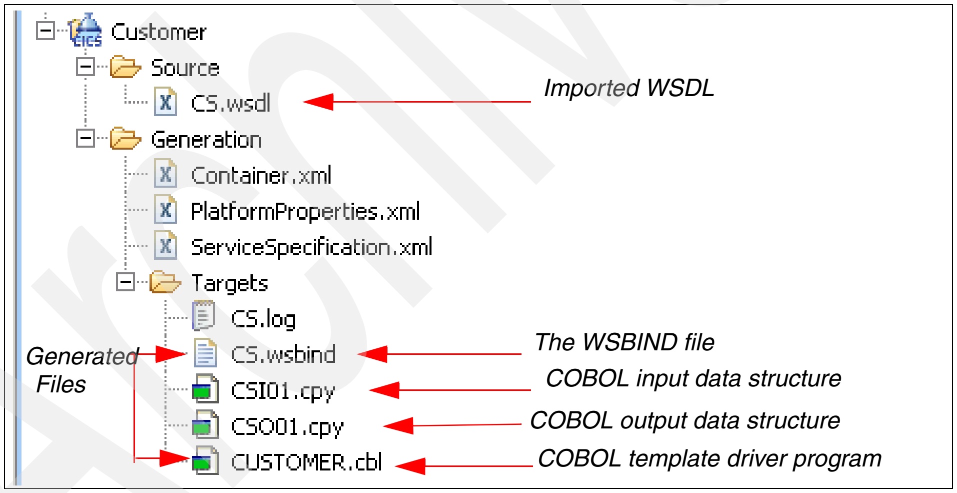


Figure 5-12 Generated artifacts

Figure 5-12 shows all the artifacts generated in the above steps:

- The WSDL file describes the Web Service to the Web Service clients. The WSDL file is the source document for mapping an XML schema representation to the COBOL data model.
- The WSBind file is a resource that describes the specifics of the Web Service to CICS. For example, it contains information about what the system should

do to convert an input XML document to a COBOL language structure and what to do to convert the output COBOL data to the output XML document.

- CSI01.cpy represents the input COBOL copybook. This contains the generated request language structure(s) for the WSDL operation getCustomerbyID. The source is shown in Example 5-2.

Example 5-2 CSI01.cpy source

```
05 getCustomerbyID.
10 CUSTXNO                       PIC S9(9) COMP-5 SYNC.
```

- CSO01.cpy represents the output COBOL copybook. It contains the generated response language structure. The source is shown in Example 5-3.

> **Note:** It is important to examine the generated Input/Output COBOL structures and make sure the data elements have a one-on-one mapping with the XML structure. If the Web Service provider were to expose a more complex, deeply nested XML structure for the request or response, the tooling may not be able to create appropriate COBOL structures. In general, it is not best practice for the Web Service provider to have a complex, deeply nested structure because of performance and interoperability concerns.

Example 5-3 CSO01.cpy source

```
05 getCustomerbyIDResponse.
             10 CUSTXNO                         PIC S9(9) COMP-5 SYNC.
             10 CUSTXLN                         PIC X(255).
             10 CUSTXFN                         PIC X(255).
             10 CUSTXADDR1                      PIC X(255).
             10 CUSTXCITY                       PIC X(255).
             10 CUSTXST                         PIC X(255).
             10 CUSTXCTRY                       PIC X(255).
             10 DEPT                            PIC X(255).
```

- CUSTOMER.cpy represents the Web Service requester driver program. In this program we populate the request language structure into the SOAP container, invoke the Web Service, receive the response language structure, and populate the results back to the COMMAREA. The main pieces of this driver program are shown in Example 5-4.

Example 5-4 Excerpts of the CUSTOMER.cpy source

```
* -------------------------------------------------------------
* Put Request Language Structure Into SOAP Container
* ------------------------------------------------------------
           EXEC CICS PUT CONTAINER(WS-CONTAINER-NAME)
             CHANNEL(WS-CHANNEL-NAME)
             FROM(LANG-CSI01)
           END-EXEC
           PERFORM CHECK-CONTAINER-COMMAND
* *-------------------------------------------------------------
* Invoke The Web Service
* *-------------------------------------------------------------
           EXEC CICS INVOKE WEBSERVICE(WS-WEBSERVICE-NAME)
             CHANNEL(WS-CHANNEL-NAME)
URI(WS-URI-OVERRIDE)
             OPERATION(WS-OPERATION-NAME)
             RESP(COMMAND-RESP) RESP2(COMMAND-RESP2)
           END-EXEC
           PERFORM CHECK-WEBSERVICE-COMMAND
* ------------------------------------------------------------
*              Receive Response Language Structure
* -----------------------------------------------------------
           EXEC CICS GET CONTAINER(WS-CONTAINER-NAME)
             CHANNEL(WS-CHANNEL-NAME)
             INTO(LANG-CSO01)
           END-EXEC
           PERFORM CHECK-CONTAINER-COMMAND
```

Note: This program will be compiled, linked, defined, and installed into CICS.

5.3.4 Creating the CICS requester pipeline

In this section, we describe how to create the CICS requester pipeline. This is done on the z/OS system and you can either use a terminal emulator of your choice to do this or use the WDz host emulation facilities.

The complete definition of a pipeline consists of a PIPELINE resource and a pipeline configuration file. The file contains the details of the message handlers that will act on Web Service requests and responses as they pass through the pipeline.

The CICS CEDA transaction is used for this task. When you have logged in to the CICS system, type the following command. Use your own group name, if appropriate.

```
CEDA DEF PIPE(PIPECUST) G(ITSOTEST)
```

You must specify the additional attributes. For our example we entered the following additional attributes:

- STATUS(Enabled)
- CONFIGFILE(/u/itso01/itso/cicsRequester.xml)
- SHELF(/u/itso01/itso/shelf/)
- WSDIR(/u/itso01/itso/wspickup/requester)

Refer to Example 5-5 to specify this pipeline.

Example 5-5 Details of the PIPECUST definitions

```
OVERTYPE TO MODIFY OR PRESS ENTER TO EXECUTE              CICS RELEASE = 0640
   CEDA  DEFine PIpeline( PIPECUST )
    PIpeline     ==> PIPECUST
    Group        ==> ITSOTEST
    Description  ==>
    STatus       ==> Enabled            Enabled | Disabled
    Configfile   ==> /u/itso01/itso/cicsRequester.xml
    (Mixed Case) ==>
                 ==>
                 ==>
                 ==>
    SHelf        ==> /u/itso01/itso/shelf/
    (Mixed Case) ==>
                 ==>
                 ==>
                 ==>
    Wsdir          : /u/itso01/itso/wspickup/requester
    (Mixed Case)   :
 +                 :

                                                           SYSID=WRKS APPLID=CITSO01

 PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

You can check the PIPELINE definition by typing `CEMT INQ PI(PIPECUST)` and pressing the Enter key. The results are shown in Example 5-6.

Example 5-6 PIPELINE definition

```
INQ PI(PIP*)
 RESULT - OVERTYPE TO MODIFY
   Pipeline(PIPECUST)
   Enablestatus( Enabled )
   Configfile(/u/itso01/itso/cicsRequester.xml)
   Shelf(/u/itso01/itso/shelf/)
   Wsdir(/u/itso01/itso/wspickup/requester/)
```

```
                                                         SYSID=WRKS APPLID=CITSO01
                                                   TIME:  18.26.25  DATE: 04.05.08
PF 1 HELP 2 HEX 3 END       5 VAR        7 SBH 8 SFH        10 SB 11 SF
```

After having defined the PIPELINE, you install it by entering the following command:

```
CEDA I PI(PIPECUST) G(ITSOTEST)
```

Note: In our example, the name of the PIPELINE is PIPECUST and the group is ITSOTEST. Substitute suitable values for your scenario.

5.3.5 Creating the CICS requester Web Service

The WSBind file needs to be copied from the WDz workbench to the pickup directory on the z/OS system[1]. Make sure to transfer this file in binary format.

Enter the PIPELINE scan command, as follows:

```
CEMT PERFORM PIPELINE(PIPECUST) SCAN
```

When this command is issued, CICS searches the pickup directory for files ending in ".wsbind". for each wsbind file, CICS dynamically creates the WEBSERVICE resource and associates it with the PIPELINE and install them ready for use.

Note: The pickup directory /u/itso01/itso/wspickup/requester/ is where the WSBind file generated by the WDz wizard from the WSDL must be copied to in order to implement the Web Service.

You can view the Web Services defined in your system by issuing the following command:

```
CEMT INQ WEBSERVICE(*)
```

[1] /u/itso01/itso/wspickup/requester

This command will show the Web Service definitions available in the CICS system, as shown in Example 5-7.

Example 5-7 Web Service definitions in CICS

```
INQ WEBSERVICE(*)
 STATUS:  RESULTS - OVERTYPE TO MODIFY
  Webs(CATAL                          ) Pip(PIPEITSO)
     Ins Uri($339070 ) Pro(DFHMADPL) Cha Con(DFHWS-DATA      )
Dat(20070515)
  Webs(CS                             ) Pip(PIPECUST)
     Ins
Dat(20070916
  Webs(CSJMS                          ) Pip(PCUSTJMS)
     Ins
Dat(20070929
  Webs(CUSTINQ                        ) Pip(PIPEITSO)
     Ins Uri($322570 ) Pro(CUSTINQ ) Com
Dat(20070515)
  Webs(CUSTJMS                        ) Pip(PCUSTJMS)
     Unu
  Webs(EotCustFacadeXXBean             ) Pip(ITSOREQ )
     Ins
Dat(20070515



                                                     SYSID=WRKS
APPLID=CITSO01
  RESPONSE: NORMAL                             TIME:  18.30.55  DATE:
04.05.08
PF 1 HELP       3 END       5 VAR       7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

You may select a Web Service for more details by typing an `s` in the list. Our Web Service is called CS. The results are shown in Example 5-8.

Example 5-8 Details of the Web Service

```
INQ WEBSERVICE(*)
 RESULT - OVERTYPE TO MODIFY
   Webservice(CS)
   Pipeline(PIPECUST)
   Validationst( Novalidation )
   State(Inservice)
   Urimap()
   Program()
   Pgminterface(Notapplic)
   Container()
   Datestamp(20070916)
   Timestamp(15:58:16)
   Wsdlfile()
   Wsbind(/u/itso01/itso/wspickup/requester/CS.wsbind)
   Endpoint(http://wtscz1.itso.ibm.com:8108/CS/services/CS)
   Binding(CS)


SYSID=WRKS APPLID=CITSO01
                                                TIME:  18.32.54  DATE: 04.05.08
PF 1 HELP 2 HEX 3 END       5 VAR        7 SBH 8 SFH       10 SB 11 SF
```

5.3.6 Modifying the driver and installing the COBOL programs into CICS

We will modify the auto generated COBOL program[2] to receive the COMM area definition from the FINDCUST program and populate the request object. We will also add code to populate the COMM area with the response from the Web Service call. Example 5-9 shows the COMM area structure.

Example 5-9 COMM area structure

```
02  CustNo       PIC S9(9) COMP-5 .
02  LastName     PIC A(25).
02  FirstName    PIC A(15).
02  Address1     PIC X(20).
02  City         PIC A(20).
02  State        PIC A(5).
02  Country      PIC X(15).
02  RetCode      PIC S9.
```

CUSTOMER and FINDCUST make use of this structure to communicate with each other. The FINDCUST program sends a BMS map and accepts the Customer number as input. If a valid customer number is entered, it executes a EXEC CICS LINK[3] to the CUSTOMER program, which then makes the Web Service request and sends the data back via the COMMAREA.

Populate the input Customer number from the COMM area into the Web Service request as shown in Example 5-10. CustNo is the field definition in the COMM area and CUSTXNO is the field definition in the generated COBOL request copybook.

[2] The CUSTOMER program

[3] EXEC CICS LINK PROGRAM ('CUSTOMER')
COMMAREA (WS-COMAREA)
END-EXEC

Example 5-10 Populating the Web Service request

```
* -------------------------------------------------------------
*             Populate Request Language Structure
* -------------------------------------------------------------
INITIALIZE LANG-CSI01[4]
MOVE CustNo  TO CUSTXNO OF LANG-CSI01
```

You may choose to change the end-point by specifying the WS-OVERRIDE parameter for the Web Service call (Example 5-11 on page 225). In this example, we override the end-point with TCP/IP monitor address end-point, so we can track the SOAP message call from this CICS CUSTOMER COBOL program to the Web Service provider. We give more details about this monitor in 5.3.7, “Testing” on page 232 and “Configuring the TCP/IP monitor” on page 233.

Example 5-11 Specifying the WS-URI-OVERRIDE

```
MOVE 'http://9.12.5.229/CS/services/CS' TO WS-URI-OVERRIDE.
EXEC CICS INVOKE WEBSERVICE(WS-WEBSERVICE-NAME)
             CHANNEL(WS-CHANNEL-NAME)
             URI(WS-URI-OVERRIDE)
             OPERATION(WS-OPERATION-NAME)
             RESP(COMMAND-RESP) RESP2(COMMAND-RESP2)
 END-EXEC
```

Next, we add the code in Example 5-12 to move the Web Service result into the COMM area. This code is placed under the “Process Response Language Structure” comment.

Example 5-12 Getting values from the Response Language Structure

```
MOVE  CUSTXLN  TO LastName.
MOVE  CUSTXFN   TO FirstName.
MOVE  CUSTXADDR1  TO Address1 .
MOVE  CUSTXCITY   TO City .
MOVE  CUSTXST    TO State .
MOVE  CUSTXCTRY TO Country .
MOVE ZEROES TO RetCode .
```

[4] The LANG-CSI01 is the structure representing the input COBOL data structure. The tool automatically generated code for importing the data structure:
1 LANG-CSI01.
        COPY CSI01.
1 LANG-CSO01.
        COPY CSO01.

Compiling and linking the CUSTOMER and FINDCUST programs[5]

We utilize the WDz environment to accomplish this step. In the WDz environment, switch to the z/OS projects perspective. This is the perspective where the COBOL builds and JCL generations are supported.

Create a new z/OS project and then copy over the assets[6] that we are working on to the subproject inside this project. This simplifies the task because we now have all the assets that are being used, grouped in the same location. We need to copy these files to the System z environment.

For creating a new z/OS connection, refer to Chapter 2, “Accessing CICS from WebSphere using Web Services over HTTP” on page 3 and 2.5.1, “Executing the sample application” on page 21.

[5] This step varies according to the environment. Refer to your CICS programming support group for guidance on this.
[6] FINDCUST, CUSTOMER,CUSI01,CUSO01

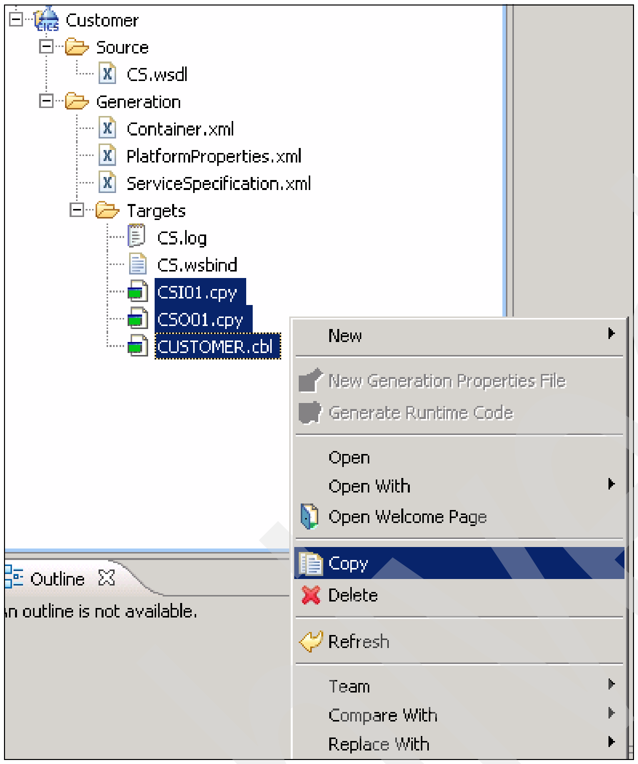


Figure 5-13 Copying files from the EST perspective

We can copy the files in the EST perspective, then switch to the z/OS projects perspective and paste the file into the appropriate MVS data set, as shown in Figure 5-14.

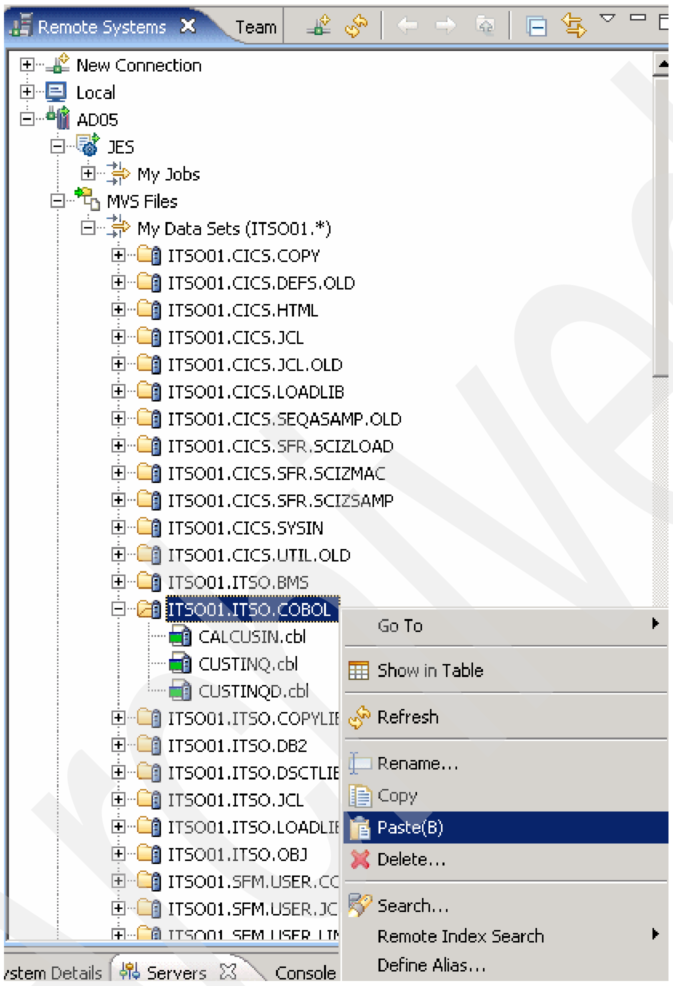


Figure 5-14 Pasting files copied from EST perspective into the MVS data set

You may want to add these artifacts to your z/OS subproject as the Development/Test/Debug cycle may warrant further changes to your program. You must have one or more subprojects in the Projects view, and you must be connected to the system where the subprojects are defined. The “Add to Sub projects” menu shown in Figure 5-15 helps you to accomplish this.

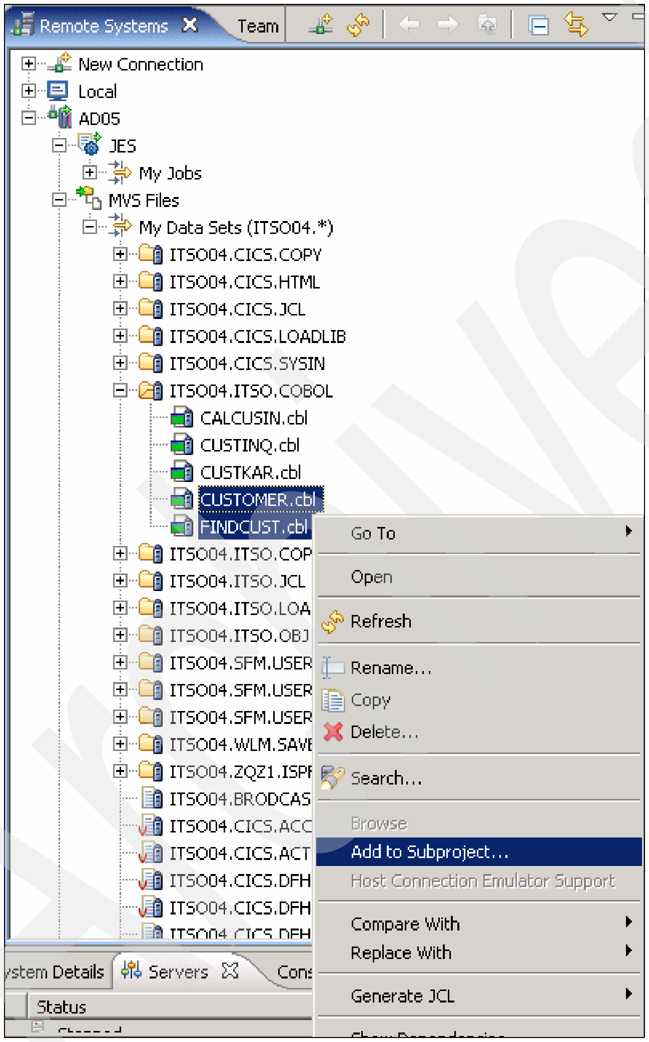


Figure 5-15 Add to sub project menu

You can use the **Generate JCL** → **For Compile Link** action to generate a JCL script that you can submit to compile and link the COBOL file. This is shown in Figure 5-16 on page 230.

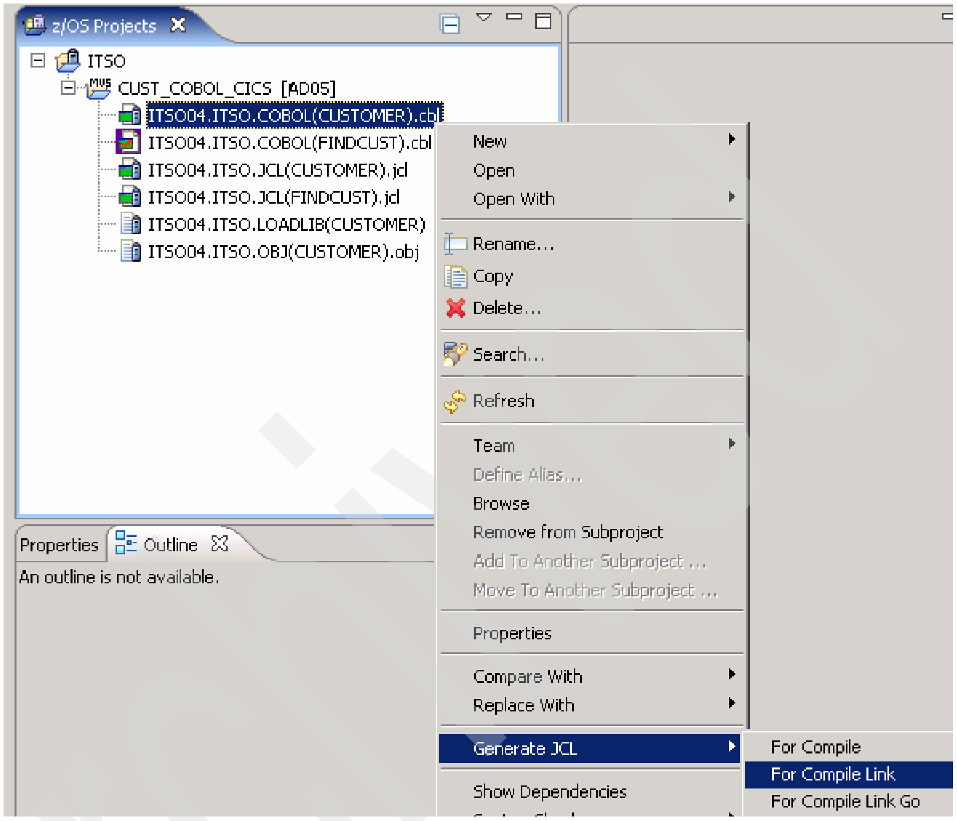


Figure 5-16 Generating the JCL

You can compile and build by choosing the JCL and using the submit action as shown in Figure 5-17.

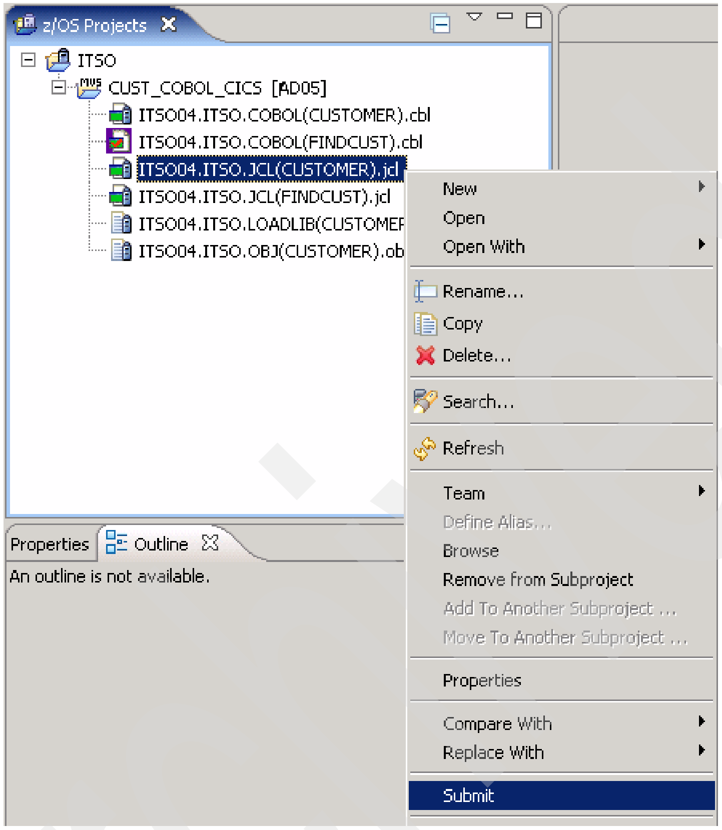
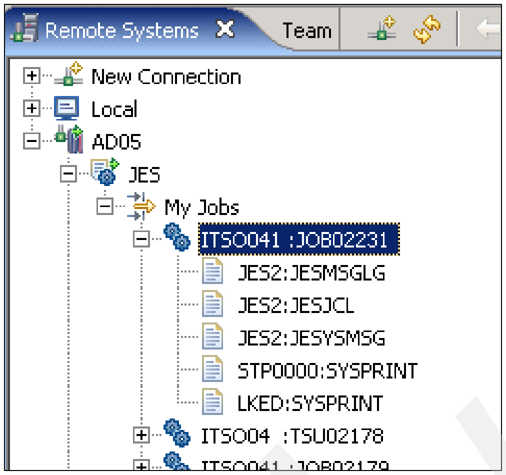

z/OS Projects
ITSO
CUST_COBOL_CICS [AD05]
ITSO04.ITSO.COBOL(CUSTOMER).cbl
ITSO04.ITSO.COBOL(FINDCUST).cbl
ITSO04.ITSO.JCL(CUSTOMER).jcl
ITSO04.ITSO.JCL(FINDCUST).jcl
ITSO04.ITSO.LOADLIB(CUSTOMER
ITSO04.ITSO.OBJ(CUSTOMER).ob
Properties
Outline
An outline is not available.
New
Open
Open With
Rename...
Copy
Delete...
Search...
Refresh
Team
Define Alias...
Browse
Remove from Subproject
Add To Another Subproject ...
Move To Another Subproject ...
Properties
Compare With
Replace With
Submit

Figure 5-17 Submitting the JCL

You can view the results of the compile and link in JES by browsing JES in the Remote System view, as shown in Figure 5-18.



```
New Connection
Local
AD05
  JES
    My Jobs
      ITSO041 :JOB02231
        JES2:JESMSGLG
        JES2:JESJCL
        JES2:JESYSMSG
        STP0000:SYSPRINT
        LKED:SYSPRINT
      ITSO04  :TSU02178
      ITSO041 :JOB02179
```

Figure 5-18 Viewing the JES logs

After the program has been successfully compiled, we must define and install the FINDCUST and CUSTOMER programs in CICS. We must also define and install a new transaction in CICS to call into the FINDCUST program. For this example we created a transaction named GCUS that invokes the FINDCUST program.

The CICS CEDA command is used to accomplish these tasks.

5.3.7 Testing

We configure a TCP/IP monitor to track the Web Service request originating from the CICS program. In this scenario, all requests and responses are routed through the TCP/IP monitor and appear in the TCP/IP Monitor view and help us monitor the remote Web Service call.

The TCP/IP Monitor view shows all the intercepted requests in the top pane, and when a request is selected, the messages passed in each direction are shown in the bottom panes (request in the left pane, response in the right). This can be a very useful tool in debugging Web Services and clients.

Configuring the TCP/IP monitor

1. In WDz choose **Window** → **Show View** → **Other**, as shown in Figure 5-19.
2. Choose **TCP/IP Monitor** under the Debug category in the pop-up, as shown in Figure 5-19 on page 233.

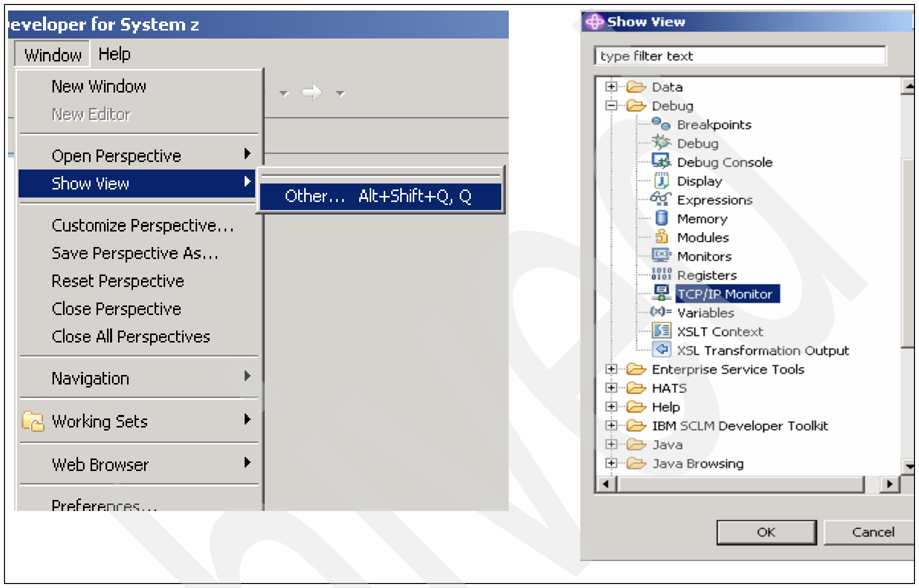


Figure 5-19 Getting to TCP/IP monitor view

3. In the monitor window, right-click and choose **Properties**, as shown in Figure 5-20.

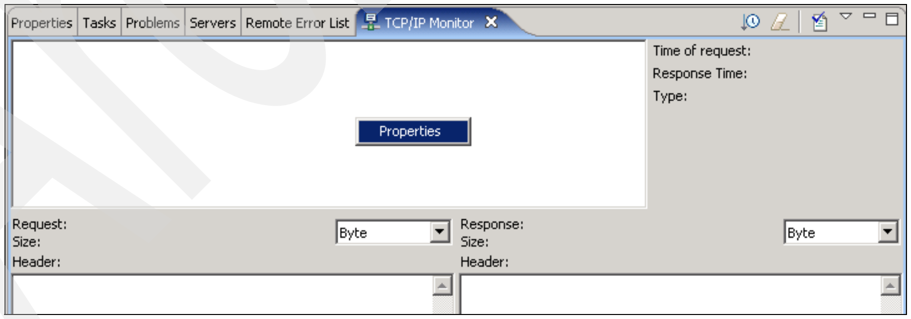


Figure 5-20 Defining a monitor

4. In the pop-up window, click **Add**, as shown in Figure 5-21.

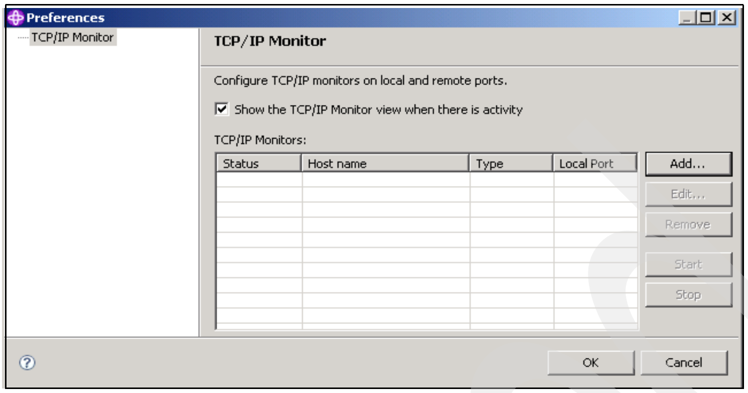
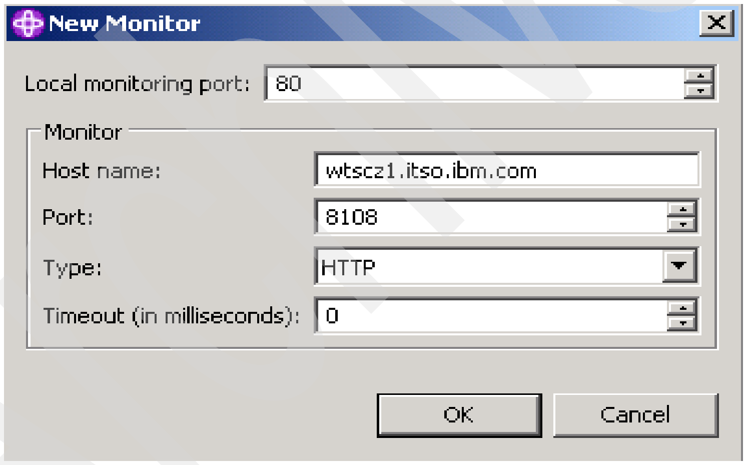


Figure 5-21 Adding a monitor

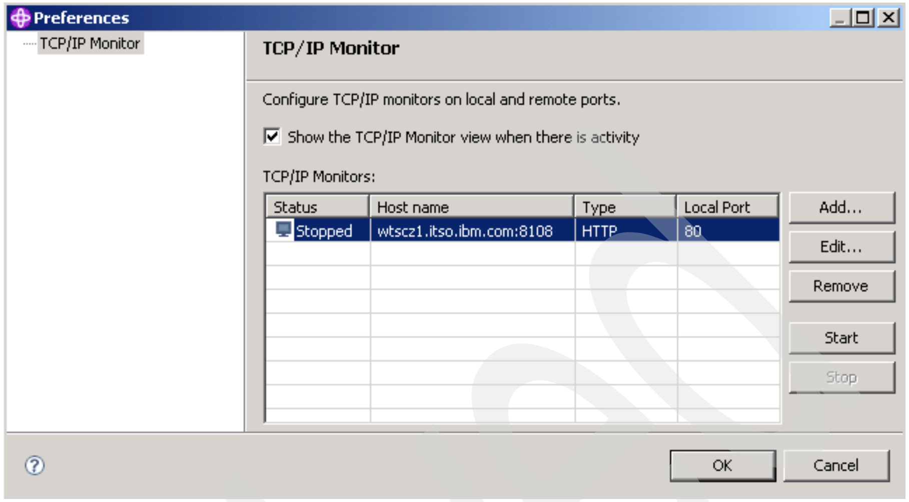
5. Specify the location where the Web Service provider is running (host name and port) and click **OK**. See Figure 5-22.



Local monitoring port: 80
Host name: wtsc21.itso.ibm.com
Port: 8108
Type: HTTP
Timeout (in milliseconds): 0

Figure 5-22 Specifying the Web Service service provider host and port

6. Start the monitor by clicking **Start**. See Figure 5-23.



TCP/IP Monitor

Configure TCP/IP monitors on local and remote ports.

☑ Show the TCP/IP Monitor view when there is activity

TCP/IP Monitors:

Status	Host name	Type	Local Port
Stopped	wtscz1.itso.ibm.com:8108	HTTP	80

Figure 5-23 Start a monitor

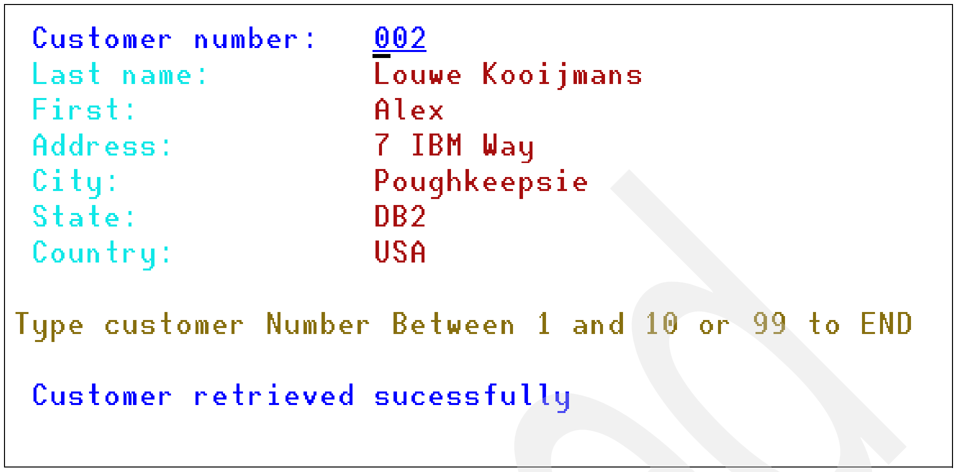
Invoking the Web Service call and getting back the response

Invoke the FINDCUST CICS program by typing in the GCUS transaction on the host system. See Example 5-13.

Example 5-13 FINDCUST transaction

```
Client Inquiry - calls CUSTINQ  (ITSOBMS)

 Customer number:   2
 Last name:
 First:
 Address:
 City:
 State:
 Country:

Type customer Number Between 1 and 10 or 99 to END
```

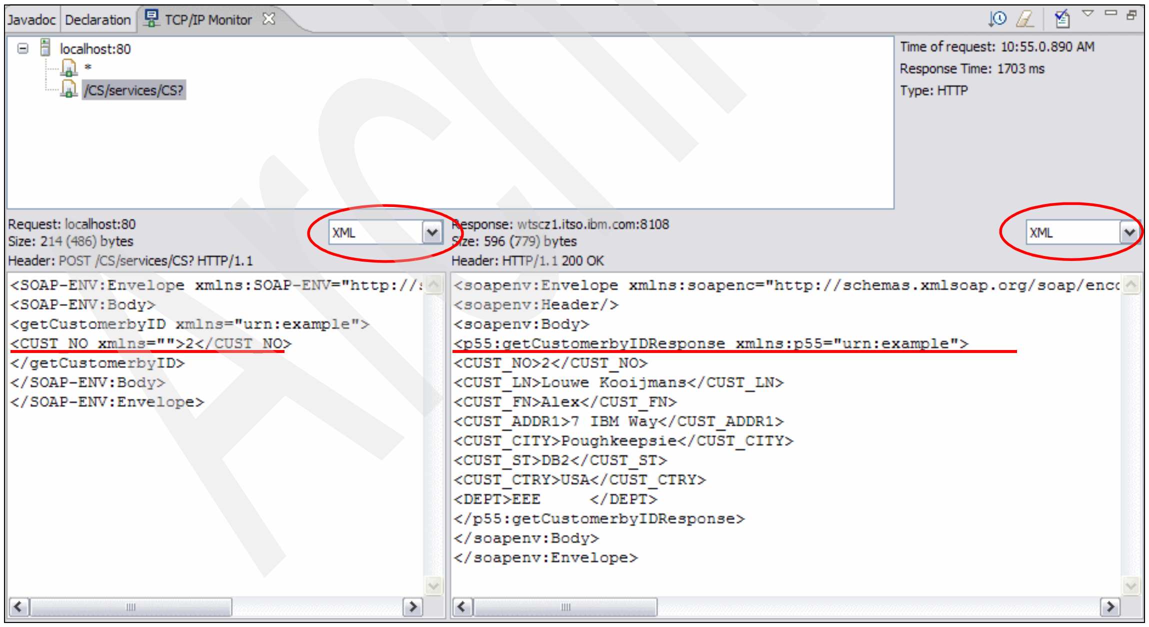
Enter a value into the Customer number field. In this example we enter 2 and press **Enter**. Results are shown in Figure 5-24.



```
 Customer number:    002
 Last name:          Louwe Kooijmans
 First:              Alex
 Address:            7 IBM Way
 City:               Poughkeepsie
 State:              DB2
 Country:            USA

Type customer Number Between 1 and 10 or 99 to END

 Customer retrieved sucessfully
```

Figure 5-24 Results from a call to the Web Service provider

The TCP/IP Monitor view shows all the intercepted requests in the top pane, and when a request is selected, the messages passed in each direction are shown in the bottom panes (request in the left pane, response in the right). Select the XML view to display the SOAP request and response in XML format (Figure 5-25).



```
Time of request: 10:55.0.890 AM
Response Time: 1703 ms
Type: HTTP
```

```
Request: localhost:80
Size: 214 (486) bytes
Header: POST /CS/services/CS? HTTP/1.1
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://
<SOAP-ENV:Body>
<getCustomerbyID xmlns="urn:example">
<CUST_NO xmlns="">2</CUST_NO>
</getCustomerbyID>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
Response: wtsczl.itso.ibm.com:8108
Size: 596 (779) bytes
Header: HTTP/1.1 200 OK
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/enc
<soapenv:Header/>
<soapenv:Body>
<p55:getCustomerbyIDResponse xmlns:p55="urn:example">
<CUST_NO>2</CUST_NO>
<CUST_LN>Louwe Kooijmans</CUST_LN>
<CUST_FN>Alex</CUST_FN>
<CUST_ADDR1>7 IBM Way</CUST_ADDR1>
<CUST_CITY>Poughkeepsie</CUST_CITY>
<CUST_ST>DB2</CUST_ST>
<CUST_CTRY>USA</CUST_CTRY>
<DEPT>EEE     </DEPT>
</p55:getCustomerbyIDResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Figure 5-25 SOAP request and response in XML format

5.4 Additional material

Table 5-1 describes all the material that has been supplied with this chapter. Refer to Appendix B, “Additional material” on page 397 for details about downloading.

Table 5-1 Supplied materials and their purpose

Name	Purpose
CS.wsdl	The WSDL used as starting point.
CS.war	The Web Service provider WAR file. Installed with context root of CS and using the same data source and database table as referenced in Chapter 3, “Accessing DB2 from WebSphere using Web Services” on page 85.
CSPI.zip	The project interchange file of the EST project in WDz referenced in “Importing the WSDL of Web Service to be called” on page 212.
CUSTOMER.cbl	The modified driver program
FINDCUST.cbl	CICS program to gather user input
ITSOBMS.bms	The CICS BMS map to show the screen
ITSOBMS.cpy	BMS interaction program
COMAREA.cpy	COMArea definition

6

Defining a messaging infrastructure in WebSphere Application Server

This chapter provides information about the configuration of WebSphere Application Server and WebSphere MQ to build the infrastructure necessary for Web Services communication in and out of WebSphere Application Server using JMS.

Our end-to-end solution consisted of WebSphere Web Services consumer application as well as CICS back-end application accessible through Web Services protocols. In this chapter we discuss the architecture, requisites and implementation aspects of WebSphere Application Server and WebSphere MQ. Configuration of CICS for sending and receiving messages over WebSphere MQ is not covered. For this part we refer you to *Developing Web Services Using CICS, WMQ, and WMB*, SG24-7425.

6.1 High-level architecture

The Service Integration Bus (SIB) is a powerful built-in feature of WebSphere Application Server. It enables WebSphere Application Server to use its own messaging provider, including mediation.

In this chapter we show how to integrate WebSphere Application Server with CICS using the default messaging provider.

WebSphere Application Server provides the J2EE container to run the Java application that consumes the Web Service hosted in CICS. By configuring the SIB and the default message provider we can make WebSphere act as a complete messaging system.

In the SIB we define a *bus* and a Messaging Engine (ME). The application accesses the bus through the JMS API to provide or consume the messages.

To complete the environment we need a WebSphere MQ (WMQ) and CICS TS. WMQ hosts the objects and triggers the CICS Web Service.

Figure 6-1 shows the architecture for this scenario.

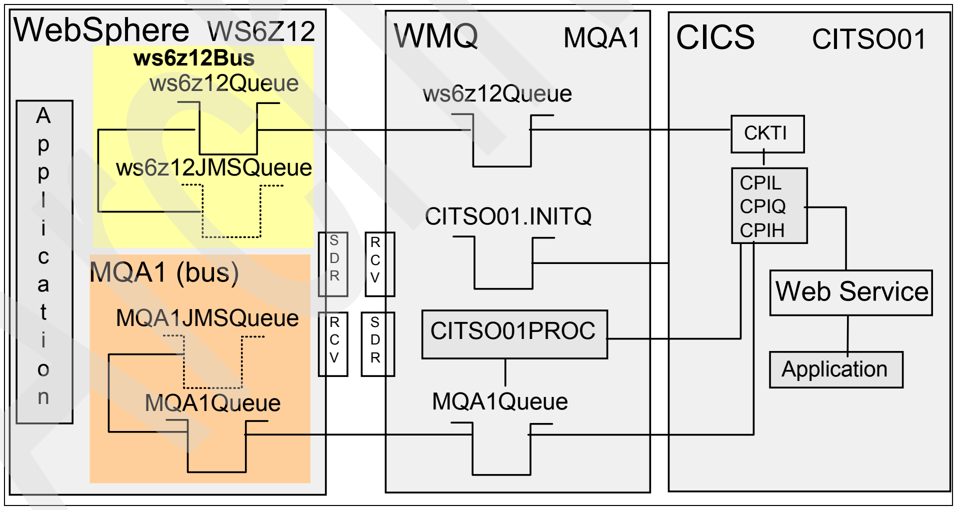


Figure 6-1 Technical architecture for WebSphere Application Server-WMQ-CICS scenario

6.2 Infrastructure setup overview

The objective of this book is to demonstrate the configuration of the connectivity of WebSphere and not the installation of the product. We are listing the most important aspects of the infrastructure.

We used z/OS Version 1.9, CICS TS Version 3.1, WebSphere Version 6.1.0.10, and WebSphere MQ Version 6 for z/OS.

6.2.1 Components

WebSphere Application Server for z/OS Version 6.1.0.10

WebSphere Application Server is the J2EE and Web Services application server that runs the front-end application used to test the solution.

CICS Transaction Server Version 3.1

CICS is the transaction server where the legacy application resides. It is not within the scope of this book to discuss the configuration of CICS for receiving and sending JMS messages. For details, refer to *Developing Web Services Using CICS, WMQ, and WMB*, SG24-7425.

WebSphere MQ for z/OS Version 6

WebSphere MQ is the messaging system that exchanges the messages between WebSphere Application Server and the CICS back-end application.

Note that even though we use Web Services protocols, WebSphere MQ deals with the Web Services requests and responses as messages and uses the same concepts and infrastructure as any other type of message.

6.2.2 Prerequisites

WebSphere Application Server requisites

We used WebSphere Application Server Version 6.1.0.10 for z/OS. This version must be installed under z/OS v 1.6 or higher.

The WebSphere Application Server uses its own JRE instance, which is the J2RE 1.5.0 IBM J9 2.3 level.

To check the system requirements for WebSphere Application Server V6.1, refer to the following Web page:

`http://www-1.ibm.com/support/docview.wss?uid=swg27007651`

WebSphere MQ Server requisites

We used WebSphere MQ Version 6 for z/OS LEVEL 003-006. This version must be installed under z/OS v 1.4 or higher.

To check the system requirements for WebSphere MQ Version 6 for z/OS, refer to the following Web page:

http://www-1.ibm.com/support/docview.wss?uid=swg27006267

CICS Transaction Server requisites

We used CICS Transaction Server V3.1. This version must be installed under z/OS v 1.4 or higher.

CICS Transaction Server Version 3.1 APAR PK04615 must be installed to provide CICS with the capability to perform Web Services using SOAP over JMS. Additionally, the WebSphere MQ SCSQLOAD data set must be added to the DFHRPL concatenation in the CICS region. If you do not do this and try a Web Service using SOAP over JMS, you will get the following message in the CICS region:

```
DFHAP0900 SCSCERW1 MQ support for CICS Web Services is not available
```

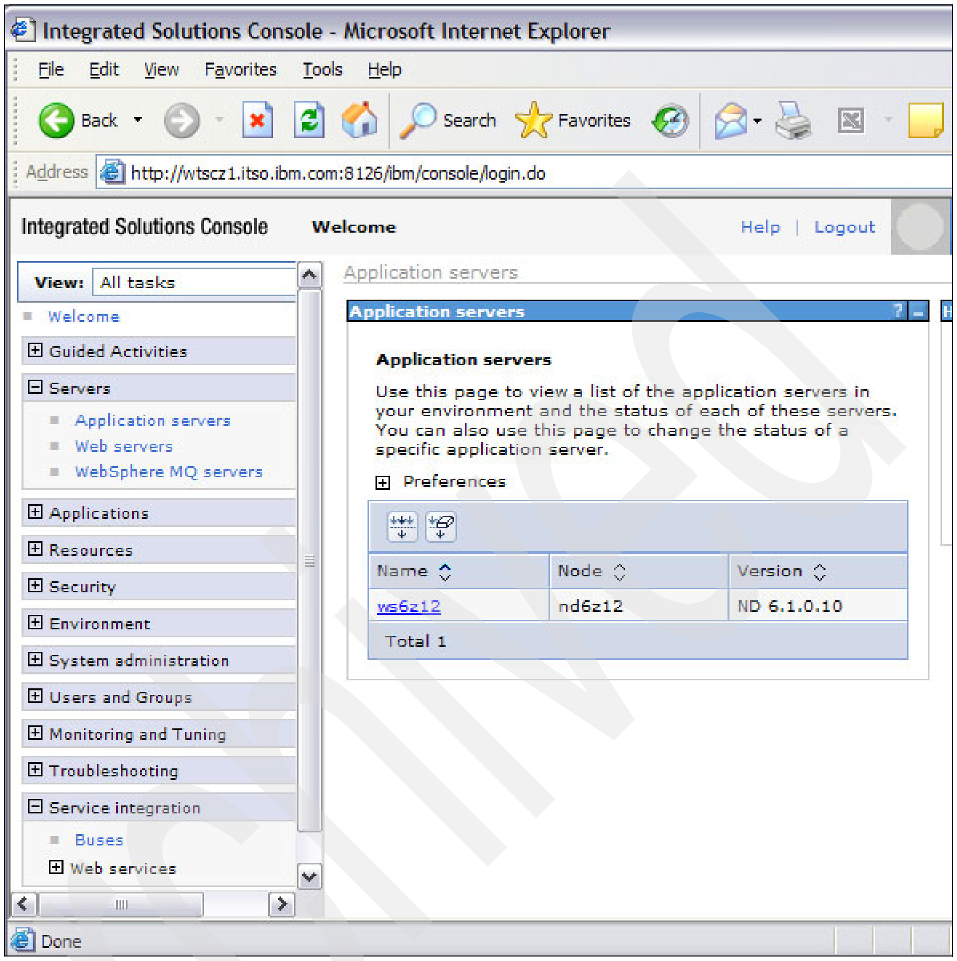
To check the CICS Transaction Server V3.1 detailed system requirements, refer to the following Web page:

http://www-1.ibm.com/support/docview.wss?uid=swg27006364

6.3 WebSphere Application Server configuration

The main configuration tool of the WebSphere Application Server on z/OS is the WebSphere Integrated Solutions Console. If you have security enabled, it is required to input user ID and password.

To query the available applications servers, click **Servers** → **Application servers**. See Figure 6-2 on page 243.



http://wtscz1.itso.ibm.com:8126/ibm/console/login.do

Application servers

Application servers

Use this page to view a list of the application servers in your environment and the status of each of these servers. You can also use this page to change the status of a specific application server.

Preferences

Name	Node	Version
ws6z12	nd6z12	ND 6.1.0.10
Total 1		

Figure 6-2 WebSphere Application Server - application servers

We discuss the following configuration tasks:

- Defining a Service Integration Bus (SIB) in “Defining a Service Integration Bus (SIB)” on page 244.
- Defining a Messaging Engine (ME) in “Defining a Messaging Engine (ME)” on page 247.
- Creating an Adjunct Server in “Adjunct server” on page 252.
- Defining the necessary queues in “Defining the queue” on page 253.
- Defining the JMS queue to the local queue in “Defining a JMS queue to the local queue” on page 257.

- Defining a JMS queue connection factory in “Defining the JMS queue connection factory” on page 260.
- Defining the JMS activation specifications in “Defining a JMS activation specification” on page 262.
- Defining a foreign bus in “Defining a foreign bus” on page 265.
- Defining a WebSphere MQ Link in “Define a WebSphere MQ Link” on page 268.
- Verifying the channels in “Verifying the channels” on page 274.
- Defining the WebSphere MQ channels in “Defining the channels in WebSphere MQ” on page 277.
- Defining the foreign queue in “Defining a foreign queue” on page 283.
- Defining the JMS queue to the foreign queue “Defining the JMS queue to the foreign queue” on page 288.

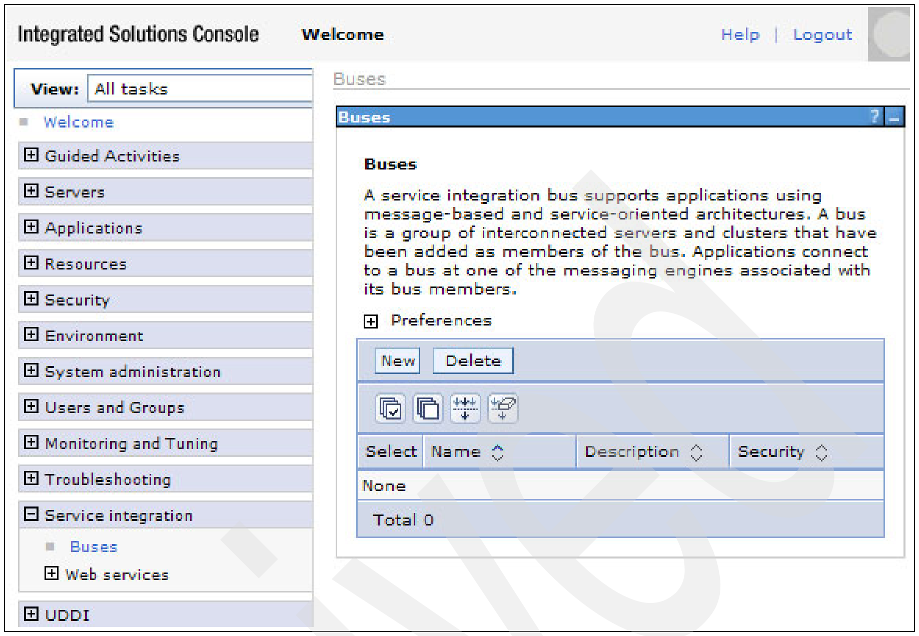
6.3.1 Defining a Service Integration Bus (SIB)

The bus object is fundamental to most other configuration tasks; it is not possible to begin creating or configuring any services or gateway resources until a bus object has been created. The *bus* is a group of one or more interconnected servers or clusters that have been added as members of the bus. Applications connect to a bus at one of the Messaging Engines associated with its bus members and can then access resources defined on the bus.

A Service Integration Bus (SIB) is an administrative concept that represents all the messaging resources in a WebSphere Application Server environment.

Further information can be found at the WebSphere Application Server Infocenter.

1. To create an SIB, using the WebSphere Integrated Solutions Console, expand **Service Integration** → **Buses** and click **New**. See Figure 6-3 on page 245.



Buses

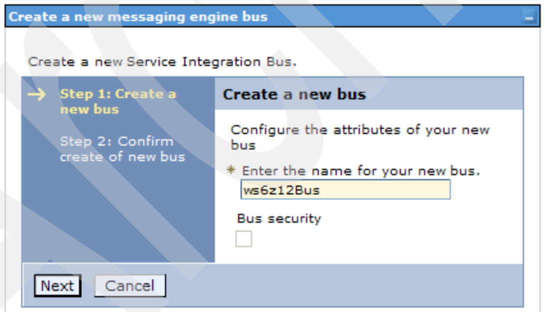
A service integration bus supports applications using message-based and service-oriented architectures. A bus is a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

Preferences

Select	Name	Description	Security
None			
Total 0			

Figure 6-3 Show Service Integration Bus

2. On the Create a new Service Integration Bus screen, in “Step 1: Create a new bus”, type ws6z12Bus in the name field, unselect the **Bus security** box, and click **Next**, as shown in Figure 6-4.


Create a new messaging engine bus
Step 1: Create a new bus
Step 2: Confirm create of new bus
Create a new bus
Next
Cancel

Create a new Service Integration Bus.

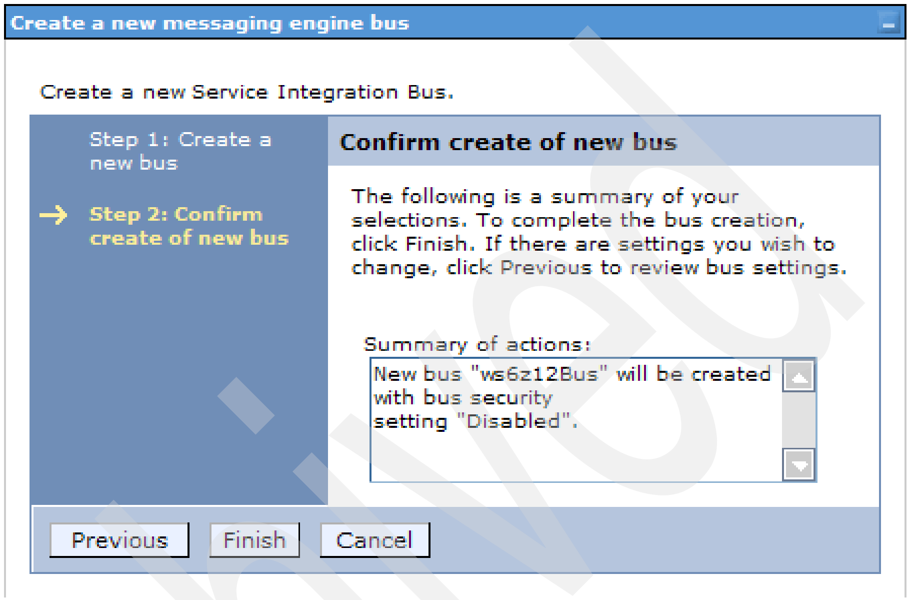
Configure the attributes of your new bus

* Enter the name for your new bus.
ws6z12Bus

Bus security
☐

Figure 6-4 Create a new Service Integration Bus (SIB)

3. In “Step 2: Confirm create of new bus” verify the information in the Summary of actions field. If everything is correct, click **Finish**. If you notice any problem click on the **Previous** button till you get the corresponding panel to change. See Figure 6-5.


Create a new messaging engine bus
Step 1: Create a new bus
Step 2: Confirm create of new bus
Previous
Finish
Cancel

Create a new Service Integration Bus.

Confirm create of new bus

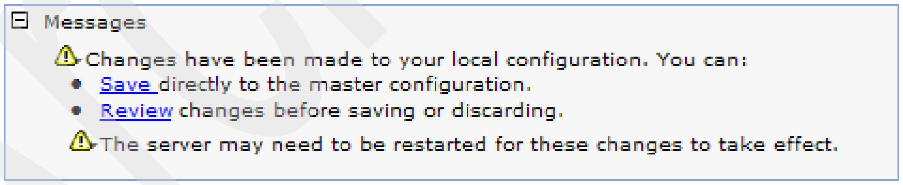
The following is a summary of your selections. To complete the bus creation, click Finish. If there are settings you wish to change, click Previous to review bus settings.

Summary of actions:

New bus "ws6z12Bus" will be created with bus security setting "Disabled".

Figure 6-5 Create new bus - Finish

4. Click **Save**.

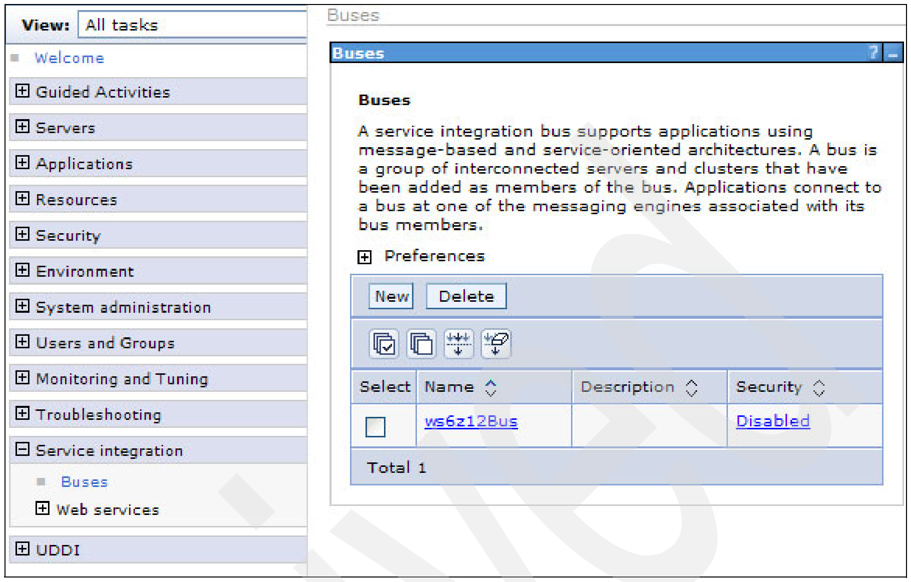


⚠ Changes have been made to your local configuration. You can:
- Save directly to the master configuration.
- Review changes before saving or discarding.

⚠ The server may need to be restarted for these changes to take effect.

Figure 6-6 Save Changes

At this point the bus ws6z12Bus is created. See Figure 6-7.



Buses

A service integration bus supports applications using message-based and service-oriented architectures. A bus is a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

⊞ Preferences

Select	Name	Description	Security
☐	ws6z12Bus		Disabled
Total 1			

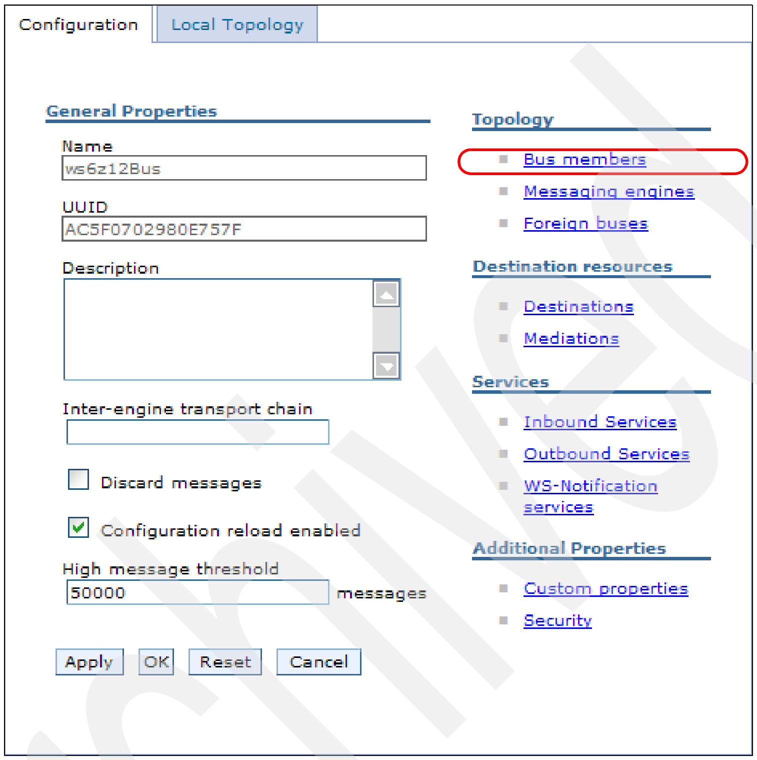
Figure 6-7 Show Service Integration Bus

6.3.2 Defining a Messaging Engine (ME)

A Messaging Engine (ME) is a server component that provides the core messaging functions of a Service Integration Bus. The Messaging Engine manages bus resources and allows applications to communicate with the bus.

To define a Messaging Engine, proceed as follows:

- Click **ws6z12Bus** on the Buses window. It shows you the SIB general properties screen. Click **Bus members**, as shown in Figure 6-8.



Name: ws6z12Bus

UUID: AC5F0702980E757F

Description:

Inter-engine transport chain:

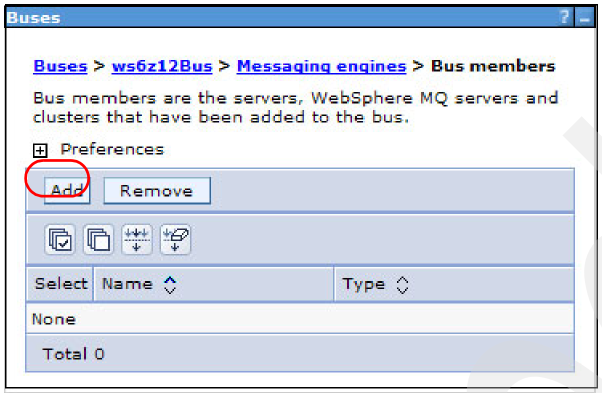
[ ] Discard messages

[✓] Configuration reload enabled

High message threshold: 50000 messages

Figure 6-8 SIB - General Properties

5. On the Bus members window, click **Add**, as shown in Figure 6-9.

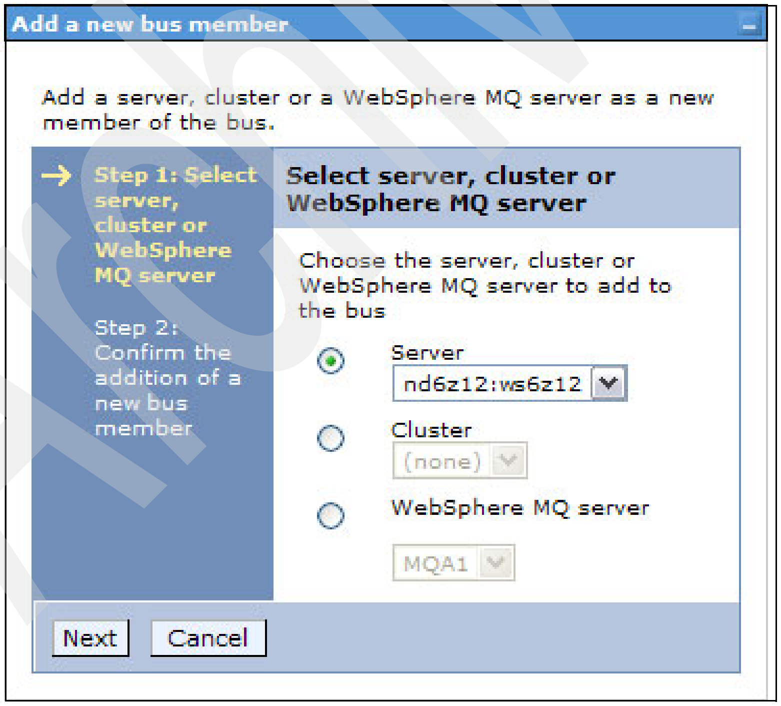


Bus members are the servers, WebSphere MQ servers and clusters that have been added to the bus.

Select	Name	Type
None		
Total 0		

Figure 6-9 Show Messaging Engines

6. In "Step 1: Select server, cluster or WebSphere MQ server" select the option **Server,** the server name **nd6z12.ws6z12** on the Server combo-box, and click **Next**, as shown in Figure 6-10.



Add a server, cluster or a WebSphere MQ server as a new member of the bus.

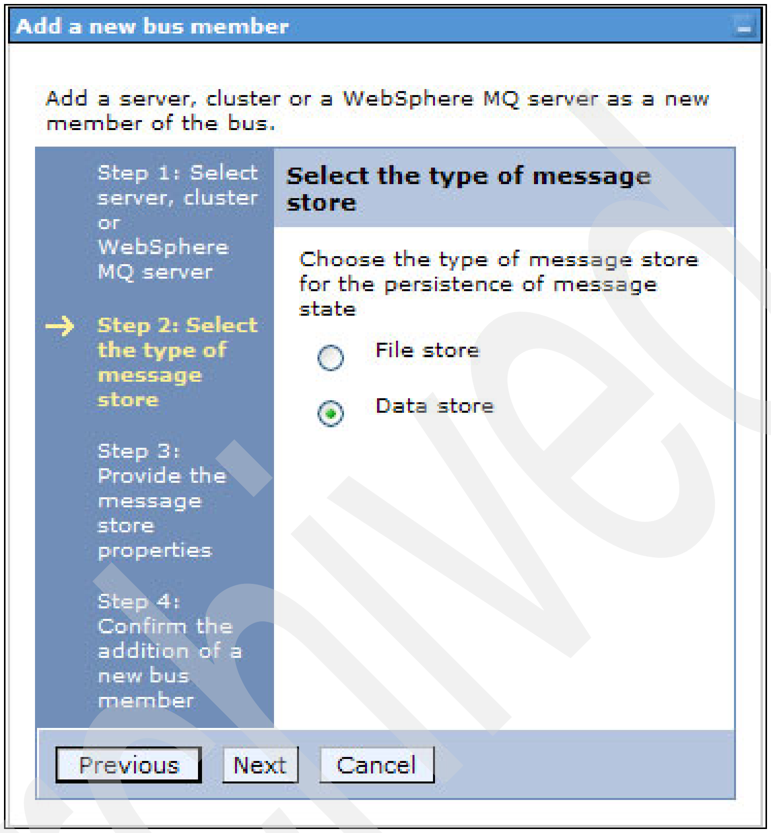
Step 1: Select server, cluster or WebSphere MQ server
Step 2: Confirm the addition of a new bus member
Select server, cluster or WebSphere MQ server

Choose the server, cluster or WebSphere MQ server to add to the bus

[✓ Server] nd6z12:ws6z12
Cluster (none)
WebSphere MQ server MQA1

Figure 6-10 Add server as a member of the bus

7. In “Step 2: Select the type of message store” select the **Data Store** option and click **Next**, as shown in Figure 6-11.



Add a new bus member

Add a server, cluster or a WebSphere MQ server as a new member of the bus.

Step 1: Select server, cluster or WebSphere MQ server

→ Step 2: Select the type of message store

Step 3: Provide the message store properties

Step 4: Confirm the addition of a new bus member

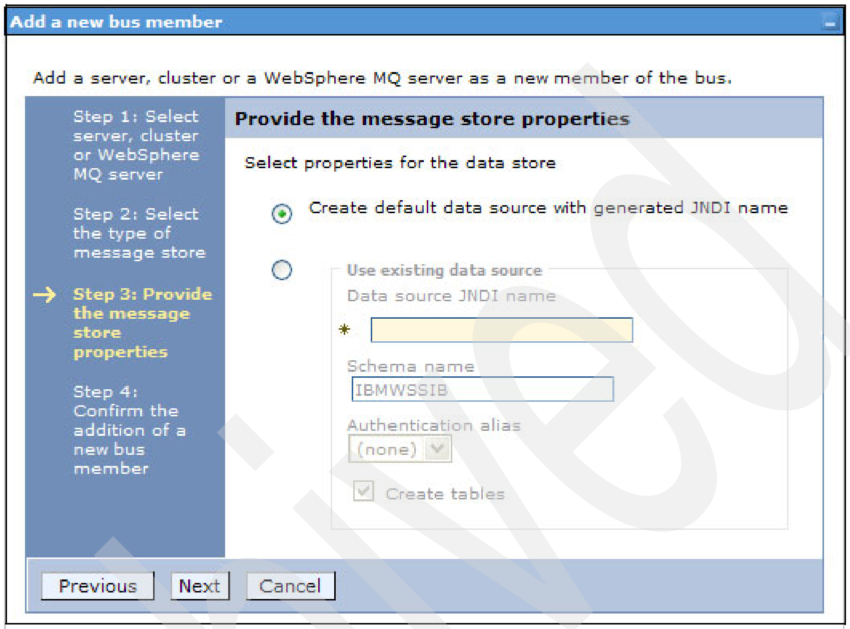
Select the type of message store

Choose the type of message store for the persistence of message state

File store [✓ Data store]

Figure 6-11 Add a new bus member - Select the type of message store

8. In “Step 3: Provide the message store properties” select the option **Create default data source with generated JNDI name**, and click **Next**. See Figure 6-12.


Add a new bus member

Add a server, cluster or a WebSphere MQ server as a new member of the bus.

Step 1: Select server, cluster or WebSphere MQ server

Step 2: Select the type of message store

→ **Step 3: Provide the message store properties**

Step 4: Confirm the addition of a new bus member

Provide the message store properties

Select properties for the data store

[● Create default data source with generated JNDI name]

○ Use existing data source

Data source JNDI name

*

Schema name

IBMWSSIB

Authentication alias

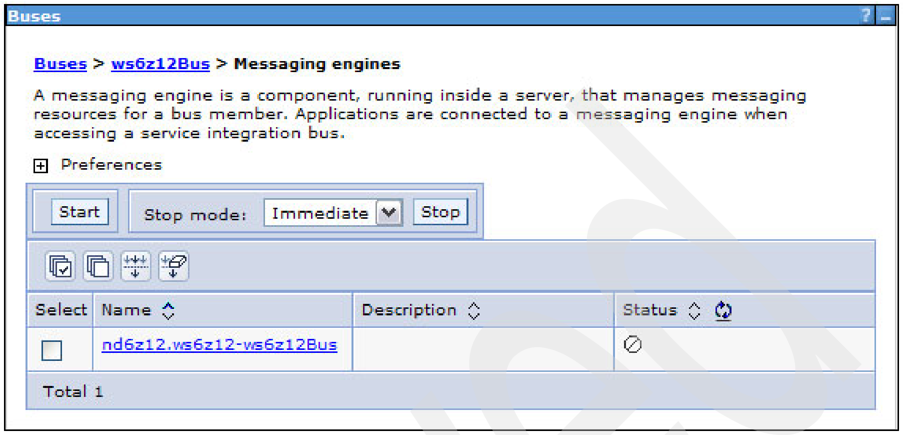
(none)

✓ Create tables

Figure 6-12 Add a new bus Member - Select properties for the data store

9. On the confirmation screen, check whether the information is correct and click **Finish**.

10. **Save** the changes, and the Messaging engines window will appear as Figure 6-13.


Buses

Buses > ws6z12Bus > **Messaging engines**

A messaging engine is a component, running inside a server, that manages messaging resources for a bus member. Applications are connected to a messaging engine when accessing a service integration bus.

⊞ Preferences

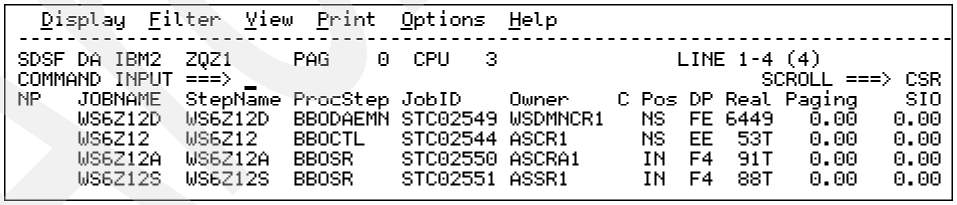
Select	Name	Description	Status
☐	nd6z12.ws6z12-ws6z12Bus		⊘
Total 1			

Figure 6-13 Message engines

6.3.3 Adjunct server

Before defining the other resources, restart the WebSphere Application Server.

If this is the first Messaging Engine you have defined for the server, you will see a new STC started along with the Control Region and Servant Region when you restart the server. This new STC is referred to as the Control Region Adjunct (CRA). The name of this STC will be the name of the Control Region with an A appended, as shown in Figure 6-14.

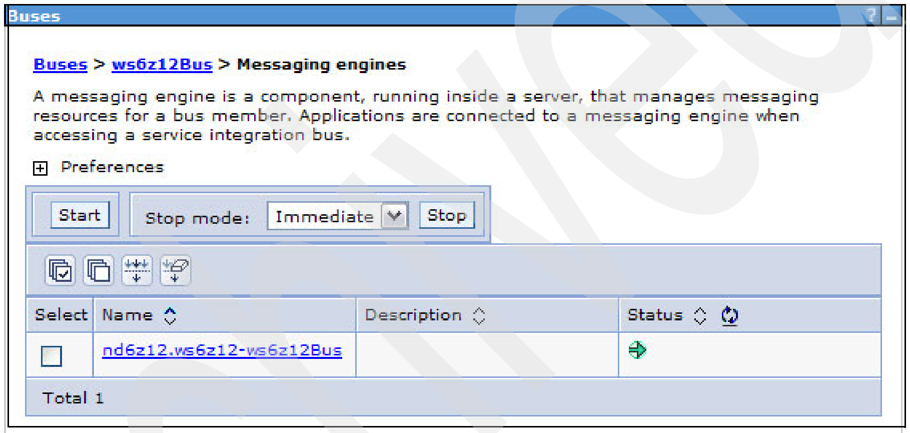

```
  Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------------
SDSF DA IBM2  ZQZ1      PAG     0  CPU    3              LINE 1-4 (4)
COMMAND INPUT ===>                                               SCROLL ===> CSR
NP     JOBNAME  StepName ProcStep JobID    Owner    C Pos DP Real Paging     SIO
       WS6Z12D  WS6Z12D  BBODAEMN STC02549 WSDMNCR1   NS  FE 6449   0.00    0.00
       WS6Z12   WS6Z12   BBOCTL   STC02544 ASCR1      NS  EE  53T   0.00    0.00
       WS6Z12A  WS6Z12A  BBOSR    STC02550 ASCRA1     IN  F4  91T   0.00    0.00
       WS6Z12S  WS6Z12S  BBOSR    STC02551 ASSR1      IN  F4  88T   0.00    0.00
```
Figure 6-14 Control Region Adjunct

The HTTP and HTTPS ports are not opened by the Control Region until both the Servant and Adjunct STCs have completed startup. In the Adjunct STC you will see messages similar to those shown in Example 6-1 on page 253, which identify the STC as being the Adjunct STC.

Example 6-1 Messages identifying the Adjunct Server

```
BBOO0222I: WSVR0001I: Server ADJUNCT PROCESS ws6z12 open for  228
e-business
.
.
BBOO0222I: "ws6z12Bus:nd6z12.ws6z12-ws6z12Bus"
CWSID0016I: Messaging  engine nd6z12.ws6z12-ws6z12Bus is in state
Started.
```

After the WebSphere restart, the Messaging Engine status appears running, as shown in Figure 6-15.



A messaging engine is a component, running inside a server, that manages messaging resources for a bus member. Applications are connected to a messaging engine when accessing a service integration bus.

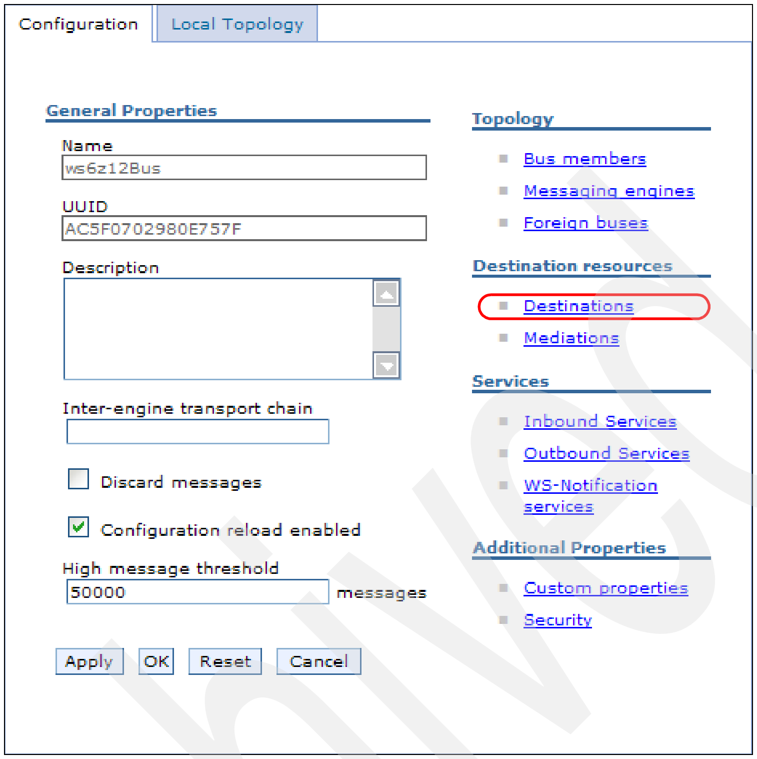
Select	Name	Description	Status
☐	nd6z12.ws6z12-ws6z12Bus		➔
Total 1			

Figure 6-15 Message engine - Status is running

6.3.4 Defining the queue

A queue is required, as this will be the place where messages for the Web Service request will be sent for processing. When a message arrives on this queue, a Message Driven Bean (MDB) will be driven to process the message. In WebSphere we actually work with the term *destination*. A destination can be of different types. Proceed as follows:

1. Select **Service Integration** → **Buses** → **ws6z12Bus**. On the Destination resources window click **Destination**, as shown in Figure 6-16.


Configuration | Local Topology

General Properties

Name
ws6z12Bus

UUID
AC5F0702980E757F

Description

Inter-engine transport chain

☐ Discard messages

☑ Configuration reload enabled

High message threshold
50000 messages

Apply | OK | Reset | Cancel

Topology
- Bus members
- Messaging engines
- Foreign buses

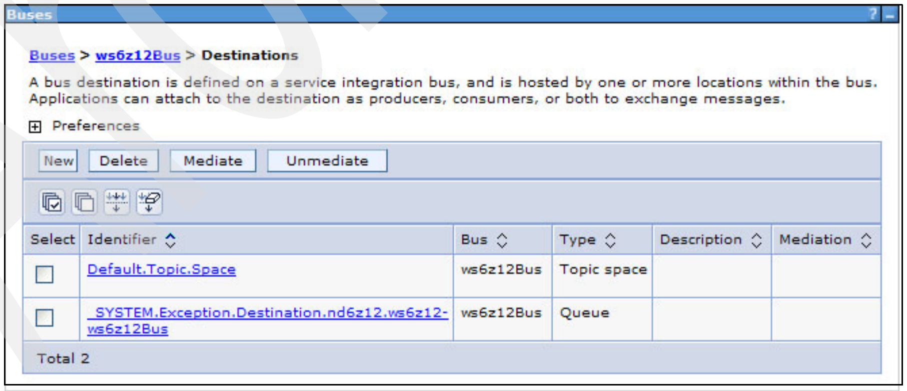
Destination resources
- Destinations
- Mediations

Services
- Inbound Services
- Outbound Services
- WS-Notification services

Additional Properties
- Custom properties
- Security

Figure 6-16 Creating a new destination

2. On the Destinations window, click **New**, as shown in Figure 6-17.


Buses

Buses > ws6z12Bus > **Destinations**

A bus destination is defined on a service integration bus, and is hosted by one or more locations within the bus. Applications can attach to the destination as producers, consumers, or both to exchange messages.

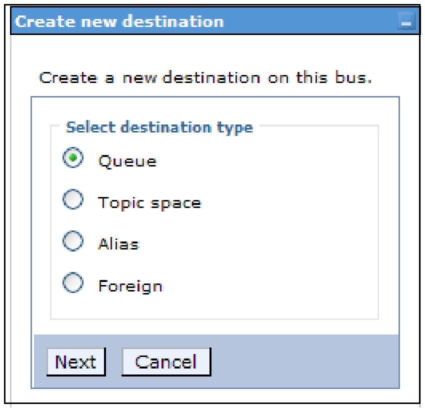
⊞ Preferences

New | Delete | Mediate | Unmediate

Select	Identifier	Bus	Type	Description	Mediation
☐	Default.Topic.Space	ws6z12Bus	Topic space		
☐	_SYSTEM.Exception.Destination.nd6z12.ws6z12-ws6z12Bus	ws6z12Bus	Queue		
Total 2					

Figure 6-17 Show bus destinations

3. Select the option **Queue** and click **Next**, as shown in Figure 6-18.


Create new destination

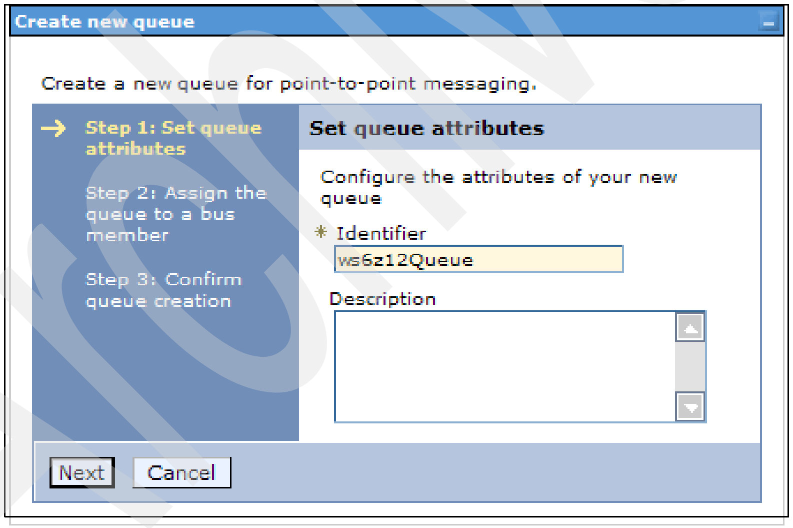
Create a new destination on this bus.

Select destination type

[✓ Queue] Topic space Alias Foreign

Figure 6-18 Selecting a destination type

4. In “Create new queue, Step 1: Set queue attributes” type `ws6z12Queue` as the identifier and click **Next**, as shown in Figure 6-19.


Create new queue

Create a new queue for point-to-point messaging.

→ Step 1: Set queue attributes
Step 2: Assign the queue to a bus member
Step 3: Confirm queue creation

Set queue attributes

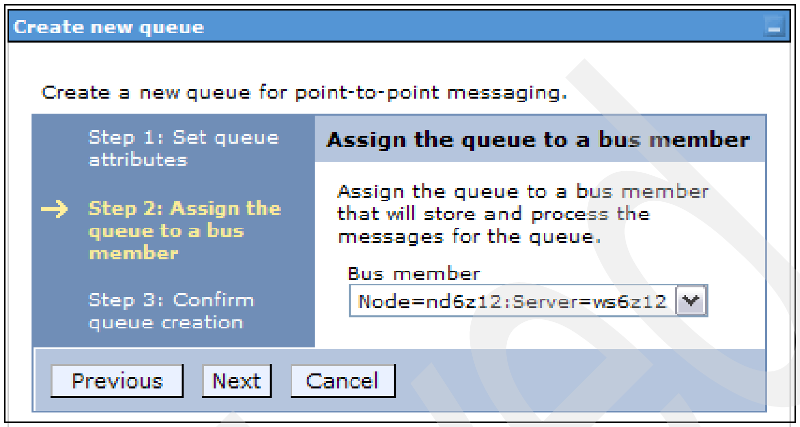
Configure the attributes of your new queue

* Identifier: ws6z12Queue

Description:

Figure 6-19 Set queue attributes

5. In “Create new queue, Step 2: Assign the queue to a bus member”, in the combo box select **Node=nd6z12:Server=ws6z12** and click **Next**. See Figure 6-20 on page 256.


Create new queue
Step 1: Set queue attributes
Step 2: Assign the queue to a bus member
Step 3: Confirm queue creation
Previous
Next
Cancel

Create a new queue for point-to-point messaging.

Assign the queue to a bus member

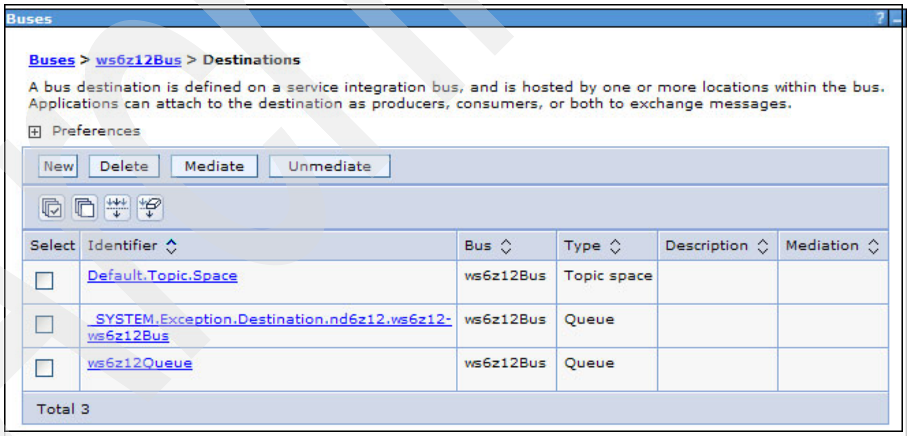
Assign the queue to a bus member that will store and process the messages for the queue.

Bus member: Node=nd6z12:Server=ws6z12

Figure 6-20 Create a new queue

On the confirmation screen, verify the information shown and click **Finish**. Save the changes.

The queue ws6z12Queue is created, as shown in Figure 6-21.



Buses > ws6z12Bus > Destinations

A bus destination is defined on a service integration bus, and is hosted by one or more locations within the bus. Applications can attach to the destination as producers, consumers, or both to exchange messages.

Select	Identifier	Bus	Type	Description	Mediation
☐	Default.Topic.Space	ws6z12Bus	Topic space		
☐	_SYSTEM.Exception.Destination.nd6z12.ws6z12-ws6z12Bus	ws6z12Bus	Queue		
☐	ws6z12Queue	ws6z12Bus	Queue		

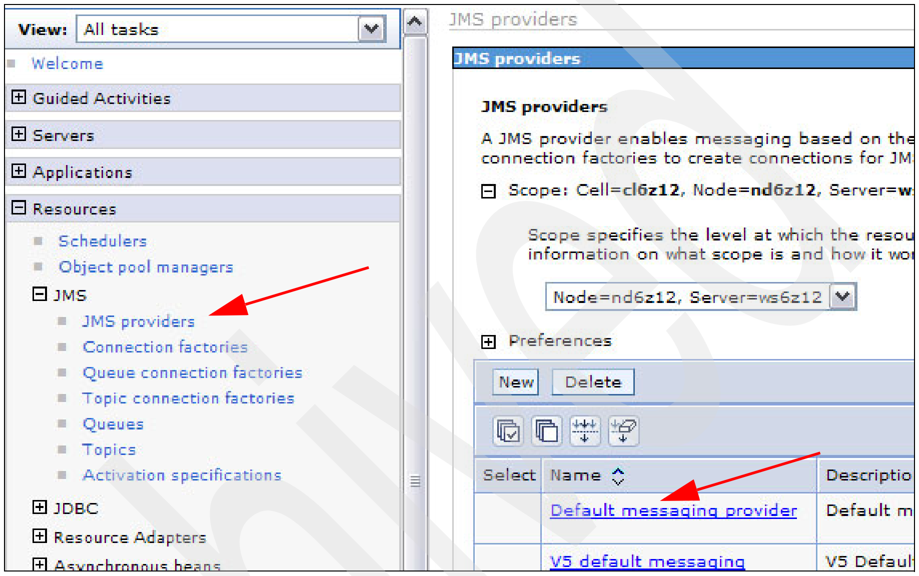
Total 3

Figure 6-21 Show bus destination

6.3.5 Defining a JMS queue to the local queue

We now need a JMS queue definition that points to the queue we just defined.

1. In the Integrated Solutions Console, expand **Resources** → **JMS** → **JMS providers** and click **Default messaging provider**, as shown in Figure 6-22.



JMS providers

A JMS provider enables messaging based on the connection factories to create connections for JM

Scope: Cell=cl6z12, Node=nd6z12, Server=w

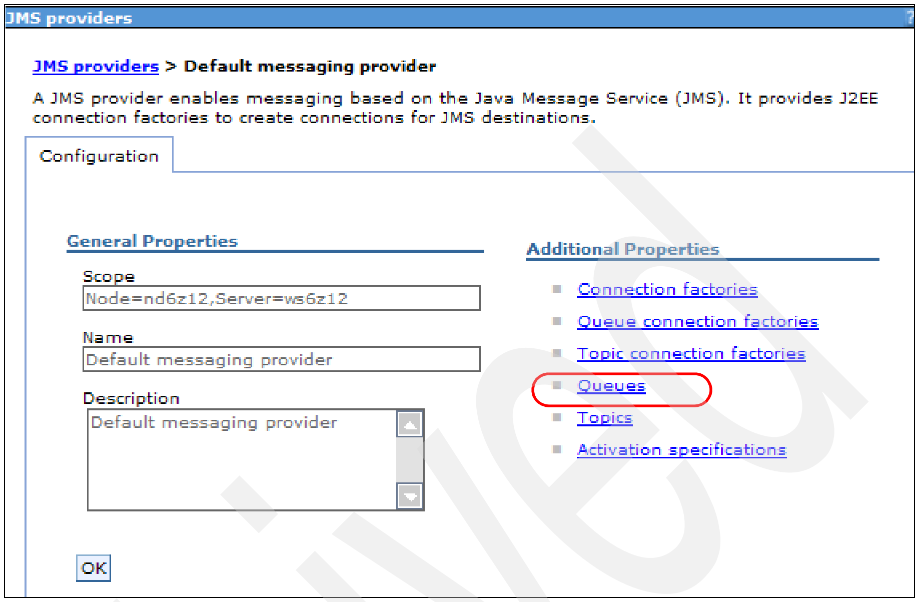
Scope specifies the level at which the resou information on what scope is and how it wo

Node=nd6z12, Server=ws6z12

Select	Name	Descriptio
	Default messaging provider	Default m
	V5 default messaging	V5 Defaul

Figure 6-22 Creating a new JMS queue - Step 1

2. In the next window, select **Queues**, as shown in Figure 6-23.


JMS providers

JMS providers > Default messaging provider

A JMS provider enables messaging based on the Java Message Service (JMS). It provides J2EE connection factories to create connections for JMS destinations.

Configuration

General Properties

Scope: Node=nd6z12,Server=ws6z12

Name: Default messaging provider

Description: Default messaging provider

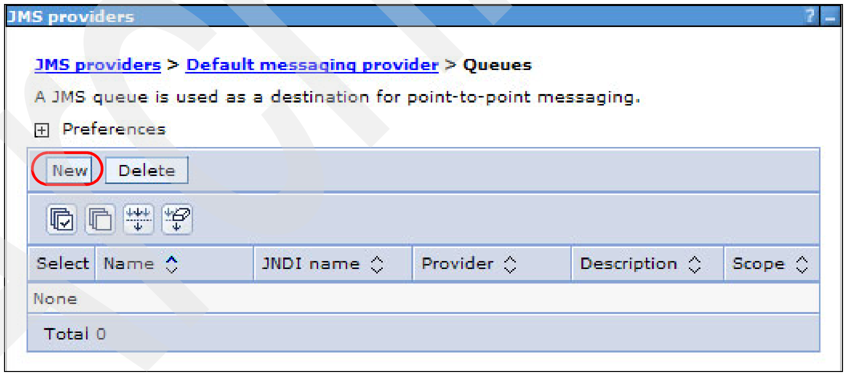
Additional Properties

- Connection factories
- Queue connection factories
- Topic connection factories
- Queues
- Topics
- Activation specifications

OK

Figure 6-23 Creating a new JMS queue - Step 2

3. On the **Queues** window, click **New**, as shown in Figure 6-24.


JMS providers

JMS providers > Default messaging provider > Queues

A JMS queue is used as a destination for point-to-point messaging.

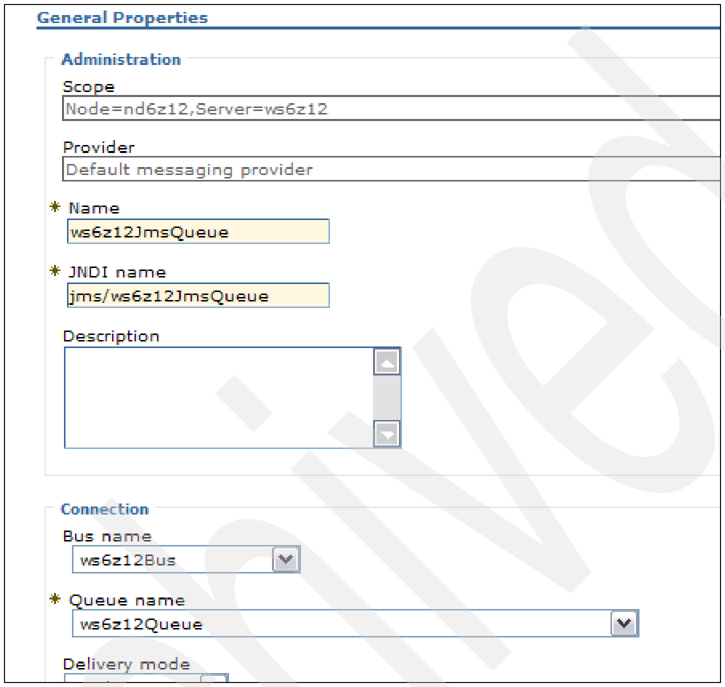
⊞ Preferences

New Delete

Select	Name	JNDI name	Provider	Description	Scope
None					
Total 0					

Figure 6-24 Creating a new JMS queue - Step 3

4. On the General Properties window, type `ws6z12JmsQueue` in the Name field, **`jms/ws6z12JmsQueue`** in the JNDI name field, select **ws6z12Bus** as a Bus name, select **ws6z12Queue** as a Queue name, and click **OK**. See Figure 6-25.


General Properties

Administration

Scope
Node=nd6z12,Server=ws6z12

Provider
Default messaging provider

* Name
ws6z12JmsQueue

* JNDI name
jms/ws6z12JmsQueue

Description

Connection

Bus name
ws6z12Bus

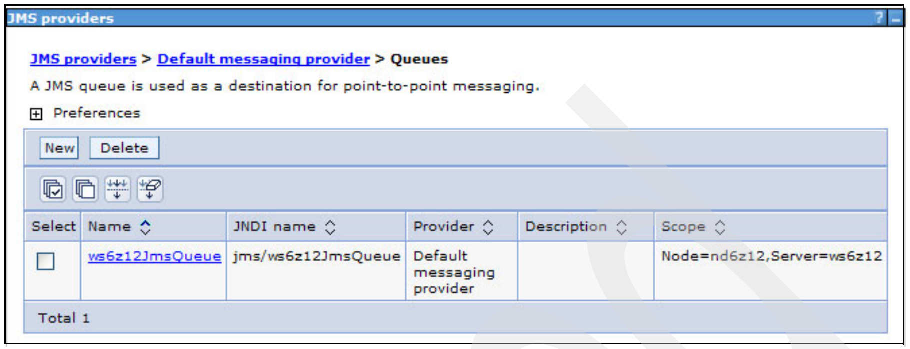
* Queue name
ws6z12Queue

Delivery mode

Figure 6-25 Creating a new JMS queue - Step 4

5. Verify the information and click **Finish** at the confirmation screen.

6. **Save** the changes and the JMS queue ws6z12JmsQueue will be created. See Figure 6-26.


JMS providers

JMS providers > Default messaging provider > Queues

A JMS queue is used as a destination for point-to-point messaging.

⊞ Preferences

Select	Name	JNDI name	Provider	Description	Scope
☐	ws6z12JmsQueue	jms/ws6z12JmsQueue	Default messaging provider		Node=nd6z12,Server=ws6z12

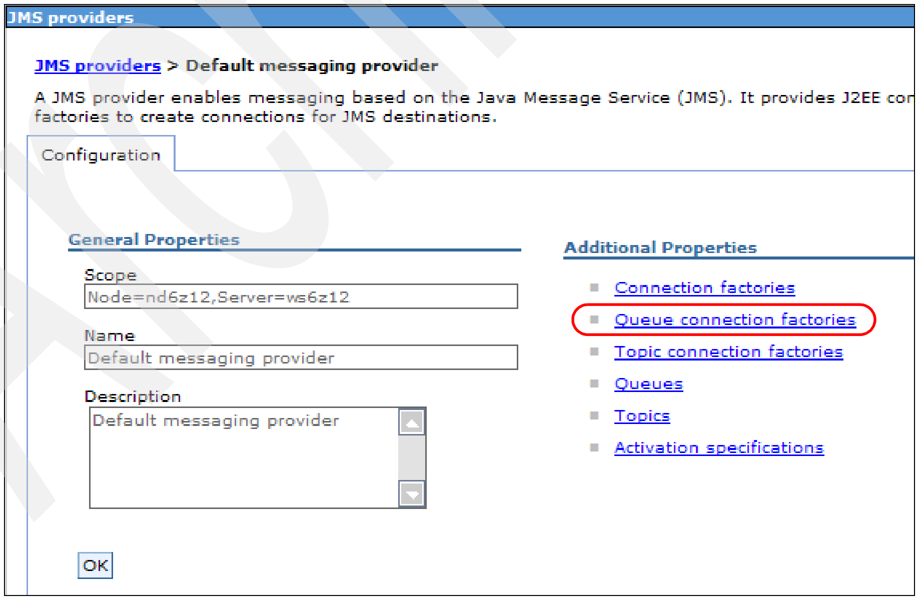
Total 1

Figure 6-26 JMS queue created

6.3.6 Defining the JMS queue connection factory

We need a JMS queue connection factory definition for the applications to get connections to the JMS provider. Proceed as follows:

1. Expand **Resources** → **JMS** → **JMS providers** → **Default messaging provider** and select **Queue connection factories**. See Figure 6-27.


JMS providers

JMS providers > Default messaging provider

A JMS provider enables messaging based on the Java Message Service (JMS). It provides J2EE cor
factories to create connections for JMS destinations.

Configuration

General Properties

Scope: Node=nd6z12,Server=ws6z12

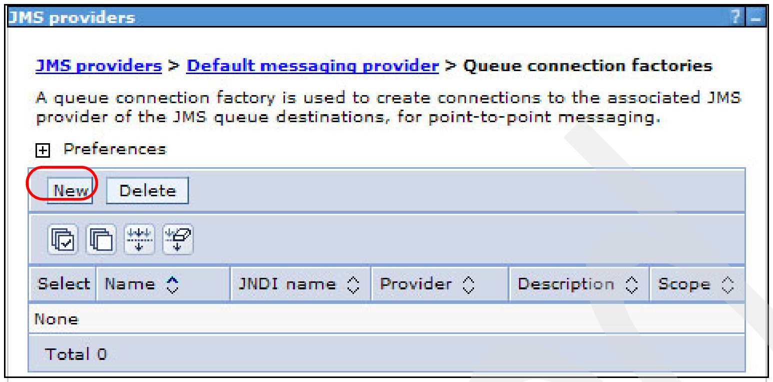
Name: Default messaging provider

Description: Default messaging provider

Additional Properties
- Connection factories
- Queue connection factories
- Topic connection factories
- Queues
- Topics
- Activation specifications

Figure 6-27 Select Queue connection factories

2. Click **New**, as shown in Figure 6-28.


JMS providers

JMS providers > Default messaging provider > Queue connection factories

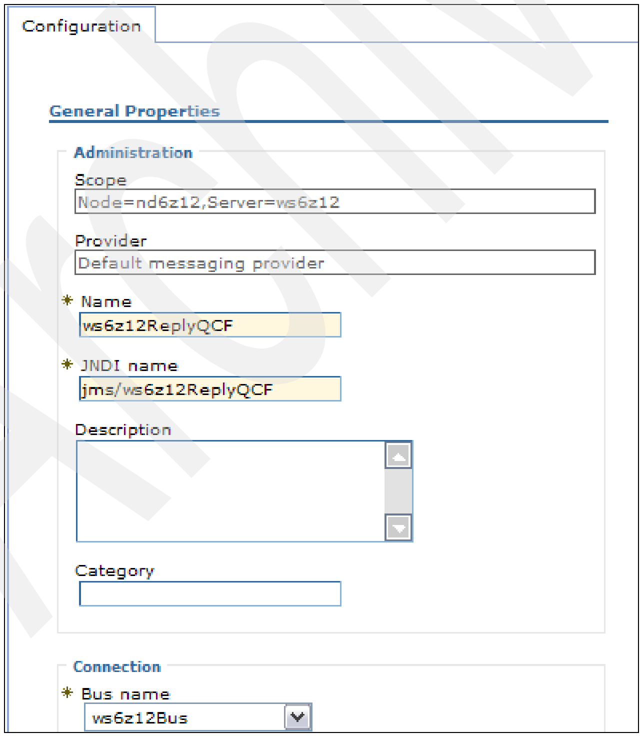
A queue connection factory is used to create connections to the associated JMS provider of the JMS queue destinations, for point-to-point messaging.

⊞ Preferences

Select	Name	JNDI name	Provider	Description	Scope
None					
Total 0					

Figure 6-28 Creating a new queue connection factory

3. In the General properties window, type `ws6z12JReplyQCF` in the Name field, `jms/ws6z12JReplyCF` in the JNDI name field, select **ws6z12Bus** as a Bus name, and click **OK**. See Figure 6-29.


Configuration

General Properties

Administration

Scope: Node=nd6z12,Server=ws6z12

Provider: Default messaging provider

* Name: ws6z12ReplyQCF

* JNDI name: jms/ws6z12ReplyQCF

Description:

Category:

Connection

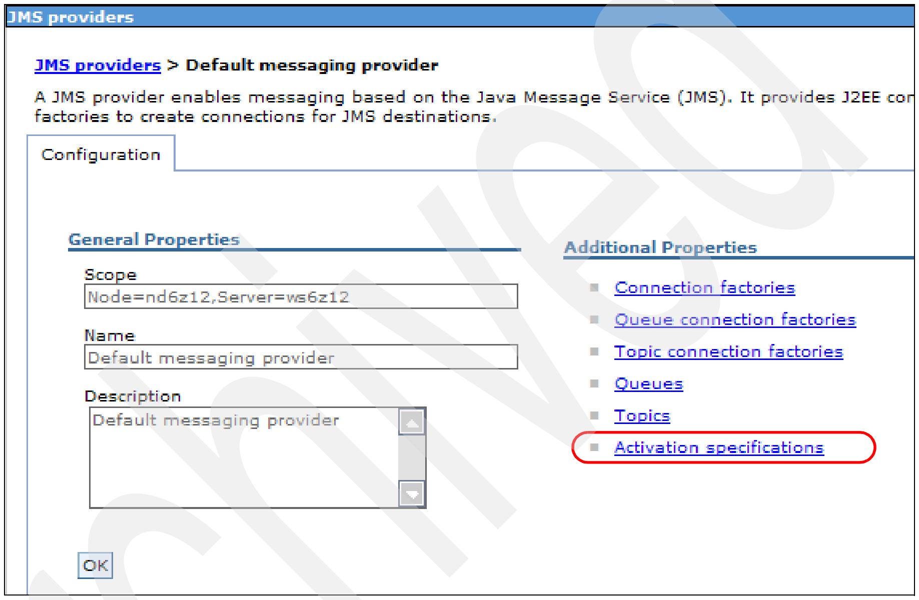
* Bus name: ws6z12Bus

Figure 6-29 Queue connection factory - General Properties

6.3.7 Defining a JMS activation specification

A *JMS activation specification* is associated with one or more Message Driven Beans (MDB) and provides the configuration necessary for them to receive messages. Proceed as follows:

1. Expand **Resources** → **JMS** → **JMS providers** → **Default messaging provider** and select **Activation specifications**. See Figure 6-30 on page 262.


JMS providers

JMS providers > Default messaging provider

A JMS provider enables messaging based on the Java Message Service (JMS). It provides J2EE cor
factories to create connections for JMS destinations.

Configuration

General Properties

Scope
Node=nd6z12,Server=ws6z12

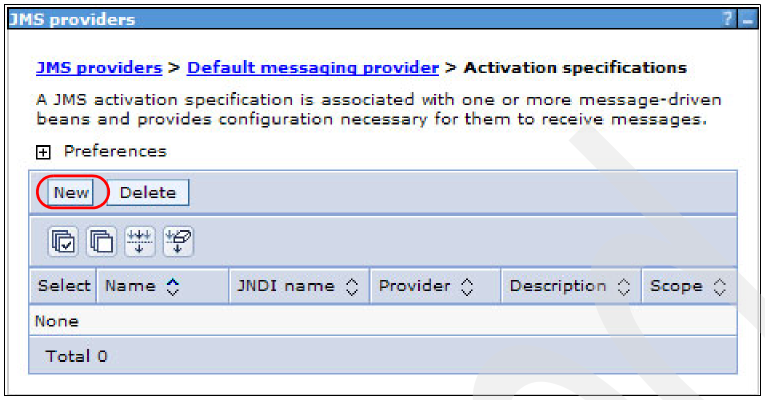
Name
Default messaging provider

Description
Default messaging provider

Additional Properties
- Connection factories
- Queue connection factories
- Topic connection factories
- Queues
- Topics
- Activation specifications

Figure 6-30 Select Activation specifications

2. In the next window, click **New**, as shown in Figure 6-31.


JMS providers

JMS providers > Default messaging provider > Activation specifications

A JMS activation specification is associated with one or more message-driven beans and provides configuration necessary for them to receive messages.

⊞ Preferences

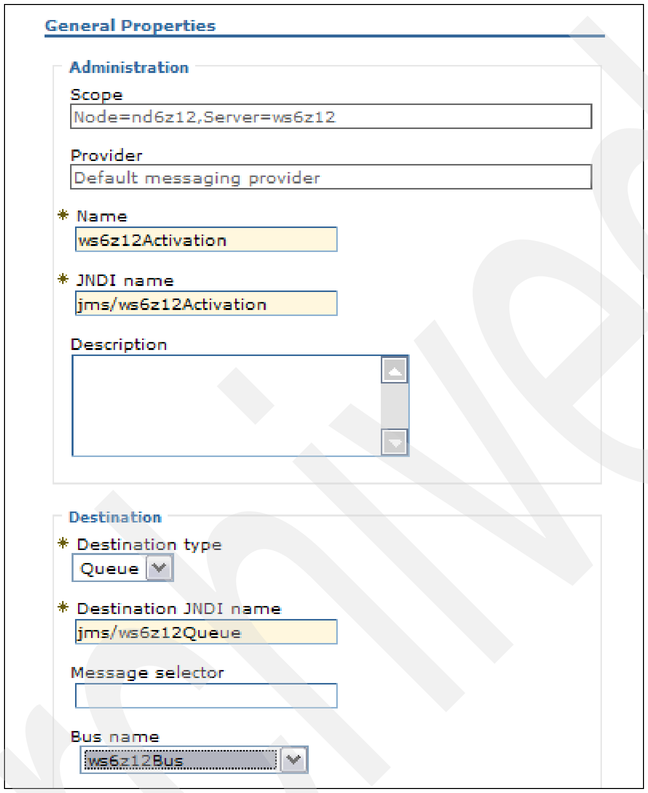
New Delete

Select	Name	JNDI name	Provider	Description	Scope
None					

Total 0

Figure 6-31 Creating a new activation specification

3. In the General Properties window, type `ws6z12Activation` in the Name field, `jms/ws6z12JActivation` in the JNDI name field, type `ws6z12Queue` in the Destination JNDI name field, select **ws6z12Bus** as a Bus name, and click **OK**. See Figure 6-32.



Scope: Node=nd6z12,Server=ws6z12

Provider: Default messaging provider

* Name: ws6z12Activation

* JNDI name: jms/ws6z12Activation

Description:

* Destination type: Queue

* Destination JNDI name: jms/ws6z12Queue

Message selector:

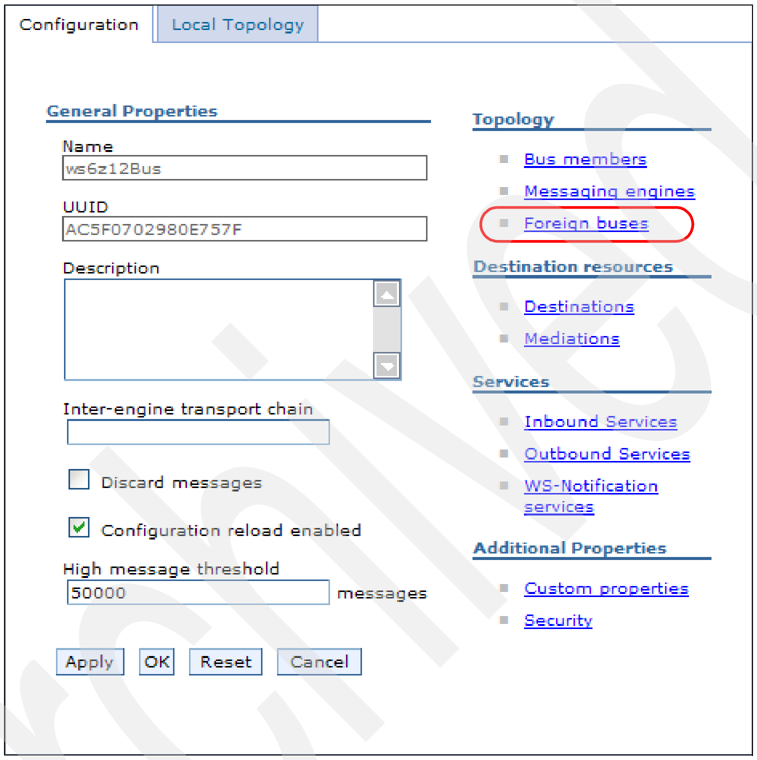
Bus name: ws6z12Bus

Figure 6-32 Creating a new JMS activation specification - General Properties

6.3.8 Defining a foreign bus

The WebSphere MQ Queue Manager will need to be defined as a *foreign bus*.

1. Expand **Service Integration** → **Buses**, and select **Foreign buses**, as shown in Figure 6-33.



Name
ws6z12Bus

UUID
AC5F0702980E757F

Description

Inter-engine transport chain

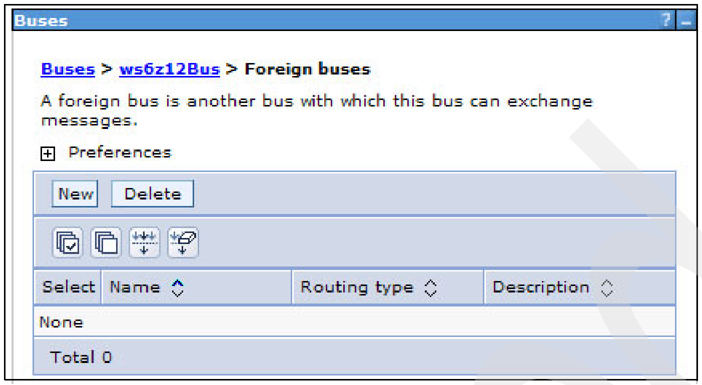
☐ Discard messages

☑ Configuration reload enabled

High message threshold
50000 messages

Figure 6-33 Defining a foreign bus

2. In the Foreign buses window, click **New**, as shown in Figure 6-34.


Buses

Buses > ws6z12Bus > Foreign buses

A foreign bus is another bus with which this bus can exchange messages.

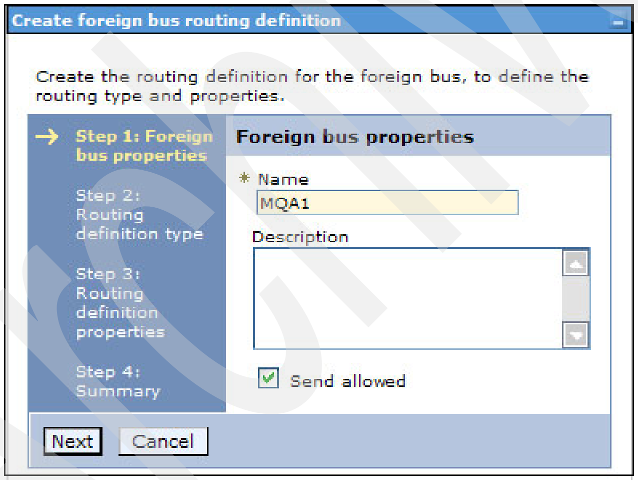
⊞ Preferences

New Delete

Select	Name	Routing type	Description
None			
Total 0			

Figure 6-34 Creating a new foreign bus

3. In the “Create foreign bus routing definition” window, in “Step 1: Foreign bus properties”, type `MQA1` as a bus name and click **Next**. See Figure 6-35.


Create foreign bus routing definition

Create the routing definition for the foreign bus, to define the routing type and properties.

→ Step 1: Foreign bus properties

Step 2: Routing definition type

Step 3: Routing definition properties

Step 4: Summary

Foreign bus properties

* Name
MQA1

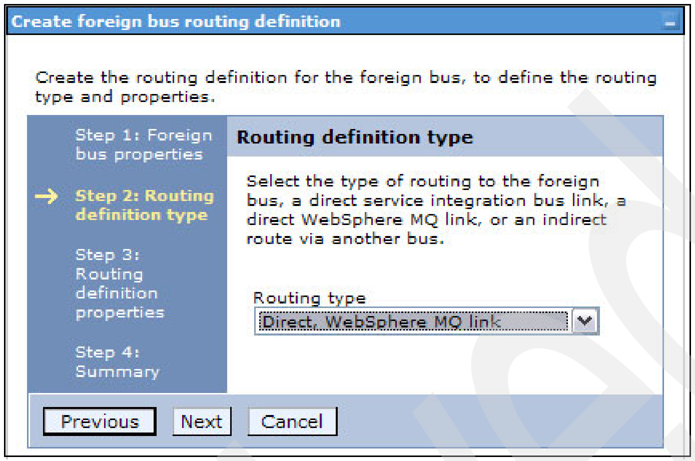
Description

✓ Send allowed

Next Cancel

Figure 6-35 Defining a new foreign bus - properties

4. In “Step 2: Routing definition type”, select **Direct, WebSphere MQ Link** as a Routing type and click **Next**, as shown in Figure 6-36.


Create foreign bus routing definition

Create the routing definition for the foreign bus, to define the routing type and properties.

Step 1: Foreign bus properties
→ Step 2: Routing definition type
Step 3: Routing definition properties
Step 4: Summary

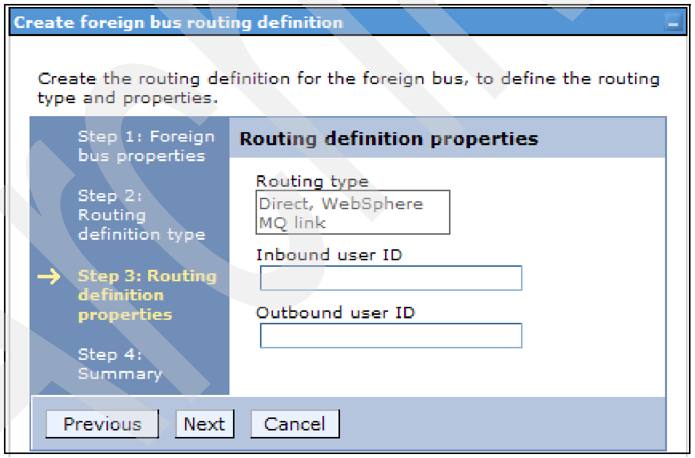
Routing definition type

Select the type of routing to the foreign bus, a direct service integration bus link, a direct WebSphere MQ link, or an indirect route via another bus.

Routing type: Direct, WebSphere MQ link

Figure 6-36 Defining a new foreign bus - Routing type

5. In “Step 3: Routing definition properties”, leave the fields blank and click **Next**. See Figure 6-37.


Create foreign bus routing definition

Create the routing definition for the foreign bus, to define the routing type and properties.

Step 1: Foreign bus properties
Step 2: Routing definition type
→ Step 3: Routing definition properties
Step 4: Summary

Routing definition properties

Routing type: Direct, WebSphere MQ link

Inbound user ID:

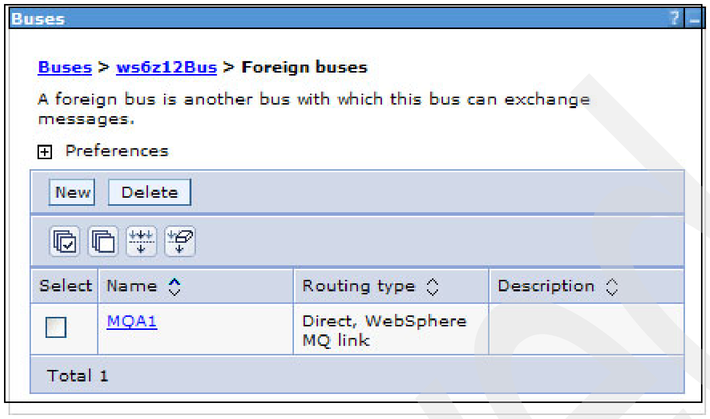
Outbound user ID:

Figure 6-37 Defining a new foreign bus - routing definition properties

6. On the confirmation window, verify the information and click **Finish**.

7. **Save** the changes.

   The Foreign bus MQA1 is created now, as shown in Figure 6-38.



Buses

Buses > ws6z12Bus > Foreign buses

A foreign bus is another bus with which this bus can exchange messages.

⊞ Preferences

New Delete

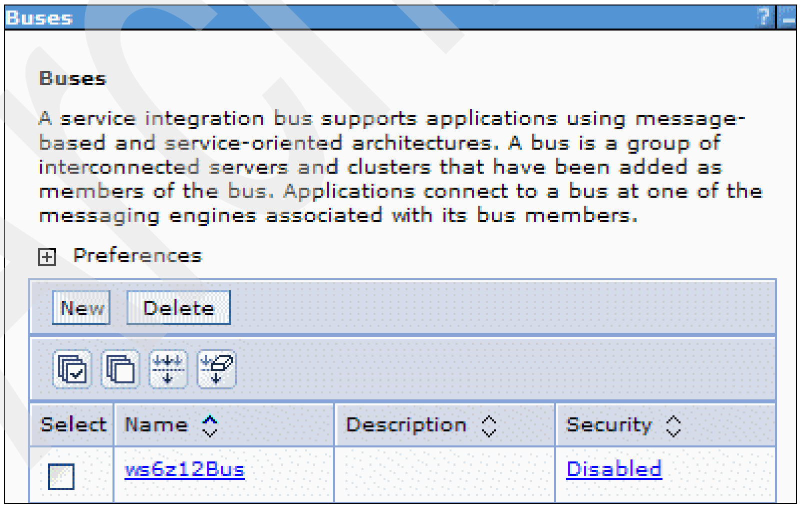
Select	Name	Routing type	Description
☐	MQA1	Direct, WebSphere MQ link	
Total 1			

Figure 6-38 Foreign bus created

6.3.9 Define a WebSphere MQ Link

We need to configure a WebSphere MQ Link to define how the WebSphere SIB connects to the external WebSphere MQ Queue Manager.

1. Expand **Service Integration** → **Buses**, and click **ws6z12Bus**.



Buses

Buses

A service integration bus supports applications using message-based and service-oriented architectures. A bus is a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

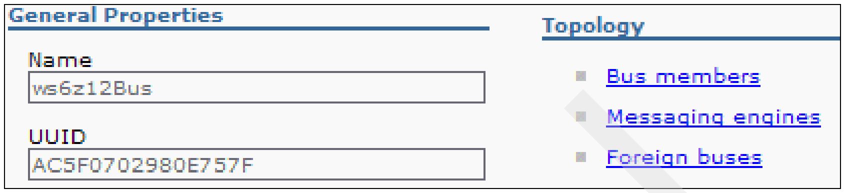
⊞ Preferences

New Delete

Select	Name	Description	Security
☐	ws6z12Bus		Disabled

Figure 6-39 Buses window

2. In the next window, click **Messaging engines** under Topology (on the right), as shown in Figure 6-40.


General Properties

Name

ws6z12Bus

UUID

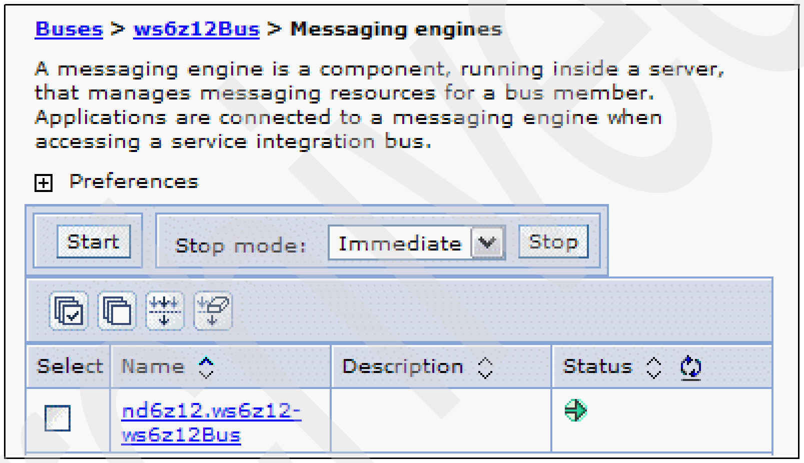
AC5F0702980E757F

Topology

- Bus members
- Messaging engines
- Foreign buses

Figure 6-40 Message engine

3. In the next window, click on the Messaging Engine **nd6z12.ws6z12-ws6z12Bus**, as shown in Figure 6-41.


Buses > ws6z12Bus > Messaging engines

A messaging engine is a component, running inside a server, that manages messaging resources for a bus member. Applications are connected to a messaging engine when accessing a service integration bus.

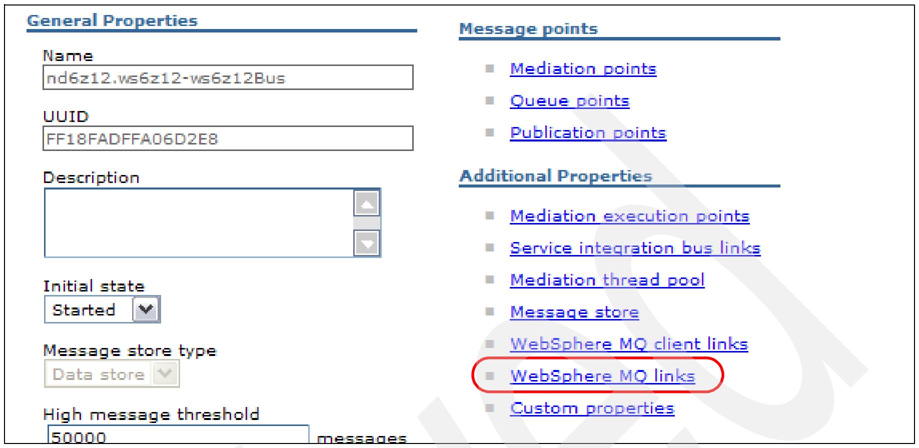
⊞ Preferences

Start | Stop mode: Immediate | Stop

Select	Name	Description	Status
☐	nd6z12.ws6z12-ws6z12Bus		➡

Figure 6-41 Selecting Message engine

4. Next, click **WebSphere MQ Links** in the Additional Properties section, as shown in Figure 6-42.


General Properties

Name: nd6z12.ws6z12-ws6z12Bus

UUID: FF18FADFFA06D2E8

Description:

Initial state: Started

Message store type: Data store

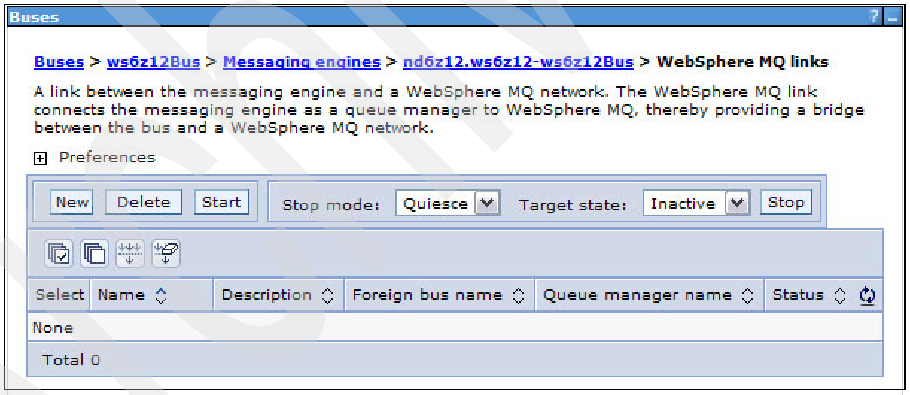
High message threshold: 50000 messages

Message points
- Mediation points
- Queue points
- Publication points

Additional Properties
- Mediation execution points
- Service integration bus links
- Mediation thread pool
- Message store
- WebSphere MQ client links
- WebSphere MQ links
- Custom properties

Figure 6-42 Defining a WebSphere MQ Link

5. On the WebSphere MQ Links window, click **New**. See Figure 6-43.


Buses

Buses > ws6z12Bus > Messaging engines > nd6z12.ws6z12-ws6z12Bus > **WebSphere MQ links**

A link between the messaging engine and a WebSphere MQ network. The WebSphere MQ link connects the messaging engine as a queue manager to WebSphere MQ, thereby providing a bridge between the bus and a WebSphere MQ network.

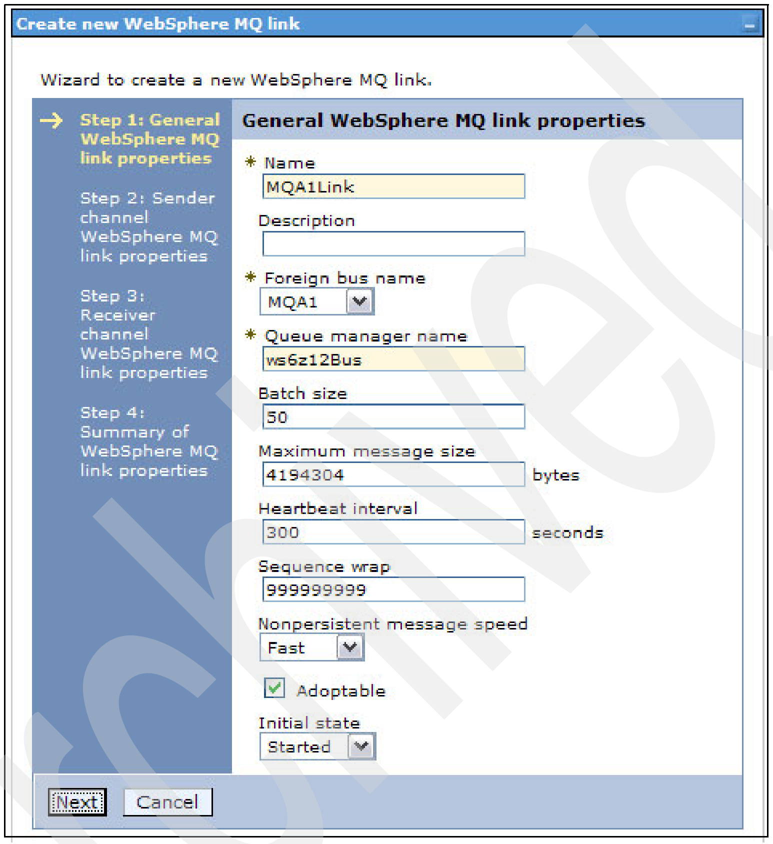
⊞ Preferences

New | Delete | Start | Stop mode: Quiesce | Target state: Inactive | Stop

Select	Name	Description	Foreign bus name	Queue manager name	Status
None					
Total 0					

Figure 6-43 Creating a new WebSphere MQ Link

6. On the Create new WebSphere MQ Link window, in “Step 1: General WebSphere MQ link properties” we typed `MQA1Link` in the Name field, selected **MQA1** as the foreign bus, typed `ws6z12Bus` as a Queue manager Name, and clicked **Next**. See Figure 6-44.



Create new WebSphere MQ link

Wizard to create a new WebSphere MQ link.

→ **Step 1: General WebSphere MQ link properties**

Step 2: Sender channel WebSphere MQ link properties

Step 3: Receiver channel WebSphere MQ link properties

Step 4: Summary of WebSphere MQ link properties

General WebSphere MQ link properties

* Name: MQA1Link

Description:

* Foreign bus name: MQA1

* Queue manager name: ws6z12Bus

Batch size: 50

Maximum message size: 4194304 bytes

Heartbeat interval: 300 seconds

Sequence wrap: 999999999

Nonpersistent message speed: Fast

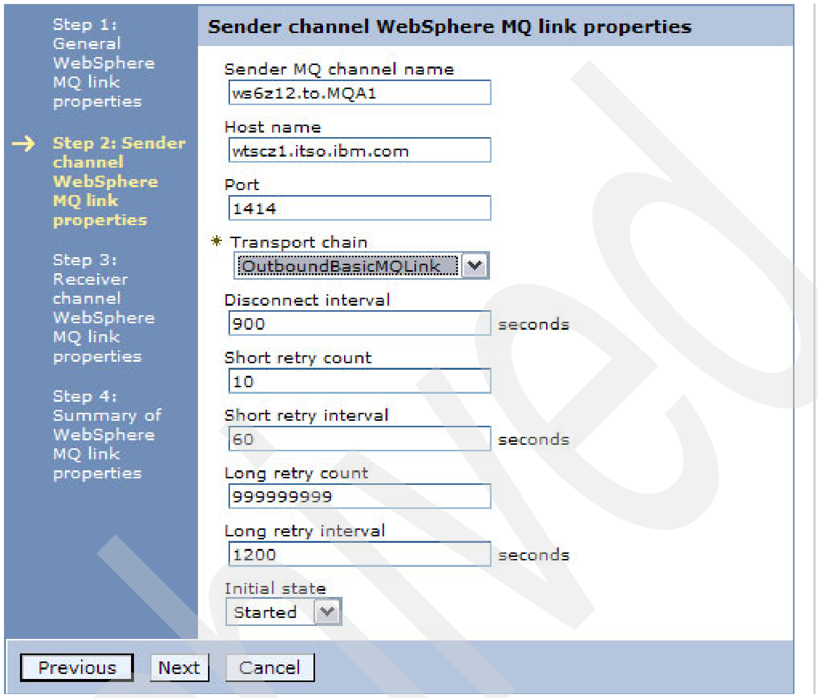
✓ Adoptable

Initial state: Started

Figure 6-44 Defining a new WebSphere MQ Link - General properties

Note: Note that the Queue manager name field is set to ws6z12Bus, the name of the bus sending the message; it is not the name of the queue manger the message is being sent to. This value becomes the reply to the queue manager name set in the message header sent in the message to the remote queue manager. This allows the remote queue manager, in this case MQA1, to know where to send the reply message.

7. In the “Step 2: Sender channel WebSphere MQ Link properties” window, we set the Sender MQ channel name to `ws6z12.to.MQA1`, the Host name to `wtscz1.itso.ibm.com`, the port to `1414`, and clicked **Next**. See Figure 6-45.



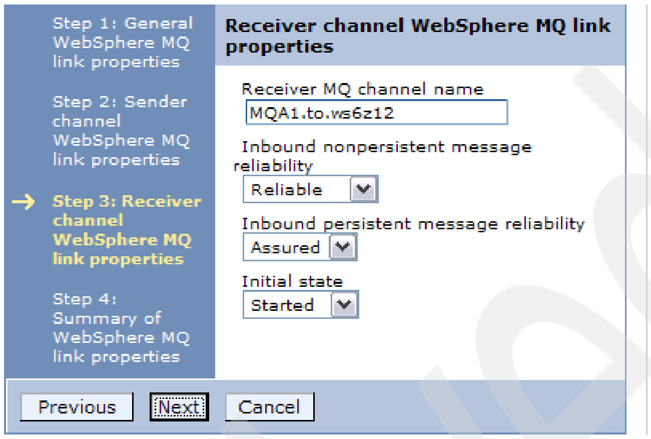
Step 1: General WebSphere MQ link properties
→ Step 2: Sender channel WebSphere MQ link properties
Step 3: Receiver channel WebSphere MQ link properties
Step 4: Summary of WebSphere MQ link properties
Previous
Next
Cancel

Sender channel WebSphere MQ link properties

Sender MQ channel name: ws6z12.to.MQA1
Host name: wtscz1.itso.ibm.com
Port: 1414
* Transport chain: OutboundBasicMQLink
Disconnect interval: 900 seconds
Short retry count: 10
Short retry interval: 60 seconds
Long retry count: 999999999
Long retry interval: 1200 seconds
Initial state: Started

Figure 6-45 Defining a new WebSphere MQ Link - Sender Channel

8. In “Step 3: Receiver Channel WebSphere MQ Link properties”, we set the Receiver MQ Channel name to `MQA1.to.ws6z12` and clicked **Next**. See Figure 6-46.


Step 1: General WebSphere MQ link properties
Step 2: Sender channel WebSphere MQ link properties
Step 3: Receiver channel WebSphere MQ link properties
Step 4: Summary of WebSphere MQ link properties

Receiver channel WebSphere MQ link properties

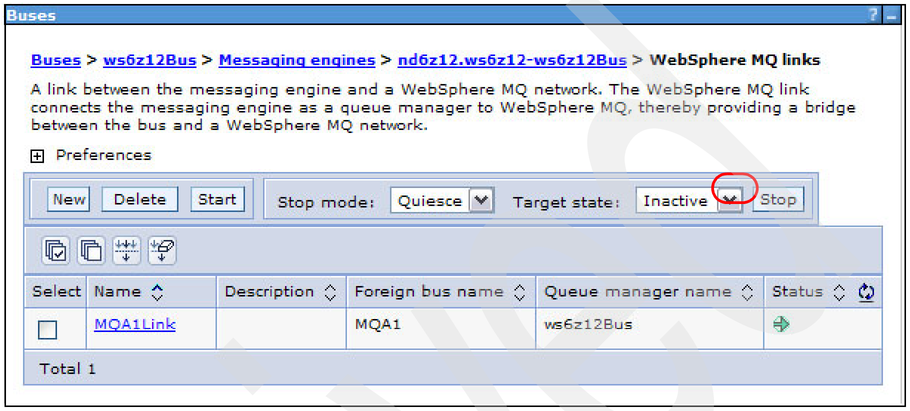
Receiver MQ channel name: MQA1.to.ws6z12
Inbound nonpersistent message reliability: Reliable
Inbound persistent message reliability: Assured
Initial state: Started

Figure 6-46 Defining a new WebSphere MQ Link - Receiver channel

9. Verify the information and click **Finish.**
10. **Save** the changes and the WebSphere MQ Link MQA1Link is created.
11. Restart the WebSphere Application Server to activate the configuration.

6.3.10 Verifying the channels

1. Navigate to **Buses** → **ws6z12Bus** → **Messaging engines** → **nd6z12.ws6r12-ws6z12Bus** → **WebSphere MQ links,** check whether the status of the MQA1Link is running, and click **MQA1Link**. See Figure 6-47.



Buses

Buses > ws6z12Bus > Messaging engines > nd6z12.ws6z12-ws6z12Bus > WebSphere MQ links

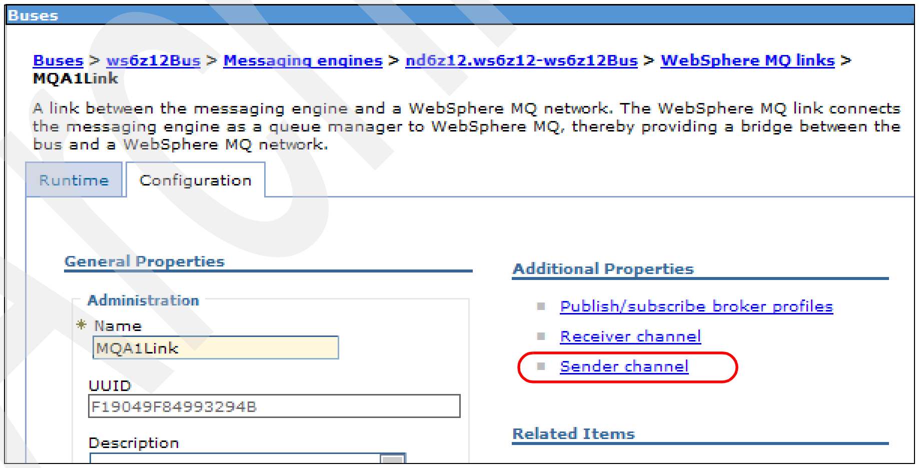
A link between the messaging engine and a WebSphere MQ network. The WebSphere MQ link connects the messaging engine as a queue manager to WebSphere MQ, thereby providing a bridge between the bus and a WebSphere MQ network.

Stop mode: Quiesce   Target state: Inactive

Select	Name	Description	Foreign bus name	Queue manager name	Status
☐	MQA1Link		MQA1	ws6z12Bus	➔
Total 1					

Figure 6-47 Show WebSphere MQ Link - Running

2. In the MQA1Link window, click **Sender Channel** in the Additional Properties section, as shown in Figure 6-48.



Buses

Buses > ws6z12Bus > Messaging engines > nd6z12.ws6z12-ws6z12Bus > WebSphere MQ links > MQA1Link

A link between the messaging engine and a WebSphere MQ network. The WebSphere MQ link connects the messaging engine as a queue manager to WebSphere MQ, thereby providing a bridge between the bus and a WebSphere MQ network.

General Properties

Administration

* Name: MQA1Link

UUID: F19049F84993294B

Description

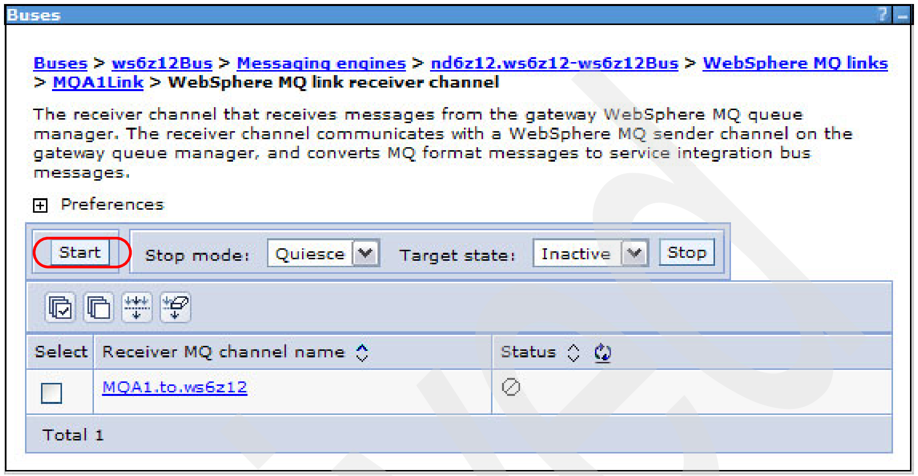
Additional Properties

- Publish/subscribe broker profiles
- Receiver channel
- Sender channel

Related Items

Figure 6-48 MQA1Link Properties

3. In the WebSphere MQ link receiver channel window we can check the channel status and start it, if necessary. As we can see in Figure 6-49, the channel is not running. You can start the channel by clicking **Start**.



Buses

Buses > ws6z12Bus > Messaging engines > nd6z12.ws6z12-ws6z12Bus > WebSphere MQ links > MQA1Link > **WebSphere MQ link receiver channel**

The receiver channel that receives messages from the gateway WebSphere MQ queue manager. The receiver channel communicates with a WebSphere MQ sender channel on the gateway queue manager, and converts MQ format messages to service integration bus messages.

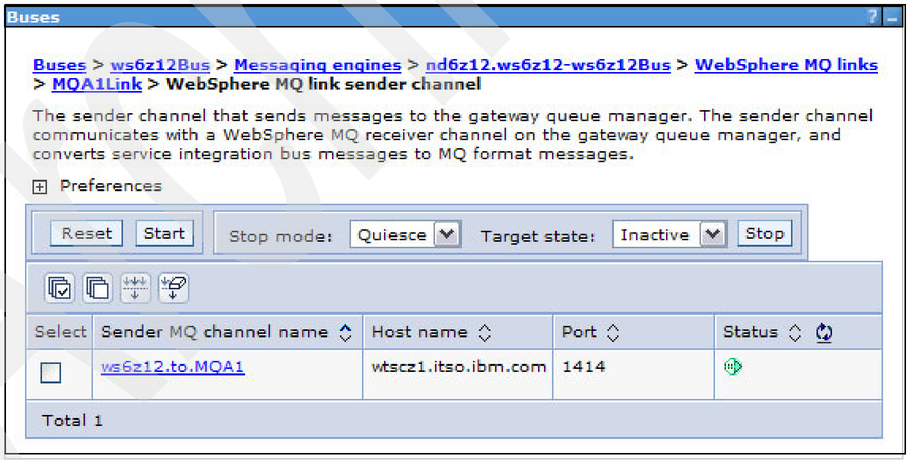
⊞ Preferences

Start | Stop mode: Quiesce | Target state: Inactive | Stop

Select	Receiver MQ channel name	Status
☐	MQA1.to.ws6z12	⊘
Total 1		

Figure 6-49 MQA1Link - Receiver channel not started

4. Go back to the MQA1Link window and select **Sender channel**. Notice that the sender channel ws6z12.to.MQA1 is running.



Buses

Buses > ws6z12Bus > Messaging engines > nd6z12.ws6z12-ws6z12Bus > WebSphere MQ links > MQA1Link > **WebSphere MQ link sender channel**

The sender channel that sends messages to the gateway queue manager. The sender channel communicates with a WebSphere MQ receiver channel on the gateway queue manager, and converts service integration bus messages to MQ format messages.

⊞ Preferences

Reset | Start | Stop mode: Quiesce | Target state: Inactive | Stop

Select	Sender MQ channel name	Host name	Port	Status
☐	ws6z12.to.MQA1	wtscz1.itso.ibm.com	1414	⇨
Total 1				

Figure 6-50 MQA1Link - Sender channel started

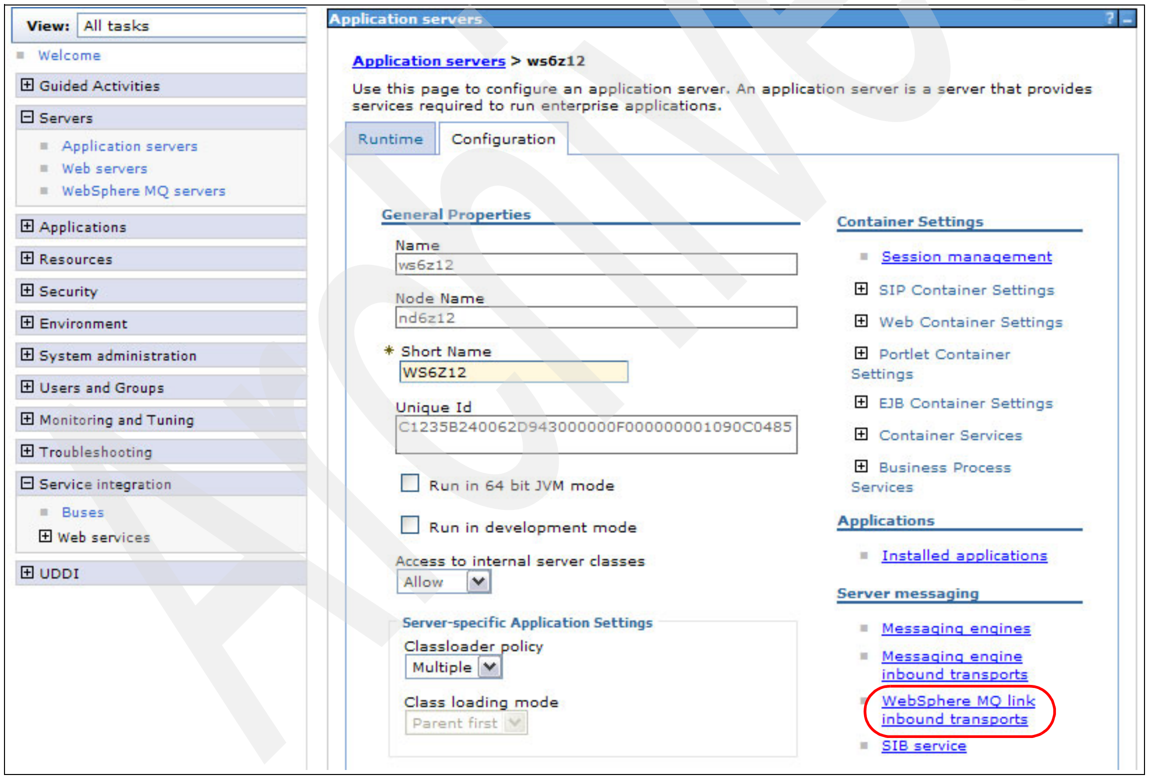
6.4 WebSphere MQ configuration

Now that almost everything is set up in WebSphere Application Server, we move over to the WebSphere MQ side and make the necessary definitions. We need to define the sender and receiver channels in WebSphere MQ to be able to communicate between WebSphere Application Server and WebSphere MQ.

6.4.1 Checking the port numbers in WebSphere Application Server

First, we need to check which TCP/IP port is used in WebSphere to support connections on the WebSphere side.

1. In the WebSphere Integrated Solutions Console, select **Server** → **Application Server** → **ws6z12,** then click **WebSphere MQ link inbound transports**. See Figure 6-51.



Application servers > ws6z12

Use this page to configure an application server. An application server is a server that provides services required to run enterprise applications.

General Properties

Name: ws6z12

Node Name: nd6z12

* Short Name: WS6Z12

Unique Id: C1235B240062D943000000F000000001090C0485

☐ Run in 64 bit JVM mode

☐ Run in development mode

Access to internal server classes: Allow

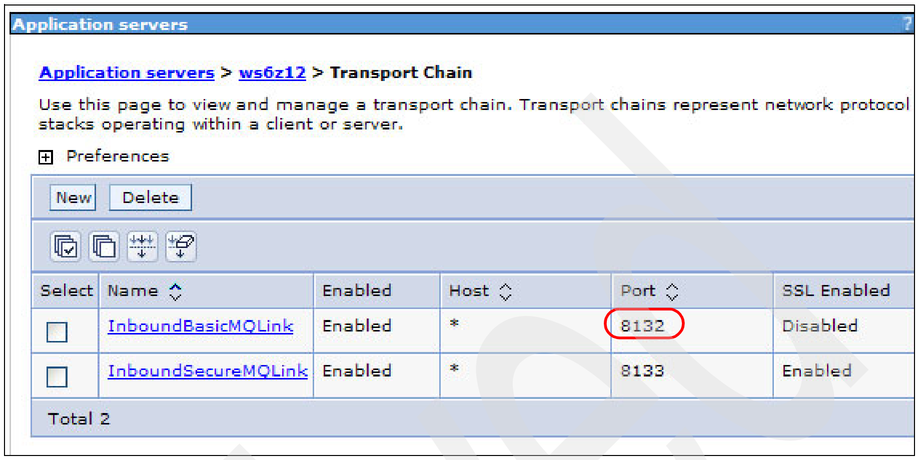
Server-specific Application Settings

Classloader policy: Multiple

Class loading mode: Parent first

Figure 6-51 Selecting WebSphere MQLink inbound transports

2. Notice the InboundBasicMQLink TCP/IP port number, as shown in Figure 6-52.



Application servers > ws6z12 > Transport Chain

Use this page to view and manage a transport chain. Transport chains represent network protocol stacks operating within a client or server.

Select	Name	Enabled	Host	Port	SSL Enabled
☐	InboundBasicMQLink	Enabled	*	8132	Disabled
☐	InboundSecureMQLink	Enabled	*	8133	Enabled

Total 2

Figure 6-52 Checking the port number of the WebSphere MQLink inbound transports

6.4.2 Defining the channels in WebSphere MQ

Using the information from the previous sections, we can run a job similar to the one shown in Example 6-2 to actually define the channels to WebSphere MQ. Besides the channels we are also defining the transmission queue. You can see in this example that the channel names and the WebSphere Application Server host name and port number correspond with definitions we used earlier when setting up WebSphere Application Server.

Example 6-2 MQ channel configuration

```
//ITSO011 JOB 'ITSO01-MQ',MSGLEVEL=(1,1),MSGCLASS=H,
//    REGION=0M,CLASS=A,NOTIFY=&SYSUID
//*
//DISP EXEC PGM=CSQUTIL,PARM='MQA1'
//STEPLIB DD DISP=SHR,DSN=MQ600.SCSQANLE
//        DD DISP=SHR,DSN=MQ600.SCSQAUTH
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
 COMMAND
//CSQUCMD DD *
define qlocal('ws6z12') +
       usage(xmitq) +
       replace
```

```
define channel('MQA1.to.ws6z12') +
       chltype(sdr) +
       conname('wtscz1.itso.ibm.com(8132)') +
       trptype(TCP) +
       xmitq('ws6z12') +
       replace

define channel('ws6z12.to.MQA1') +
       chltype(rcvr) +
       trptype(TCP) +
       replace
```

Make sure that this job finished with `return code=0` and `0 failed`, as shown in Example 6-3.

Example 6-3 Job output of defining channels to WebSphere MQ

```
CSQU057I  3 commands read
 CSQU058I  3 commands issued and responses received, 0 failed
 CSQU143I  1 COMMAND statements attempted
 CSQU144I  1 COMMAND statements executed successfully
 CSQU148I CSQUTIL Utility completed, return code=0
```

Now that we can start the channels on both sides (WebSphere Application Server and WebSphere MQ). Proceed as follows:

1. First start the sender channel on the WebSphere MQ side. To do this, open the MQ administration panel on z/OS. Type number 1 as the Action, select **CHANNEL** as the Object Type, **MQA1.to.ws6z12** as the Name, and press Enter. This will list the available defined channels with this name. See Example 6-4.

Example 6-4 Selecting channels to browse

```
IBM WebSphere MQ for z/OS - Main Menu

 Complete fields. Then press Enter.

 Action  . . . . . . . . . . 1     0. List with filter   4. Manage
                                   1. List or Display    5. Perform
                                   2. Define like        6. Start
                                   3. Alter              7. Stop
 Object type . . . . . . . . CHANNEL      +
 Name  . . . . . . . . . . . MQA1.to.ws6z12
 Disposition . . . . . . . . A  Q=Qmgr, C=Copy, P=Private, G=Group,
                                S=Shared, A=All
 Connect name  . . . . . . . MQA1  - local queue manager or group
 Target queue manager  . . . MQA1
            - connected or remote queue manager for command input
 Action queue manager  . . . MQA1  - command scope in group
 Response wait time  . . . . 30    5 - 999 seconds

 (C) Copyright IBM Corporation 1993,2005. All rights reserved.

 Command ===>
  F1=Help      F2=Split     F3=Exit      F4=Prompt    F9=SwapNext F10=Messages
 F12=Cancel
```

2. In the List Channels screen type the number 6 besides the sender channel name and press Enter, as shown in Example 6-5.

Example 6-5 Selecting the sender channel

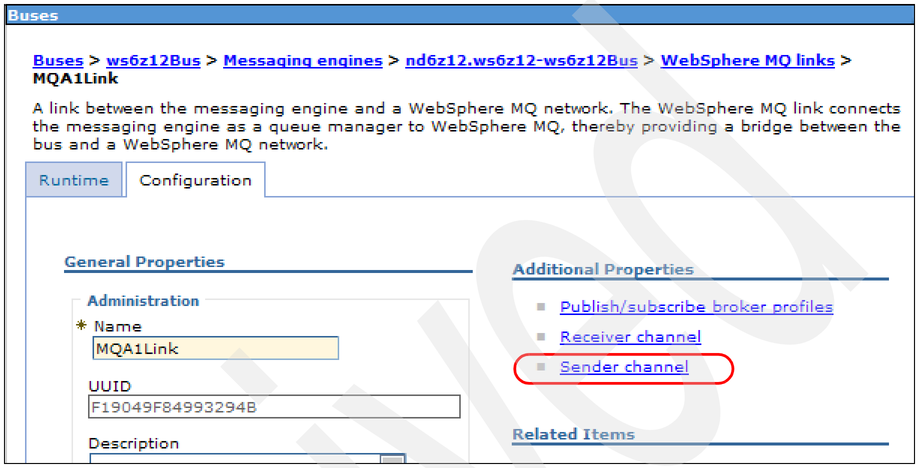
```
List Channels - MQA1                    Row 1 of 1

 Type action codes, then press Enter.  Press F11 to display connection status.
  1=Display   2=Define like   3=Alter   4=Manage   5=Perform
  6=Start     7=Stop

     Name                 Type          Disposition   Status
 <>  MQA1.to.ws6z12       CHANNEL       PRIVATE MQA1
     MQA1.to.ws6z12       SENDER        QMGR    MQA1  INACTIVE
                      ******** End of list ********









 Command ===>
  F1=Help      F2=Split     F3=Exit      F4=Filter    F5=Refresh   F7=Bkwd
  F8=Fwd       F9=SwapNext F10=Messages F11=Status   F12=Cancel
```

3. In the Start a Channel screen press Enter to confirm the start command and the MQ message CSQ9022I will appear, as shown in Example 6-6. It is important to have WebSphere Application Server up and running and listening on the incoming port. If this is not the case, WebSPhere MQ cannot start the channel and will give an error message.

Example 6-6 Starting the sender channel

```
Start a Channel

 Select disposition, then press Enter to start channel.


 Channel name  . . . . . . . : MQA1.to.ws6z12
 Channel type  . . . . . . . : SENDER
 Description . . . . . . . . :


 Disposition . . . . . . . . . P     P=Private on MQA1
                                     S=Shared on MQA1
                                     A=Shared on any queue manager






        EssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssN
        e CSQ9022I -MQA1 CSQXCRPS ' START CHANNEL' NORMAL COMPLETION e
        DssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssM
 Command ===>
  F1=Help      F2=Split     F3=Exit      F9=SwapNext F10=Messages F12=Cancel
```

4. If the sender channel on the WebSphere Application Server side is not started, start it using the Integrated Solutions Console. Expand **Service Integration** → **Buses** → **wsz12Bus** → **Message engine** → **nd6z12.ws6z12-ws6z12Bus** → **WebSphere MQ links** → **MQA1Link**, and at Additional Properties click **Sender channel**, as shown in Figure 6-53.



Buses

Buses > ws6z12Bus > Messaging engines > nd6z12.ws6z12-ws6z12Bus > WebSphere MQ links > MQA1Link

A link between the messaging engine and a WebSphere MQ network. The WebSphere MQ link connects the messaging engine as a queue manager to WebSphere MQ, thereby providing a bridge between the bus and a WebSphere MQ network.

Runtime | Configuration

General Properties

Administration

* Name: MQA1Link

UUID: F19049F84993294B

Description

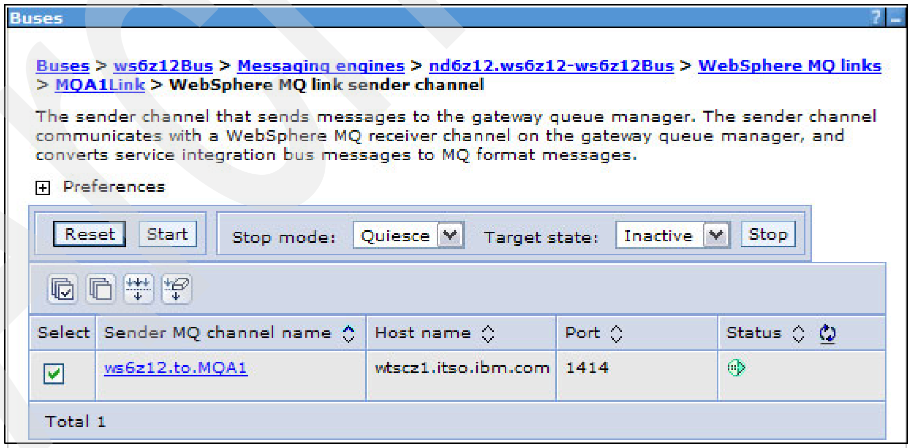
Additional Properties

- Publish/subscribe broker profiles
- Receiver channel
- Sender channel

Related Items

Figure 6-53 MQA1Link Additional Properties

5. On the WebSphere MQ link sender channel window select the channel **ws6z12.to.MQA1** and click **Start**, as shown in Figure 6-54.



Buses

Buses > ws6z12Bus > Messaging engines > nd6z12.ws6z12-ws6z12Bus > WebSphere MQ links > MQA1Link > WebSphere MQ link sender channel

The sender channel that sends messages to the gateway queue manager. The sender channel communicates with a WebSphere MQ receiver channel on the gateway queue manager, and converts service integration bus messages to MQ format messages.

Preferences

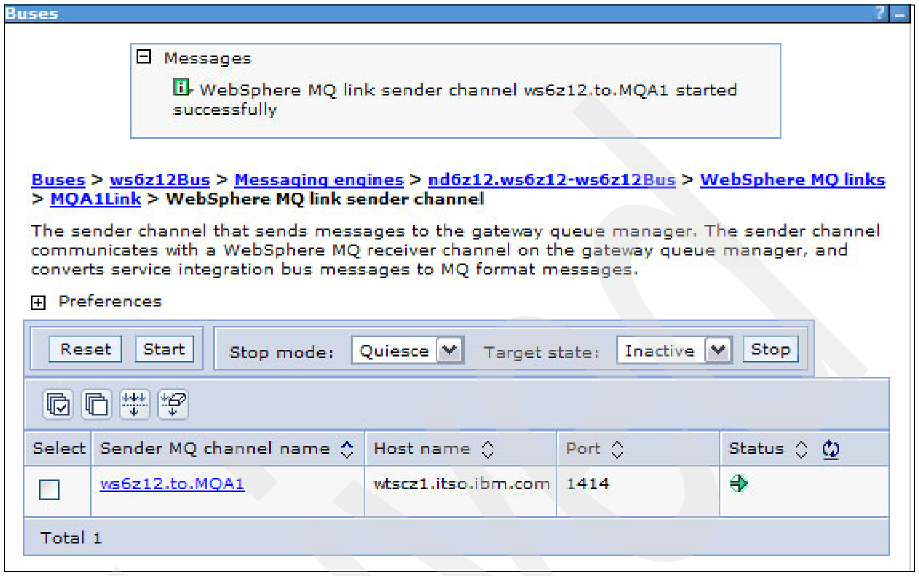
Reset | Start | Stop mode: Quiesce | Target state: Inactive | Stop

Select	Sender MQ channel name	Host name	Port	Status
✓	ws6z12.to.MQA1	wtscz1.itso.ibm.com	1414	⊕

Total 1

Figure 6-54 Starting the channel sender

If the start is successful, a message is issued at the Console and the status of the channel becomes a green solid arrow, as shown in Figure 6-55.



Messages

WebSphere MQ link sender channel ws6z12.to.MQA1 started successfully

Buses > ws6z12Bus > Messaging engines > nd6z12.ws6z12-ws6z12Bus > WebSphere MQ links > MQA1Link > **WebSphere MQ link sender channel**

The sender channel that sends messages to the gateway queue manager. The sender channel communicates with a WebSphere MQ receiver channel on the gateway queue manager, and converts service integration bus messages to MQ format messages.

Select	Sender MQ channel name	Host name	Port	Status
☐	ws6z12.to.MQA1	wtscz1.itso.ibm.com	1414	➔
Total 1				

Figure 6-55 Channel sender started

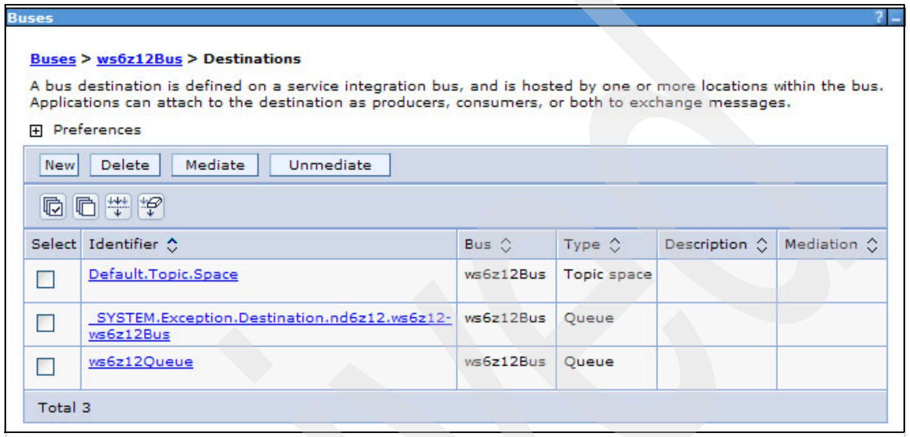
6.5 More WebSphere Application Server configurations

In the next sections we continue with the configuration on the WebSphere Application Server side.

6.5.1 Defining a foreign queue

We have to define the queue that will be used by foreign bus MQA1.

1. Expand **Service Integration** → **Buses** → **ws6z12Bus** → **Destinations** and click **New**, as shown in Figure 6-56.



Buses > ws6z12Bus > Destinations

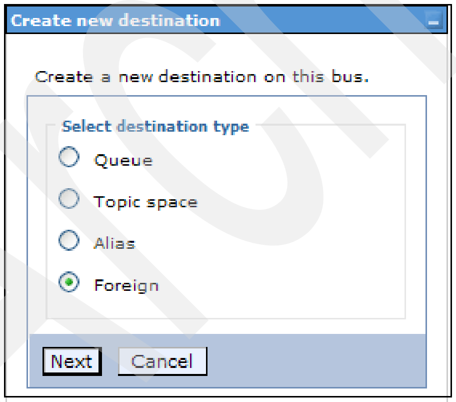
A bus destination is defined on a service integration bus, and is hosted by one or more locations within the bus. Applications can attach to the destination as producers, consumers, or both to exchange messages.

Select	Identifier	Bus	Type	Description	Mediation
☐	Default.Topic.Space	ws6z12Bus	Topic space		
☐	_SYSTEM.Exception.Destination.nd6z12.ws6z12-ws6z12Bus	ws6z12Bus	Queue		
☐	ws6z12Queue	ws6z12Bus	Queue		

Total 3

Figure 6-56 Defining a new foreign queue

2. In Create new destination, select **Foreign**, as shown in Figure 6-57.



Create new destination

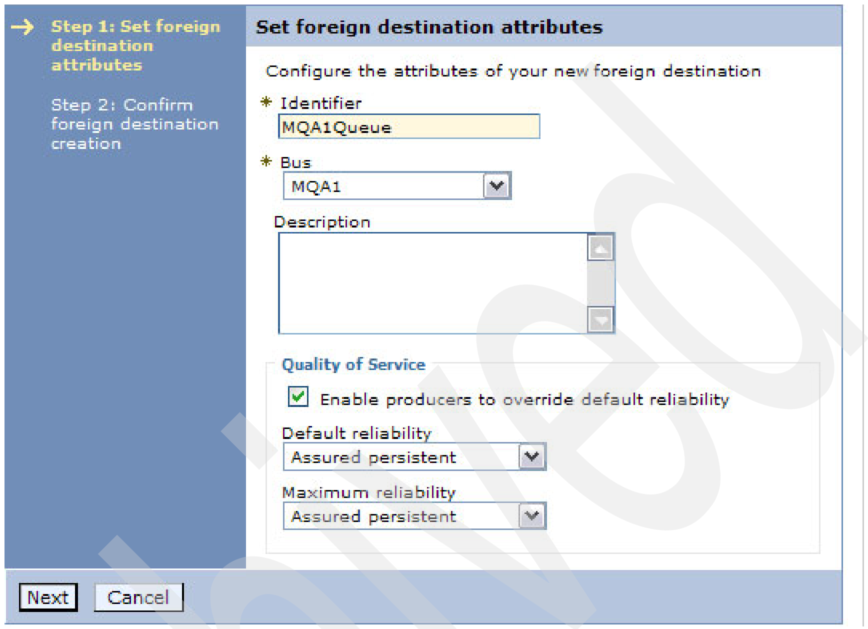
Create a new destination on this bus.

Select destination type

( ) Queue
( ) Topic space
( ) Alias
[✓ Foreign]

Figure 6-57 Selecting the destination type for the foreign queue

3. In “Step 1: Set foreign destination attributes” type **MQA1Queue** as the Identifier, select **MQA1** in the Bus combo box, and click **Next**. See Figure 6-58.


Step 1: Set foreign destination attributes
Step 2: Confirm foreign destination creation

Set foreign destination attributes

Configure the attributes of your new foreign destination

* Identifier
MQA1Queue

* Bus
MQA1

Description

Quality of Service

[✓] Enable producers to override default reliability

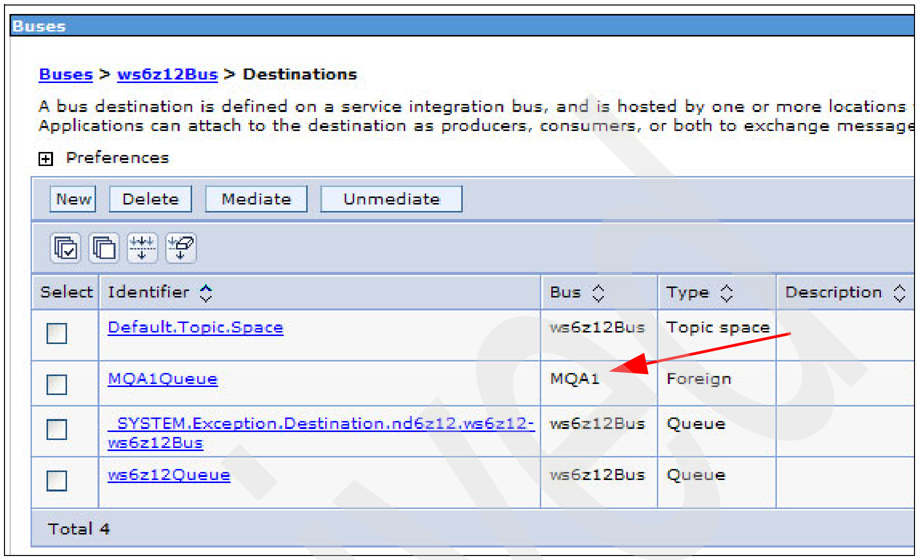
Default reliability
Assured persistent

Maximum reliability
Assured persistent

Figure 6-58 Defining a new foreign queue - attributes

4. Check the information and click **Finish**.

5. **Save** the changes and the foreign queue MQA1Queue is created. The result can be seen in Figure 6-59.



Buses

Buses > ws6z12Bus > Destinations

A bus destination is defined on a service integration bus, and is hosted by one or more locations
Applications can attach to the destination as producers, consumers, or both to exchange message

⊞ Preferences

New | Delete | Mediate | Unmediate

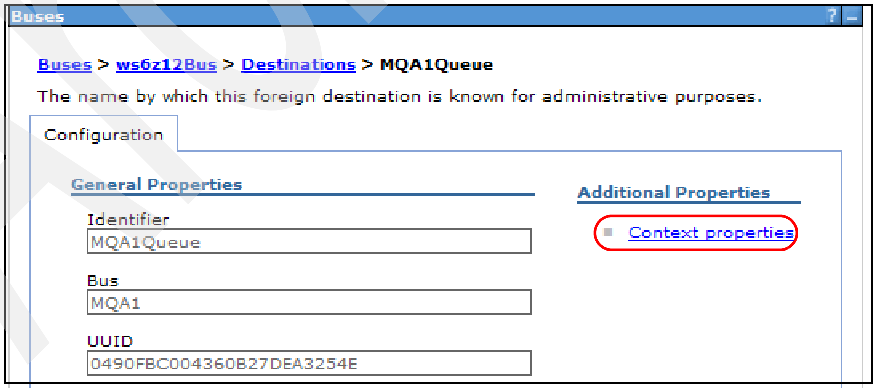
Select	Identifier	Bus	Type	Description
☐	Default.Topic.Space	ws6z12Bus	Topic space	
☐	MQA1Queue	MQA1	Foreign	
☐	_SYSTEM.Exception.Destination.nd6z12.ws6z12-ws6z12Bus	ws6z12Bus	Queue	
☐	ws6z12Queue	ws6z12Bus	Queue	

Total 4

Figure 6-59 Showing Destinations - Foreign queue

6. Because we want JMS messages to be sent to the external WebSphere MQ, we need to define destination context properties to the MQRFH2 header to be included in these messages.

   Expand **Service Integration** → **Buses** → **ws6z12Bus** → **Destinations** → **MQA1Queue** and select **Context properties**, as shown in Figure 6-60.



Buses

Buses > ws6z12Bus > Destinations > MQA1Queue

The name by which this foreign destination is known for administrative purposes.

Configuration

General Properties

Identifier: MQA1Queue

Bus: MQA1

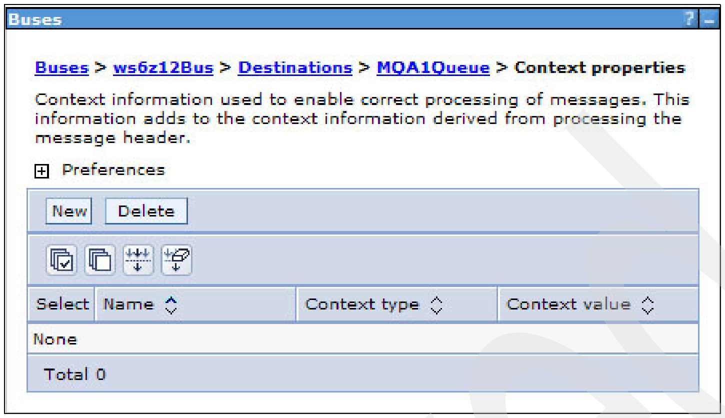
UUID: 0490FBC004360B27DEA3254E

Additional Properties

Context properties

Figure 6-60 Foreign queue properties

7. In the Context properties window, click **New**, as shown in Figure 6-61.



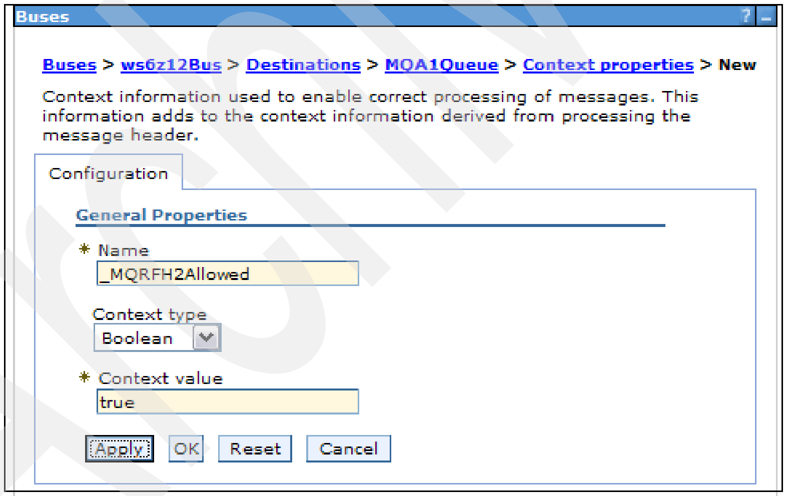
Buses > ws6z12Bus > Destinations > MQA1Queue > Context properties

Context information used to enable correct processing of messages. This information adds to the context information derived from processing the message header.

Select	Name	Context type	Context value
None			
Total 0			

Figure 6-61 Foreign queue properties

8. Type `_MQRH2Allowed` as the Name, select Context Type **Boolean**, type `true` as the Context value, and click **OK**. See Figure 6-62.



Buses > ws6z12Bus > Destinations > MQA1Queue > Context properties > New

Context information used to enable correct processing of messages. This information adds to the context information derived from processing the message header.

* Name
_MQRFH2Allowed

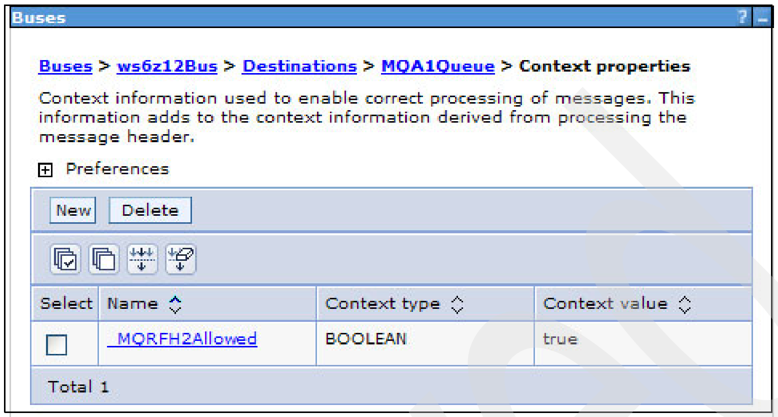
Context type
Boolean

* Context value
true

Figure 6-62 Create a foreign queue property

9. Check the information and click **Finish**.

10. **Save** the changes, and the context property _MQRH2Allowed is created. See Figure 6-63.


Buses

Buses > ws6z12Bus > Destinations > MQA1Queue > **Context properties**

Context information used to enable correct processing of messages. This information adds to the context information derived from processing the message header.

⊞ Preferences

Select	Name	Context type	Context value
☐	_MQRFH2Allowed	BOOLEAN	true

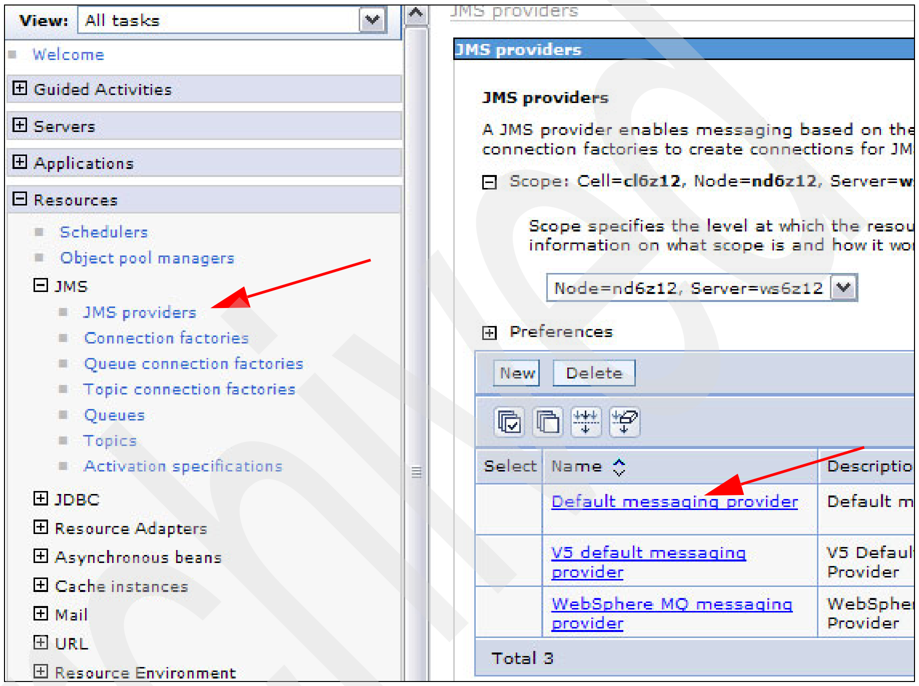
Total 1

Figure 6-63 Foreign queue property created

6.5.2 Defining the JMS queue to the foreign queue

We now need a JMS queue definition that will point to the MQA1Queue foreign queue.

1. Expand **Resources** → **JMS** → **JMS providers** and click **Default messaging provider**, as shown in Figure 6-64.



JMS providers

JMS providers

A JMS provider enables messaging based on the
connection factories to create connections for JM

Scope: Cell=**cl6z12**, Node=**nd6z12**, Server=**w**

Scope specifies the level at which the resou
information on what scope is and how it wo

Node=nd6z12, Server=ws6z12

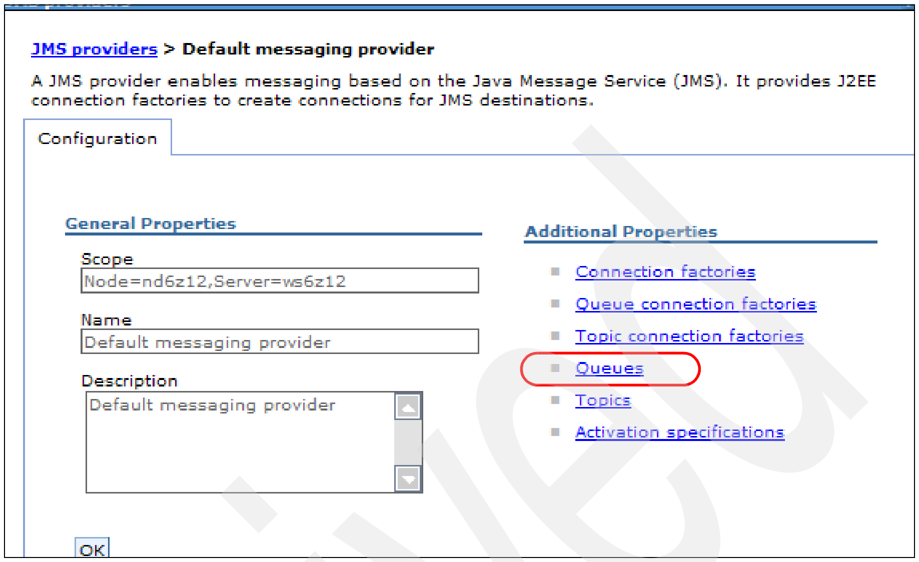
Preferences

Select	Name	Descriptio
	Default messaging provider	Default m
	V5 default messaging provider	V5 Defaul Provider
	WebSphere MQ messaging provider	WebSphe Provider

Total 3

Figure 6-64 Creating a new JMS queue

2. Under the Additional Properties select **Queues**, as shown in Figure 6-65.


JMS providers > Default messaging provider

A JMS provider enables messaging based on the Java Message Service (JMS). It provides J2EE connection factories to create connections for JMS destinations.

Configuration

General Properties

Scope: Node=nd6z12,Server=ws6z12

Name: Default messaging provider

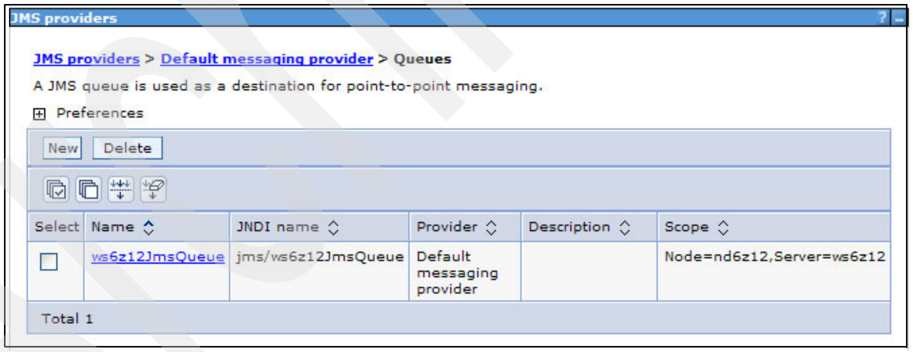
Description: Default messaging provider

Additional Properties

- Connection factories
- Queue connection factories
- Topic connection factories
- Queues
- Topics
- Activation specifications

Figure 6-65 Selecting Queues

3. This will show the existing JMS queues, as shown in Figure 6-66. Click **New**.



JMS providers > Default messaging provider > Queues

A JMS queue is used as a destination for point-to-point messaging.

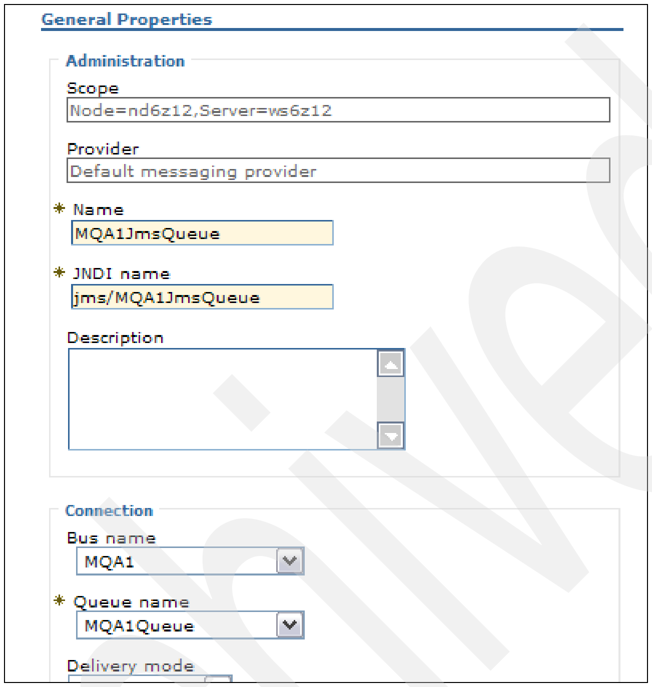
⊞ Preferences

Select	Name	JNDI name	Provider	Description	Scope
☐	ws6z12JmsQueue	jms/ws6z12JmsQueue	Default messaging provider		Node=nd6z12,Server=ws6z12

Total 1

Figure 6-66 Defining a new JMS queue to the foreign queue

4. On the General Properties window, type `MQA1JmsQueue` as the Name, `jms/MQA1JmsQueue` as the JNDI name, select **MQA1** in the Bus name combo box, select **MQA1Queue** in the Queue name combo box, and click **OK**. See Figure 6-67.

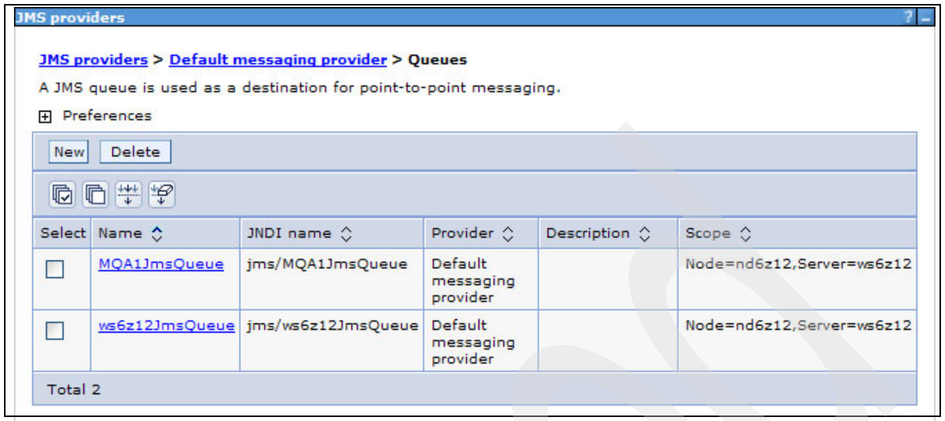


Scope: Node=nd6z12,Server=ws6z12
Provider: Default messaging provider
* Name: MQA1JmsQueue
* JNDI name: jms/MQA1JmsQueue
Description:
Bus name: MQA1
* Queue name: MQA1Queue

Figure 6-67 Defining a new JMS queue to the foreign queue - General Properties

5. At the confirmation screen, verify the information and click **Finish**.
6. Save the changes.

The Queue MQA1JmsQueue is created, as shown in Figure 6-68.



A JMS queue is used as a destination for point-to-point messaging.

Select	Name	JNDI name	Provider	Description	Scope
☐	MQA1JmsQueue	jms/MQA1JmsQueue	Default messaging provider		Node=nd6z12,Server=ws6z12
☐	ws6z12JmsQueue	jms/ws6z12JmsQueue	Default messaging provider		Node=nd6z12,Server=ws6z12

Total 2

Figure 6-68   New JMS queue to the Foreign queue

6.5.3 Defining the WebSphere MQ queues and process

We need to define the WebSphere MQ queues that will communicate with WebSphere Application Server queues.

A job similar to the one shown in Example 6-7 can be used for this.

Example 6-7   Defining WebSphere MQ queues

```
//ITSO011 JOB 'ITSO01-MQ',MSGLEVEL=(1,1),MSGCLASS=H,
//    REGION=0M,CLASS=A,NOTIFY=&SYSUID
//*
//DISP EXEC PGM=CSQUTIL,PARM='MQA1'
//STEPLIB DD DISP=SHR,DSN=MQ600.SCSQANLE
//        DD DISP=SHR,DSN=MQ600.SCSQAUTH
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
 COMMAND
//CSQUCMD DD *
define qremote('ws6z12Queue') +
       rname('ws6z12Queue') +
       rqmname('ws6z12Bus') +
       xmitq('ws6z12') +
       replace

define process('ws6z12Proc') +
       appltype(CICS) +
       applicid(CPIL)+
       replace

define qlocal('MQA1Queue') +
       process('ws6z12Proc') +
       trigger +
       trigtype(first) +
       trigdata('/itso/CUSTINQ') +
       initq(CITSO01.INITQ) +
       bothresh(1) +
       replace
```

Make sure that this job finishes with `return code=0` and `0 failed`, as shown in Example 6-8.

Example 6-8 Job output from defining queues

```
CSQU057I  3 commands read
 CSQU058I  3 commands issued and responses received, 0 failed
 CSQU143I  1 COMMAND statements attempted
 CSQU144I  1 COMMAND statements executed successfully
 CSQU148I CSQUTIL Utility completed, return code=0
```

6.5.4 CICS Transaction Server configuration

The configuration of the CICS TS was not our objective for this book.

7

Using WebSphere Message Broker for connectivity

WebSphere Message Broker (WMB) is a powerful integration solution for building Service Oriented Architectures. The communication through the Broker is based on the principles of messaging, but with many enhancements. Messages flow through the Broker between disparate applications and across multiple hardware and software platforms. Business rules can be applied to the data that is flowing through the Broker in order to route, store, retrieve, and transform the information.

This chapter demonstrates the use of the WebSphere Message Broker (WMB) as an Enterprise Service Bus (ESB). We go through the tasks to create mediations that isolate a client application from the backend server application.

At a high level, this chapter is organized into the following sections:

- Message Broker Overview, discussed in 7.1, “WebSphere Message Broker overview” on page 295, with the following subtopics:
  - Capabilities
    - Message routing
    - Message transformation and enrichment
  - Components

    - Development environment (application development perspective, configuration manager, message flows, message sets)
    - Runtime environment (Broker, Execution groups, Broker domain, User name server)
- Sample applications, discussed in 7.2, “Sample applications” on page 300, with the following subtopics:
  - Routing scenario
    - Defining the message format
    - Defining the sample message flow and customizing the message
    - Deploying and testing
  - Protocol conversion and message transformation scenario
    - Defining the message format and message sets
    - Designing the sample message flow and customizing the message
    - Deploying and testing
  - Web Services call out from the Broker

Finally, in 7.3, “Additional material” on page 347, we provide information about the supplied material for this book.

7.1 WebSphere Message Broker overview

WebSphere Message Broker (WMB) provides many features, such as:

- Optimization for efficiently analyzing (parsing) incoming data
- Efficient stateless data transformations
- A wide range of direct data access methods for message enrichment
- Routing of WebSphere MQ and HTTP transports between end points
- A simple graphic and standards-based development environment
- A robust and highly scalable execution environment

7.1.1 Capabilities

The primary capabilities of WebSphere Message Broker are *message routing*, *message transformation*, *message enrichment*, and *publish/subscribe*. Together, these capabilities make WebSphere Message Broker a powerful tool for business integration.

Message routing

WebSphere Message Broker provides connectivity for both standards-based and non-standards-based applications and services. The routing can be simple point-to-point routing or it can be based on matching the content of the message to business rules defined to the Broker.

WebSphere Message Broker contains a choice of transports that enable secure business to be conducted at any time and any place, providing powerful integration using mobile, telemetry, and Internet technologies. WebSphere Message Broker is built upon WebSphere MQ and therefore supports the same transports. However, it also extends the capabilities of WebSphere MQ by adding support for other protocols, including real-time Internet, intranet, and multicast endpoints.

Message transformation and enrichment

Transformation and enrichment of in-flight messages are important capabilities of WebSphere Message Broker, and allow for business integration without the requirement of any additional logic in the applications themselves. You can transform messages between applications to use different formats, for example, transforming from a custom format in an existing system to XML messages that you can use with a Web Service. This provides a powerful mechanism to unify organizations, because business information can now be distributed to applications that handle completely different message formats without a requirement to reprogram or add to the applications themselves.

Messages can also be transformed and enriched by integration with multiple sources of data, such as databases, applications, and files. This allows for any type of data manipulation, including logging, updating, and merging. For the messages that flow through the Broker, you can store business information in databases or you can extract them from databases and files and add to the message for processing in the target applications.

You can perform complex manipulation of message data using the facilities provided in the Message Broker Toolkit, such as ESQL and Java.

In WebSphere Message Broker, message transformation and enrichment is dependant upon a Broker understanding the structure and content of the incoming message. Self-defining messages, such as XML messages, contain information about their own structure and format. However, before other messages, such as custom format messages, can be transformed or enhanced, a message definition of their structure must exist. The WebSphere Message Broker Toolkit contains facilities for defining messages to the Message Broker. These facilities are discussed in more detail in this chapter.

7.1.2 Components

WebSphere Message Broker is comprised of two principal parts: a *development environment* for the creation of message flows, message sets, and other message flow application resources, and a *runtime environment*, which contains the components for running those message flow applications that are created in the development environment. Figure 7-1 on page 298 gives the overview.

Development environment

The development environment is where the message flow applications that provide the logic to the Broker are developed. The Broker uses the logic in the message flow applications to process messages from business applications at runtime. In the Message Broker Toolkit, you can develop both message flows and message sets for message flow applications.

Application Development perspective

The *Application Development perspective* is the part of the Message Broker Toolkit that is used to design and develop message flows and message sets. It contains editors that create message flows, and transformation code, such as ESQL or Java, that define messages.

This perspective is also used for the deployment of message flows and message sets to Brokers in the defined Broker domains. The *Administration perspective* also contains tools for creating WebSphere Message Broker archive (bar) files. Bar files are used to deploy message flow application resources, such as message flows and message sets.

The Administration perspective also contains tools to help test message flows. These tools include Enqueue and Dequeue for putting and getting messages from WebSphere MQ queues.

Configuration Manager

The *Configuration Manager* is the interface between the WebSphere Message Broker Toolkit and the Brokers in the Broker domain. The Configuration Manager stores configuration details for the Broker domain in an internal repository, providing a central store for resources in the Broker domain.

The Configuration Manager is responsible for deploying message flow applications to the Brokers. The Configuration Manager also reports back on the progress of the deployment and on the status of the Broker. When the WebSphere Message Broker Toolkit connects to the Configuration Manager, the status of the Brokers in the domain is derived from the configuration information stored in the Configuration Manager's internal repository.

Message flows

Message flows are programs that provide the logic that the Broker uses to process messages from business applications. Message flows are created by connecting *nodes* together, with each node providing some basic logic. A selection of built-in nodes is provided with WebSphere Message Broker for performing particular tasks. These tasks can be combined to perform complex manipulations and transformations of messages. Methods are available for defining the logic in the message flows to transform data. Depending on the different types of data or the skills of the message flow application developer, the following options are available:

- Extended Structured Query Language (ESQL)
- Java
- Extensible Style-sheet Language for Transformations (XSLT)
- Drag-and-drop mappings

The nodes in the message flows define the source and the target transports the message, any transformations and manipulations based on the business data, and any interactions with other systems such as databases and files.

Message sets

A *message set* is a definition of the structure of the messages that are processed by the message flows in the Broker. As mentioned previously, in order for a message flow to be able to manipulate or transform a message, the Broker must know the structure of the message. You can use the definition of a message to verify the message structure and to assist with the construction of message flows and mappings in the WebSphere Message Broker Toolkit.

Message sets are compiled for deployment to a Broker as a message dictionary, which provides a reference for the message flows to verify the structure of messages as they flow through the Broker.

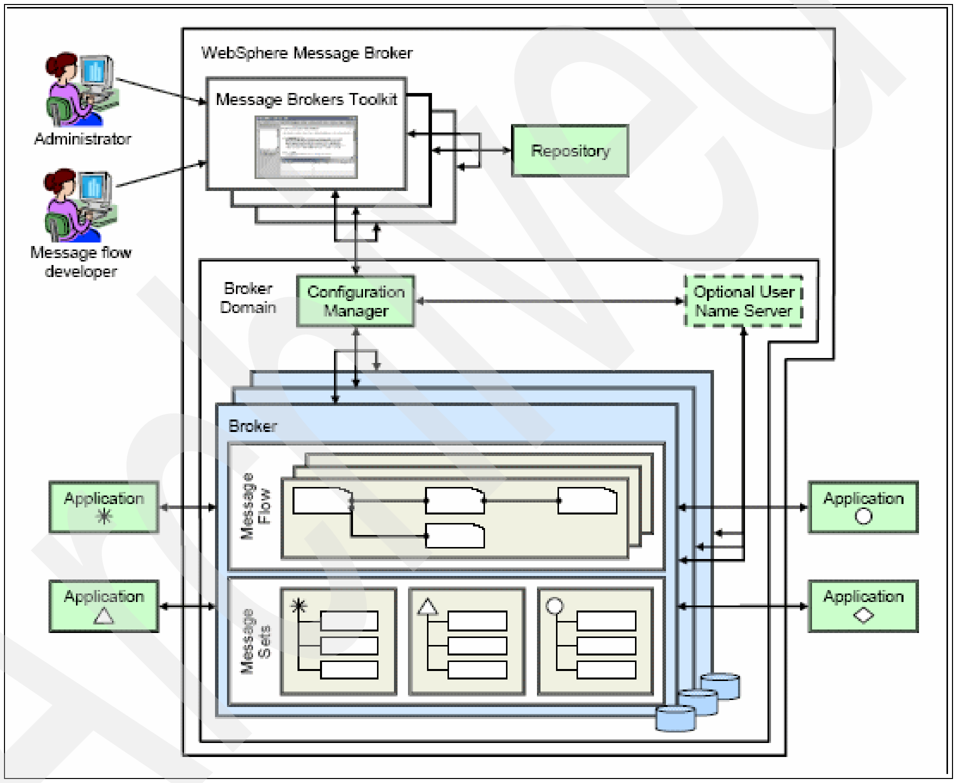


Figure 7-1   WMB environment

Runtime environment

The runtime consists of the *Broker*, one or multiple *execution groups* and eventually a *Broker domain* and a *User Name Server*.

Note that the Configuration Manager is logically part of the development environment, but runs on the runtime (deployment) platform.

Broker

The *Broker* is a set of application processes that host and run message flows. When a message arrives at the Broker from a business application, the Broker processes the message before passing it on to one or more other business applications. The Broker routes, transforms, and manipulates messages according to the logic that is defined in their message flow applications.

A Broker uses WebSphere MQ as the transport mechanism both to communicate with the Configuration Manager, from which it receives configuration information, and to communicate with any other Brokers to which it is associated. Each Broker has a database in which it stores the information that it has to process messages in at runtime.

Execution groups

Execution groups enable message flows within the Broker to be grouped. Each Broker contains a default execution group. You can create additional execution groups, if they are given unique names within the Broker. Each execution group is a separate operating system process and, therefore, the contents of an execution group remain separate from the contents of other execution groups within the same Broker. This can be useful for the isolating pieces of information in regard to security, because the message flows execute in separate address spaces or as unique processes.

Message flow applications are deployed to a specific execution group. To enhance performance, the same message flows and message sets can be running in different execution groups.

Broker domain

Brokers are grouped together in *Broker domains*. The Brokers in a single Broker domain share a common configuration that is defined in the Configuration Manager. A Broker domain contains one or more Brokers and a single Configuration Manager. It can also contain a *User Name Server*. The components in a Broker domain can exist on multiple machines and operating systems, and are connected with WebSphere MQ channels. A Broker might belong to only one Broker domain.

User Name Server

A *User Name Server* is an optional component that is required only when publish/subscribe message flow applications are running, and where extra security is required for applications to be able to publish or subscribe to topics. The User Name Server provides authentication for topic-level security for users and groups that are performing publish/subscribe operations.

7.2 Sample applications

In this chapter we discuss a few scenarios typical for a z/OS runtime environment. In all cases the flows are driven by a Web application running in WebSphere Application Server, driving the requests into WMB. Based on the flows developed, WMB will decide where to send the request and in which format to send it. The following interfaces are used to access the back-end applications in our scenario:

- DB2 using the DB2 node
- VSAM file reads using the VSAM node
- CICS program calls via EXCI using the EXCI node
- CICS program calls by Web services over HTTP using the SOAP node

We use the WebSphere Message Broker Toolkit to build and illustrate routing, protocol conversion and data transformation using the above back-end interfaces.

7.2.1 Routing

We now demonstrate WMB routing in which we design the message flow to determine whether to access DB2 or a VSAM data set, depending on the values of the input data. This is known as *content-based routing*.

We also demonstrate methods for accessing DB2 and VSAM data. The data retrieved is similar in both cases, and it is the flow's responsibility to normalize the data, that is, to ensure that the calling client does not see any difference in the format of the data regardless of the back-end source (DB2 or VSAM). This is a common role for an Enterprise Service Bus. The mediation should determine where to route requests and compensate for any eventual differences in the back-end systems. Figure 7-2 on page 301 shows our flow. The numbers in the bullets correspond with the numbers in the diagram.

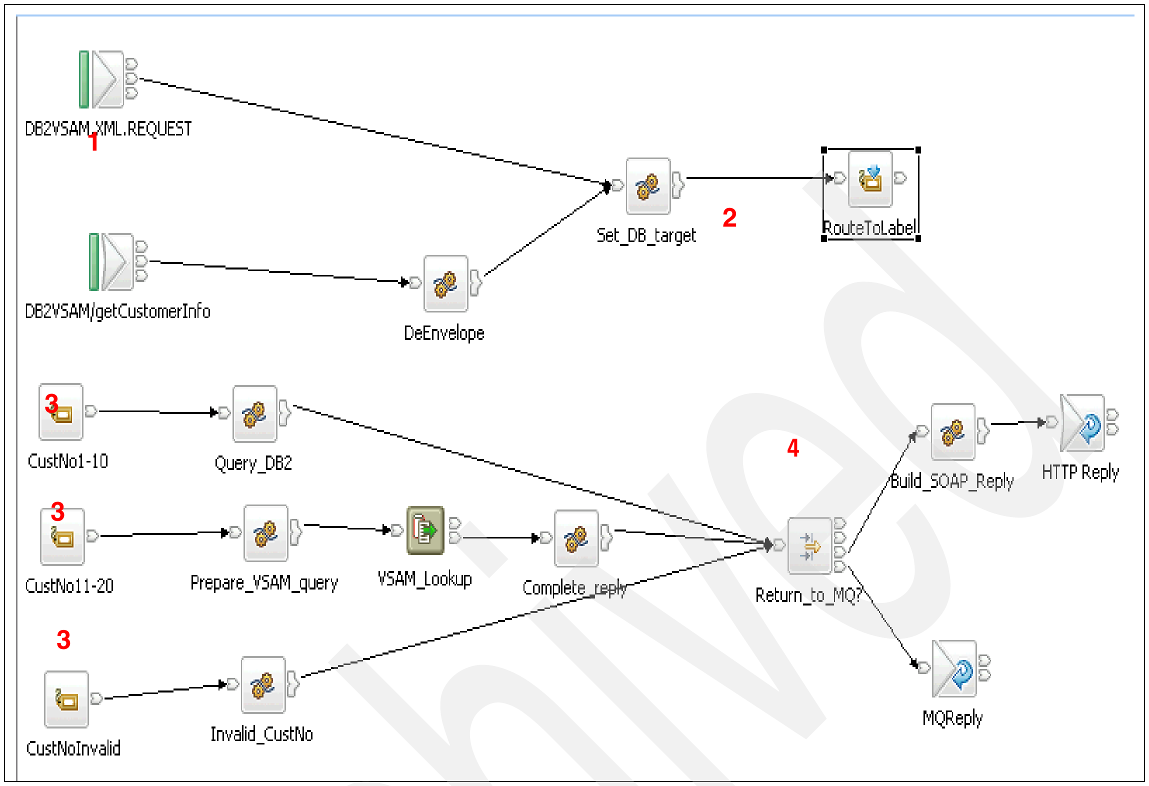


Figure 7-2 Message flow demonstrating content-base routing and data access

The flow begins by accepting input from two potential sources. The numbers in Figure 7-2 correspond to the description that follows:

1. Accept input either as XML transported over MQ (the MQInput node labeled DB2VSAM.XML.REQUEST) or alternatively as SOAP transported over HTTP (the HTTPInput node labeled DB2VSAM/getCustomerInfo). The MQ message content is shown here. The customer number is a key field that will later be used to query the DB2 or VSAM data.

Example 7-1 Application part of a message

```
<Customer>
<CustNo>8</CustNo>
</Customer>
```

The SOAP part of the input message sent over HTTP looks as follows:

```
<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">
   <soapenv:Body>
      <CustomerReq>
         <CustNo>9</CustNo>
      </CustomerReq>
   </soapenv:Body>
</soapenv:Envelope>
```

2. After a few small data manipulations performed by *Compute* nodes, the RouteToLabel node is used to perform the routing based upon the input data.
3. In the lower part of the flow, three label nodes can be seen (CustNo1-10, and so on). These nodes are the targets of the RouteToLabel node (2). Depending upon the nature of the input data, one of the following three paths is taken:
   - CustNo1-10: In this branch, the Compute node, Query_DB2, performs the direct query to the DB2 database.
   - CustNo11-20: In this branch, a series of nodes prepares for the direct VSAM query, followed by the query and additional logic to post-process the resulting data.
   - CustNoInvalid: In this third possible branch, the Compute node Invalid_CustNo simply prepares an error code to be returned.
4. The remaining part of the flow provides simple logic (Filter node) to determine the format and transport of the response, either XML over MQ or SOAP over HTTP. An example of the application part of the output message is shown here:

```
<CustomerReply>
<CustNo>8</CustNo>
<LastName>Farkas</LastName>
<FirstName>Carl</FirstName>
<Address1>1 rue de la Paix</Address1>
<City>Paris</City><State>FR</State>
<Country>FRANCE</Country>
<RetCode>0</RetCode>
</CustomerReply>
```

Given this basic logic used, we can begin to develop this flow. We explain each node, as well as providing instructions to configure several of the nodes.

The development process entails:

1. Defining the format of any messages that will be used (not always required; the Broker can recognize certain message types dynamically.)
2. Designing the flow
3. Filling in the parameters for the nodes and doing any necessary programming
4. Deploying and testing the flow

Defining the format of the message

We define and build the format for the VSAM record definition:

1. Start the WebSphere Message Broker Toolkit. Select **Start** → **Programs** → **IBM WebSphere Message Brokers 6.0** → **WebSphere Message Brokers Toolkit**

   (or simply use the icon shortcut on the Windows desktop).

   Enter a workspace location and name of choice in the Workspace prompt (if not already there) and click **OK**.

   The WMB Toolkit includes several *perspectives*. A perspective is a precustomized set of window panes, organized in a particular way to facilitate the development. For this part we will use the Broker Application Development perspective. The current perspective is displayed in the title bar of the Toolkit window. If this perspective is not the correct one, you should switch to the correct one: Select **Window** → **Open Perspective** → **Other** → **Broker Application Development (default).**

2. Right-click in any blank area of the Broker Development pane (upper left window) and select **New** → **Message Set**, as shown in Figure 7-3.

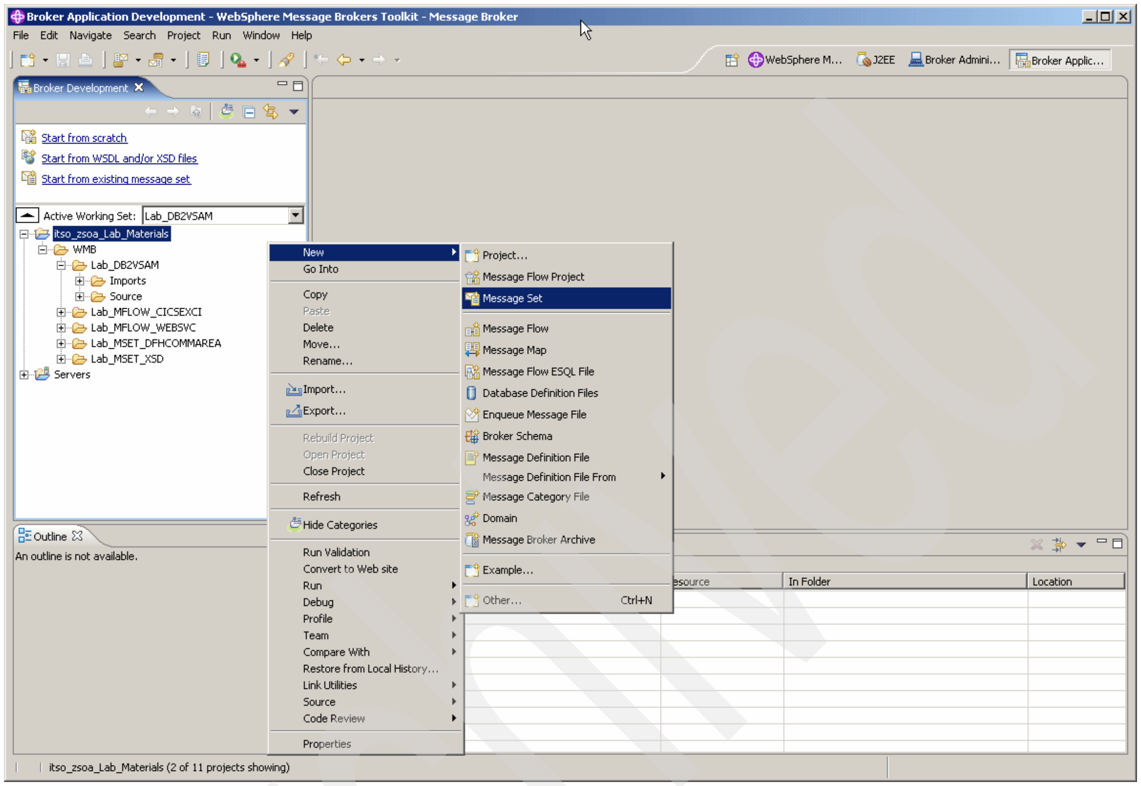
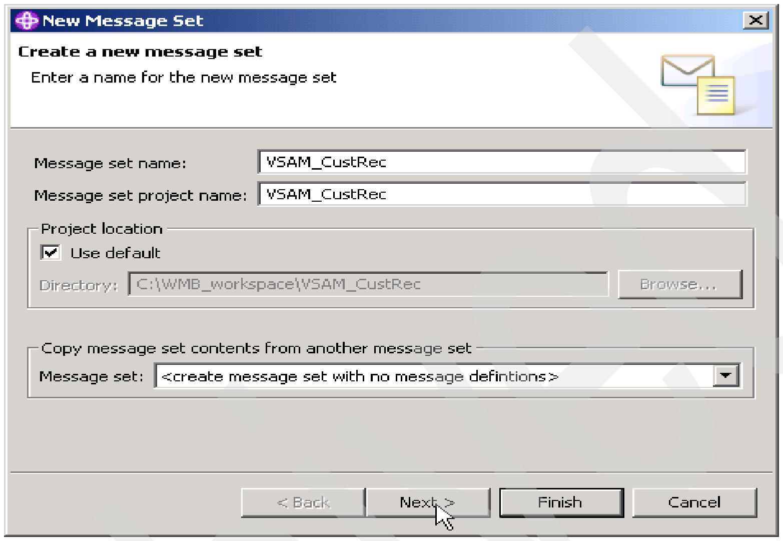


Figure 7-3 New message set

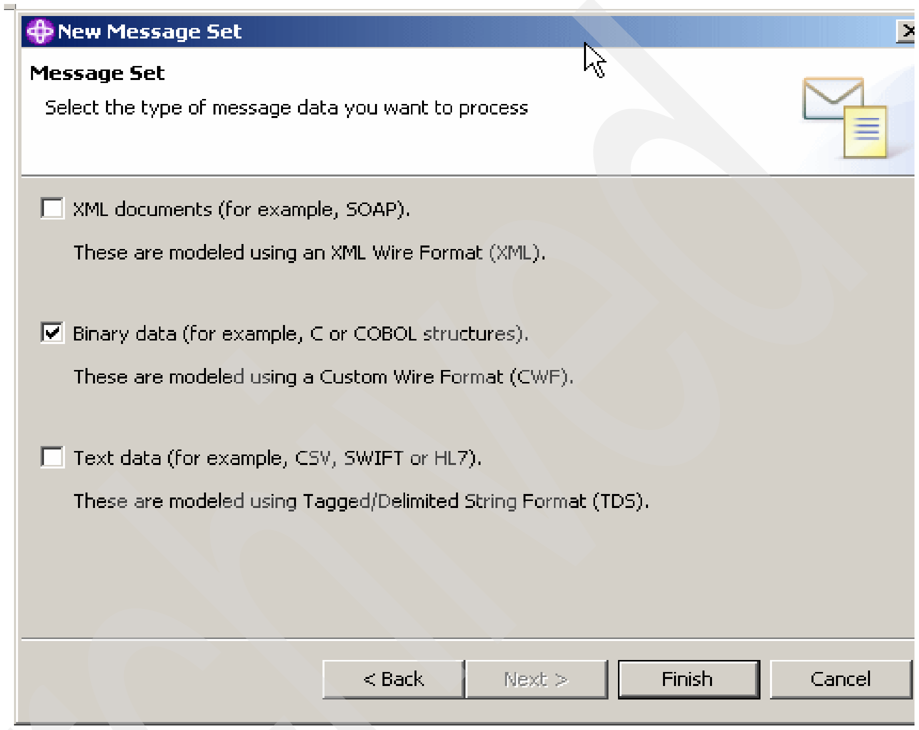
3. Enter the name for the Message Set. The wizard automatically uses the same name for the Message Set project, which can generally be accepted. Note that the message set name typically should be unique in a Broker. We entered `VSAM_CustRec`. Select **Next**. See Figure 7-4.



Message set name: VSAM_CustRec
Message set project name: VSAM_CustRec
Project location
[✓] Use default
Directory: C:\WMB_workspace\VSAM_CustRec
Copy message set contents from another message set
Message set: <create message set with no message defintions>

Figure 7-4 Name the message set

4. The physical message format window is presented next. This message set definition is used to parse data read from a VSAM file. The data is retrieved in a simple structure-like format as used in C or COBOL. You should therefore uncheck the default **XML documents** choice and check the **Binary data** choice. See Figure 7-5.



Message Set

Select the type of message data you want to process

[ ] XML documents (for example, SOAP).

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

[✓] Binary data (for example, C or COBOL structures).

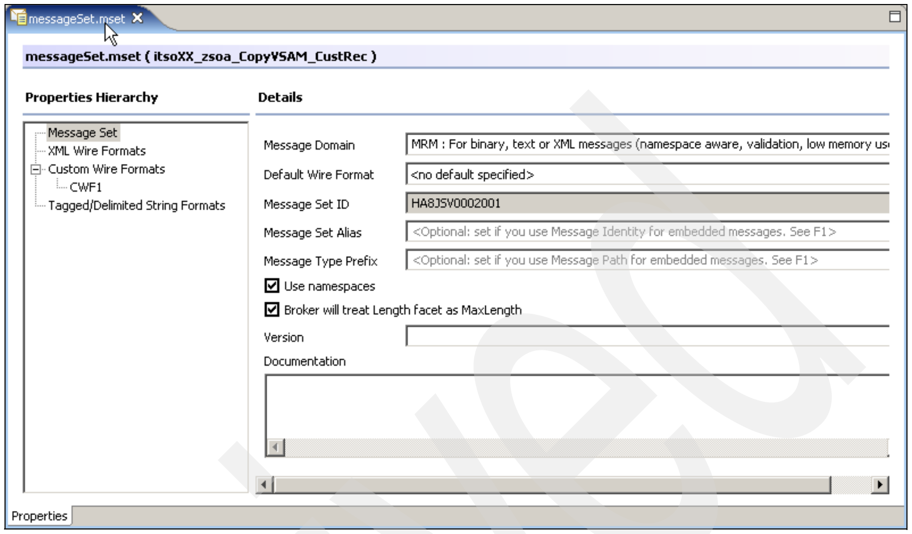
These are modeled using a Custom Wire Format (CWF).

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

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

Figure 7-5 Physical type choice

5. Click **Finish**. The wizard should create the new folder and message set. The message set editor window should be displayed, as shown in Figure 7-6.


messageSet.mset (itsoXX_zsoa_CopyVSAM_CustRec)

Properties Hierarchy

- Message Set
- XML Wire Formats
- Custom Wire Formats
  - CWF1
- Tagged/Delimited String Formats

Details

Message Domain	MRM : For binary, text or XML messages (namespace aware, validation, low memory us
Default Wire Format	<no default specified>
Message Set ID	HA8JSV0002001
Message Set Alias	<Optional: set if you use Message Identity for embedded messages. See F1>
Message Type Prefix	<Optional: set if you use Message Path for embedded messages. See F1>

☑ Use namespaces

☑ Broker will treat Length facet as MaxLength

Version

Documentation

Figure 7-6 Message set editor

The wizard should create the new folder and message set. The message set editor window should be displayed as seen in Figure 7-6. Note here the alphanumeric Message Set ID that is displayed—it will be referenced in later sections. You can now close the message set editor window (click the **X** on the tab).

6. We now import the COBOL COPYBOOK definition of the VSAM file structure into this message set. This copybook is supplied as additional material in this book. As you can see, this is a standard COPYBOOK definition that the z/OS programmer of the VSAM application provided. We import this into the message set created in the last step. The WebSphere Message Broker allows you to build message definitions manually using the Toolkit and a simple graphical interface, but it is more typical to import the definition when possible from a variety of sources such as a COPYBOOK, a C header, WSDL, an XML schema, or others.

Example 7-2 VSAMREC.cpy

```
01  CUSTREC.
02  CustNo      PIC 9(3).
02  LastName    PIC X(25).
02  FirstName   PIC X(15).
02  Address1    PIC X(20).
02  City        PIC X(20).
```

```
02  State        PIC X(5).
02  Country      PIC X(15).
```

7. The next step creates a *message definition* file (.mxsd), which describes the physical as well as logical structure of a message. This is done by importing the COBOL COPYBOOK file viewed above. Navigate to **VSAM_CustRec** → **VSAM_CustRec** → **Message Definitions** → **New** → **Message Definition File From** → **COBOL File** and press **Next**. See Figure 7-7.

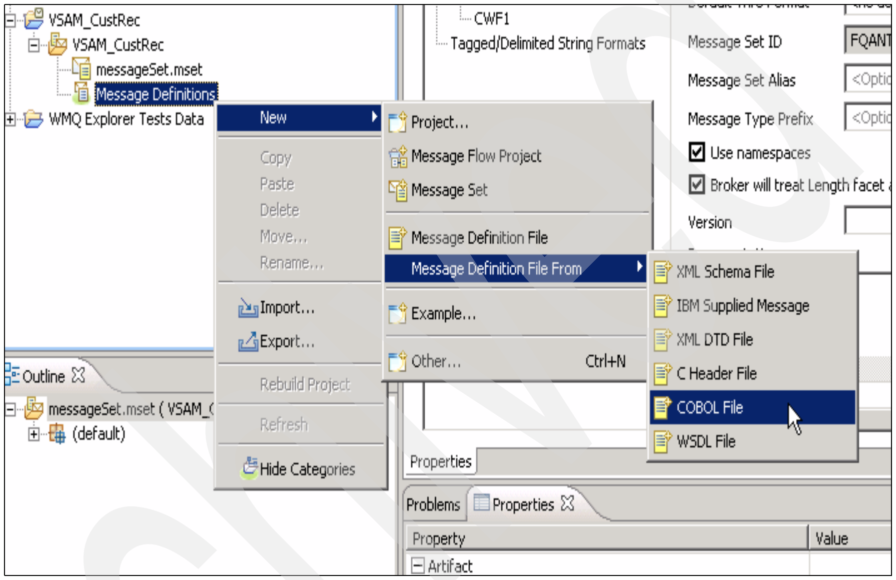


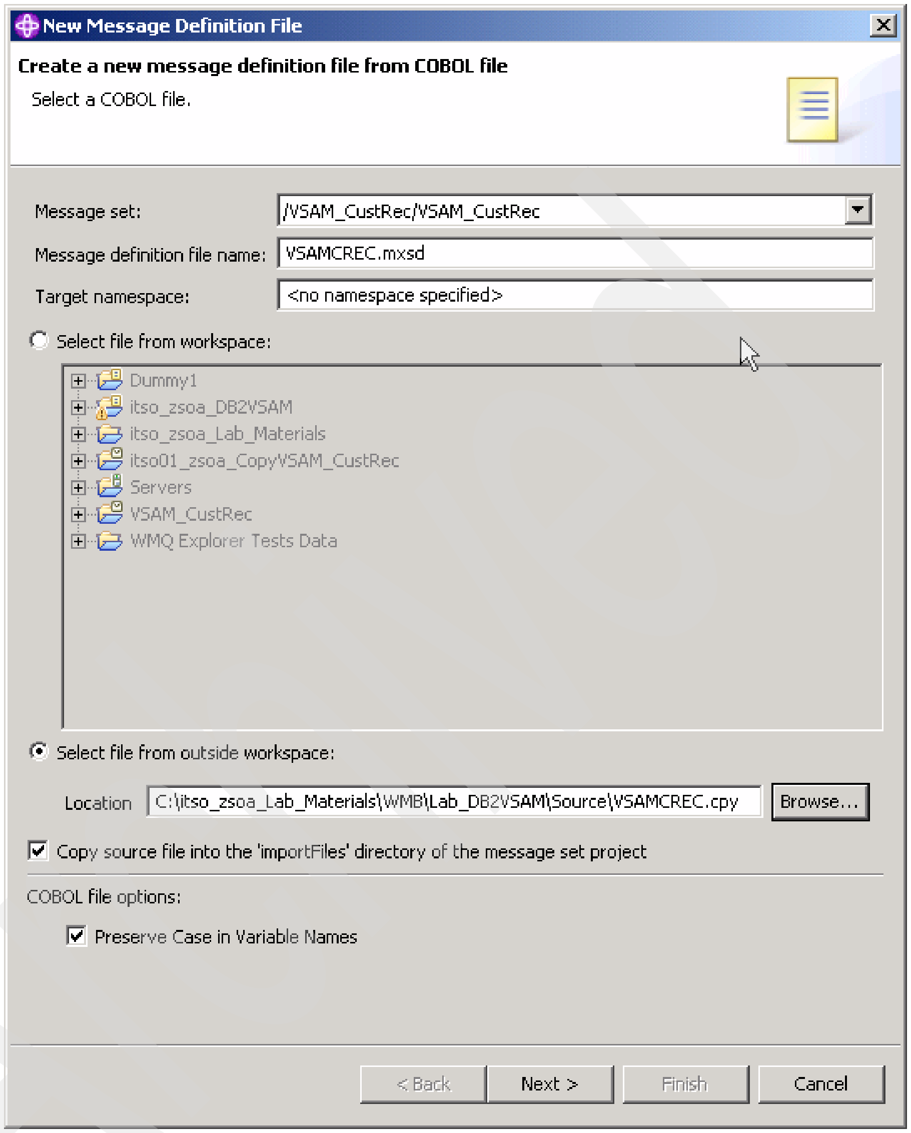
Figure 7-7 New message definition file

8. The next window displays the various structures that the importer found in the COPYBOOK. In this case there is only one source structure. We select the COBOL structure to be used in this window:

   a. Click the structure name **CUSTREC** in the Source structure pane.

   b. Click the **>** button to move it to the Imported structure pane.

   c. Now select (check) the **CUSTREC** structure on the right side.


New Message Definition File

Create a new message definition file from COBOL file

Select a COBOL file.

Message set: /VSAM_CustRec/VSAM_CustRec

Message definition file name: VSAMCREC.mxsd

Target namespace: <no namespace specified>

( ) Select file from workspace:

(•) Select file from outside workspace:

Location C:\itso_zsoa_Lab_Materials\WMB\Lab_DB2VSAM\Source\VSAMCREC.cpy

[✓] Copy source file into the 'importFiles' directory of the message set project

COBOL file options:

[✓] Preserve Case in Variable Names

Figure 7-8 Choose to import the copybook from outside workspace

9. Click **Finish** to complete the message definition file creation.

   At this point the message definition editor window should be visible in the upper right pane, and the **Outline** pane in the lower left should show the **VSAMCREC.mxsd** hierarchy. Navigate to the **VSAMCREC.mxsd** → **Messages** → **msg_CUSTREC** → **LastName** element. Now within the message definition pane, select the **Properties** tab. In the Properties page, select the **Local Element** under CWF1 to display the physical representation of the LastName field.

You should see the field length count of 25, for example, which corresponds to the COBOL X(25) definition. Other characteristics such as padding to apply, character justification, numeric representation system (packed decimal versus, for example), are all determined here in the Physical representation section.

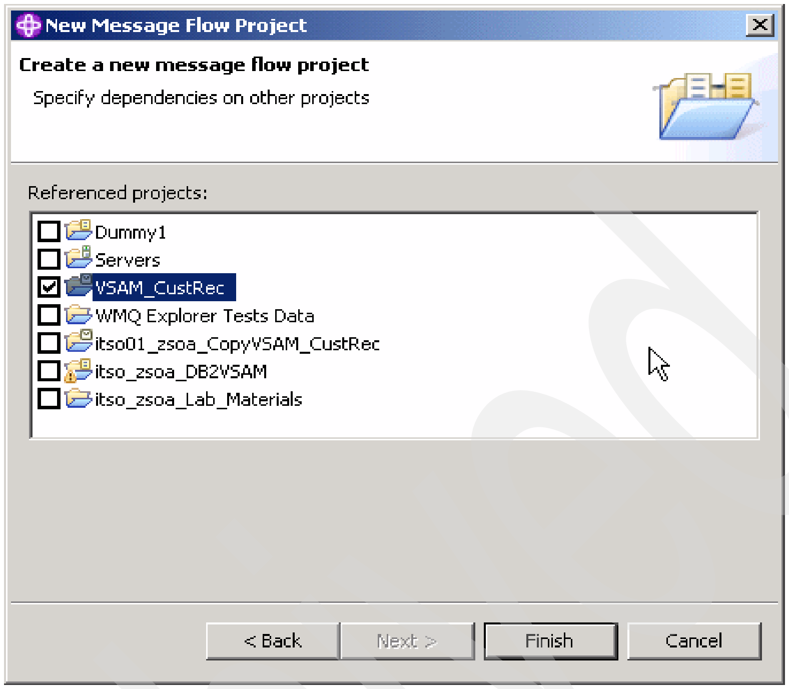
Defining the flow of the message

Having defined the VSAM message in the previous step, we will now begin to create the actual flow. Note that WebSphere Message Broker does not require that all messages be pre-defined in order to work with them. We will work with non-predefined XML messages and the pre-defined VSAM message. The incoming and outgoing XML messages used in this flow are not pre-defined. This will be seen in the ESQL code in later steps.

1. In the Broker Development pane, click with the right mouse button in any area and select **New** → **Message Flow Project**. This starts the Message Flow Project wizard. Give it a project name and Select **Next** to proceed to the dependency window.

   Note: The project name is only local in this case, so any name may be used. Typically, however, it is advisable to choose a name that indicates the nature of the flow within the project.

2. The project dependency window allows you to specify any other projects which may be referenced by your flow. This can enable the various flow editors to better assist you in pre-selecting options and verifying that necessary elements are available. You should select the **VSAM_CustRec** project message definition as a dependency. This flow makes use of this project with the imported VSAM data structure. Press **Finish** to complete this wizard. See Figure 7-9 on page 311. An empty project is created.


New Message Flow Project

Create a new message flow project

Specify dependencies on other projects

Referenced projects:

- [ ] Dummy1
- [ ] Servers
- [x] VSAM_CustRec
- [ ] WMQ Explorer Tests Data
- [ ] itso01_zsoa_CopyVSAM_CustRec
- [ ] itso_zsoa_DB2VSAM
- [ ] itso_zsoa_Lab_Materials

Figure 7-9 Choose the message definition project as a dependency for this message flow project

3. Expand the flow project just created in the Broker Development pane. Click the **Flows** heading and select **New** → **Message Flow** to start the message flow wizard. The wizard window appears and we specify the Message flow name.

Note: The combination of the schema name and flow name must be unique in the Broker Execution Group at runtime.

4. Click **Finish** and the message flow editor window should open with the node Palette and an empty flow canvas.

Note: You can now begin to draw the flow. In order to draw the flow, you must use the palette on the left side of the flow editor pane. As with most Eclipse-based flow drawing interfaces, you proceed as follows for each node.

Click on a folder, for example **WebSphere MQ** or **Routing**, to display the nodes within that folder. Click on a node within the folder, for example the **MQInput node** in the **WebSphere MQ** folder, to select the node. Afterwards, move the cursor to an empty area of the canvas, and click once with the left mouse button to place the chosen node on the canvas.

The name of the node is now high-lighted. You can retype the name of the node.

5. Figure 7-10 on page 313 and Table 7-1 on page 313 show how the flow has to be composed. Connect all of the nodes using the image to guide you. If you are not sure which terminal to use, check in the table. We next customize the various nodes as we explain the purpose of each node.

Note: Figure 7-10 on page 313 will be referenced several times in the upcoming sections. It will be convenient to bookmark this page if you are reading this material as a hard copy or note down the page if you reading this as PDF. Also we will mostly refer to the annotated form (A, B, C and so forth) of the nodes because we found it more convenient to quickly reference, in the later sections.

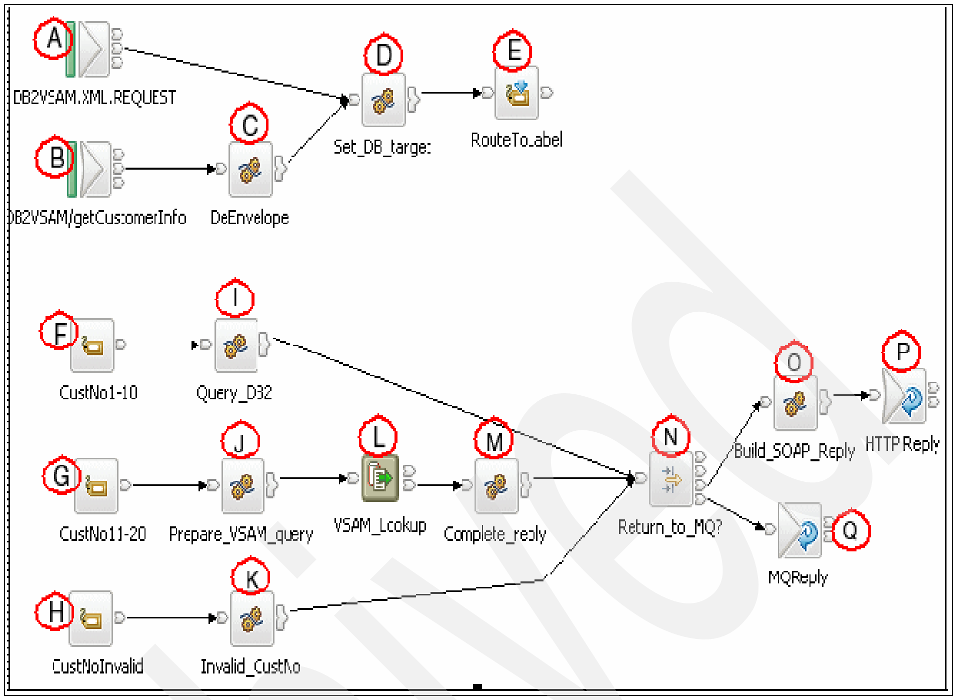


Figure 7-10 Annotated DB2VSAM flow

Table 7-1 DB2VSAM flow nodes

Node	In folder	Node Type	Rename to	Outgoing terminal
A	WebSphere MQ	MQInput	DB2VSAM.XML.REQUEST	Out
B	HTTP	HTTPInput	DB2VSAM/getCustomerInfo	Out
C	Transformation	Compute	DeEnvelope	Out
D	Transformation	Compute	Set_DB_target	Out
E	Routing	RouteToLabel	RouteToLabel (default)	(none)
F	Routing	Label	CustNo1-10	Out
G	Routing	Label	CustNo11-20	Out
H	Routing	Label	CustNoInvalid	Out
I	Transformation	Compute	Query_DB2	Out
J	Transformation	Compute	Prepare_VSAM_query	Out
K	Transformation	Compute	Invalid_CustNo	Out

Node	In folder	Node Type	Rename to	Outgoing terminal
L	Additional IBM VSAM nodes	VSAMRead	VSAM_Lookup	Out
M	Transformation	Compute	Complete_reply	Out
N	Routing	Filter	Return_to_MQ?	False to **Build_SOAP_Reply** True to **MQReply**
O	Transformation	Compute	Build_SOAP_Reply	Out
P	HTTP	HTTPReply	HTTPReply (default)	
Q	WebSphere MQ	MQReply	MQReply (default)	

Note: Though we are building this flow in the routing section of this chapter, this same flow will be used for illustrating the protocol conversion and data transformations.

Node descriptions and customizations

Next we explain all of the above node descriptions and perform certain customizations and explain them.

- The MQInput node labeled DB2VSAM.XML.REQUEST is the MQInput node where an incoming MQ (or JMS) message arrives. We will modify a few of this node's properties.

  a. Select the **DB2VSAM.XML.REQUEST** node by clicking it. Below the flow editor window, in the lower right pane, you should see the Properties tab displayed with this particular node's properties. Be sure that the Basic properties *side tab* is selected.

  b. Fill in the Queue name field with the request queue that was created on the z/OS MQ queue manager. For our example, we used ZSOA.DB2VSAM.XML.CUST.REQUEST.

  Note: The Message Broker is built upon WebSphere MQ. The Queue referenced here is the Queue where the initial message is posted and is available as MQ objects inside WebSphere MQ. All the Queue definitions and MQ objects were created using the script Define_Queue_Script.txt supplied as additional material in this book.

  c. Next click the side tab just below marked Input Message Parsing. In the Message domain drop-down box, select **XMLNS**. This tells the Broker that by default it should parse the incoming message with an XML parser that supports namespaces. Note that WebSphere Message Broker also includes other XML parsers, for instance the MRM XML parser which requires that the message be predefined with a schema. For simplicity, we chose to use the simpler XMLNS parser.

- The HTTPInput node labeled DB2VSAM/getCustomerInfo is where the incoming HTTP message arrives.

> **Note:** WMB has a very basic HTTP listener that can listen for incoming requests.

  a. Make sure that the lower Properties tab is pressed and you see the Basic side tab, and fill in the URL suffix that the Broker monitors for incoming HTTP requests. For our example the URL is /db2vsam/getCustomerInfo.

  b. Click the side tab just below labeled **Input Message Parsing**. Similar to what we did previously with the MQInput node, in the Message domain drop-down box, select **XMLNS.**

- All Compute nodes. *Compute nodes* (as well as JavaCompute nodes) contain pieces of code. Although there are many graphical nodes available in the WebSphere Message Broker, using Compute nodes with small programs can be quite efficient. And there is also the advantage that it is easier (and faster) to describe than scripting graphical "drag 'n drop" logic. For this flow, as in the previous flows, the small ESQL programs have been supplied in a file called DB2VSAM_flow.esql.txt, included in the additional material for this book. We will now "import" all of the supplied ESQL code into our flow.

  a. Double-click the first Compute node labeled **DeEnvelope** (alternatively, you can right-click and select **Open ESQL**). This should bring up the ESQL editor in the upper right panel.

  b. Carefully select all of the lines in this window except for the first line, which identifies the schema. Paste over all the selected lines from the file DB2VSAM_flow.esql.txt, included in the additional material (Ctrl-v, or right-click and select **Paste**) into this editor.

  c. Save the ESQL code flow. You probably will notice when you do this that the Outline pane in the lower left is filled in with the various ESQL module names included in the ESQL file because the Toolkit detects their presence.

  The code for the DeEnvelope Compute node is shown in Example 7-3.

Example 7-3 DB2VSAM DeEnvelope code

```
CREATE COMPUTE MODULE itso_zsoa_DB2VSAM_DeEnvelope
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        /* Remove the SOAP envelope and place customer request id in "standard" place */
        CALL CopyMessageHeaders();
        SET OutputRoot.XMLNS.CustomerReq.CustNo  =
            InputRoot.XMLNS.*:Envelope.*:Body.*:CustomerReq.*:CustNo;
    END;

    CREATE PROCEDURE CopyMessageHeaders() BEGIN
        DECLARE I INTEGER 1;
        DECLARE J INTEGER;
        SET J = CARDINALITY(InputRoot.*[]);
        WHILE I < J DO
            SET OutputRoot.*[I] = InputRoot.*[I];
            SET I = I + 1;
        END WHILE;
    END;
END MODULE;
```

Although almost 20 lines of ESQL code appear, in fact, only one line was written. The lower procedure, named CopyMessageHeaders(), is a standard procedure supplied automatically by the ESQL editor.

The only ESQL line actually written (and supplied for you in this exercise) is:

```
SET OutputRoot.XMLNS.CustomerReq.CustNo  =
InputRoot.XMLNS.*:Envelope.*:Body.*:CustomerReq.*:CustNo;
```

This code is executed after the message has been received by the HTTPInput node. The incoming message expected is SOAP over HTTP. The incoming HTTP SOAP data should be of the format shown in Example 7-4.

Example 7-4   DB2VSAM incoming SOAP HTTP message

```
<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">
   <soapenv:Body>
      <Customer>
         <CustNo>8</CustNo>
      </Customer>
   </soapenv:Body>
</soapenv:Envelope>
```

The line of ESQL code above removes all of the namespace references and creates a simplified version of the message with only the `<Customer>` element. This is done to “normalize” the subsequent processing because it now resembles the data format expected in the MQ node. Example 7-1 on page 301 showed the application part of the input message.

- The Set_DB_target node: You can display the ESQL code associated with this node by performing the same steps as above. Click the tab **itso_zsoa_DB2VSAM.msgflow**, and then double-click the node named **Set_DB_target.**

The only code developed follows:

```
SET
OutputLocalEnvironment.Destination.RouterList.DestinationData[1].lab
elname =
      CASE
         WHEN InputRoot.XMLNS.CustomerReq.CustNo < 11 THEN 'do_db2'
         WHEN InputRoot.XMLNS.CustomerReq.CustNo < 20 THEN 'do_vsam'
         ELSE 'do_invalid'
      END;
```

The code associated with this node performs the heart of the “routing” function of this mediation. It inspects the incoming data (the CustNo field) and based upon its value, it sets up the label that the flow will branch to in the subsequent RouteToLabel node. This sequence of nodes performs the classic “select” logic found in most second generation programming languages.

In addition to the bit of ESQL code, there is a small configuration change that is necessary for the Set_DB_target node:

a. Return to the message flow editor window tab by clicking the tab **itso_zsoa_DB2VSAM.msgflow**, and click the **Set_DB_target** node once. In the lower Properties page, make sure that the **Basic** tab is selected as the left tab. Select **LocalEnvironment and Message** as the Compute mode. This is necessary to ensure that this node passes to the following node the local environment data which is additional data carried in the flow along with the message. This is set in the ESQL code displayed above.

- The RouteToLabel node: This node simply performs a branch to the label set in the prior Set_DB_target node.
- The CustNo1-10 node: Select the node and make sure that you have the **Properties** tab selected, as well as the **Basic** left tab. This is the branch point when the incoming request was for a customer number 1 to 10, in which case the flow should query DB2.

  a. Type the value `do_db2` in the branch Label name field.

  b. Perform the same customization on the other two label nodes. The CustNo11-20 node should have the branch Label name do_vsam.

  c. The CustNoInvalid node should have the branch Label name **d**o_invalid.
- The Query_DB2 node: It contains perhaps the most complex code in this flow.

Example 7-5 ESQL for Query_DB2 node

```
SET OutputRoot.XMLNS.CustomerReply[ ] = (SELECT
         A.CUST_NO AS CustNo, A.CUST_LN AS LastName, A.CUST_FN AS 
FirstName,
         A.CUST_ADDR1 AS Address1, A.CUST_CITY AS City, A.CUST_ST AS 
State,
         A.CUST_CTRY AS Country FROM Database.ITSOOC.EOTCUST AS A
         WHERE A.CUST_NO = InputRoot.XMLNS.CustomerReq.CustNo);
      /* Verify if DB2 inquiry found the customer record */
      IF (Cardinality(OutputRoot.XMLNS.CustomerReply[ ]) = 1 ) THEN
         SET OutputRoot.XMLNS.CustomerReply.RetCode = 0;
      ELSE
         SET OutputRoot.XMLNS.CustomerReply.RetCode = 8;
      END IF;
```

The first line of code (SET) performs the actual database lookup with a SELECT statement that is virtually identical to standard SQL. This, of course, explains why the native Broker language is called ESQL (Extended SQL). The second line of code (IF) simply validates that reasonable data was returned from the DB2 SELECT and assigns an appropriate return code.

In addition to supplying the above ESQL code, it is necessary to do a small bit of configuration for this Compute node. Click the **Query_DB2** node in the message flow editor window. In the lower Properties page, make sure that the **Basic** tab is selected as the left tab.

a. The Data source field must be set to the DB2 location ID where the customer database is found. We set it to DB9H. This is the same database referenced in Chapter 3, "Accessing DB2 from WebSphere using Web Services" on page 85.

Note: The customer table definition (DDL) is provided in the file customer_table_create.sql.

- The Return_to_MQ? node: This node performs the role of determining whether the original request came via MQ or HTTP, and uses this to determine whether the reply should be formatted in XML over MQ or SOAP over HTTP:

```
IF (Root.Properties.ReplyProtocol = 'MQ') THEN
         RETURN TRUE;
      ELSE
         RETURN FALSE;
      END IF;
```

The code inspects a field associated with the original incoming message that indicates the protocol used.

- The True branch is taken if the original message came via MQ. In this case, the MQReply node is used.

  Important: This MQReply node uses standard MQ application protocol. It automatically sends an MQ reply message to the queue specified in the ReplyToQueue field of the original incoming MQ message. This normally is set by the application client that emits the original request message. The MQReply node requires no customization at all; its default settings can be used. It is very important that the application client set the ReplyToQueue field in the message. WMB will not process the flow execution if this field is not set.

- The False branch is taken otherwise, that is, when the original message arrived via HTTP. In this case, the return message must be transformed into a valid SOAP message format. This is done by the following Compute node labeled Build_SOAP_Reply.

► The Build_SOAP_Reply node: The following ESQL code has been supplied to do this transformation:

```
/* Declare the XML namespaces we will be using */
      DECLARE mySoapNS NAMESPACE
'http://schemas.xmlsoap.org/soap/envelope/';
      DECLARE myNS1 NAMESPACE 'http://itso.ibm.com';
      DECLARE myXSI NAMESPACE
'http://www.w3.org/2001/XMLSchema-instance';
      DECLARE myXSD NAMESPACE 'http://www.w3.org/2001/XMLSchema';

      /* Create a standard broker message Body for application
data with standard header */
      CREATE FIELD OutputRoot.XML;
      SET OutputRoot.XML.(XML.XmlDecl) = '';
      SET OutputRoot.XML.(XML.XmlDecl).(XML.Version) = '1.0';
      SET OutputRoot.XML.(XML.XmlDecl).(XML.Encoding) = 'UTF-8';

      /* Add the SOAP Envelope and Body and pointers  */
      CREATE FIELD OutputRoot.XML."soapenv:Envelope";
      Declare soapEnvelope REFERENCE TO
OutputRoot.XML."soapenv:Envelope";
      CREATE FIELD soapEnvelope."soapenv:Body";
      Declare soapBody REFERENCE TO soapEnvelope."soapenv:Body";

      /* Create the standard SOAP Envelope attributes */
      SET soapEnvelope.(XML.Attribute)"xmlns:soap" =
'http://schemas.xmlsoap.org/soap/envelope/' ;
```

```
      SET soapEnvelope.(XML.Attribute)"xmlns:soapenv"
='http://schemas.xmlsoap.org/soap/envelope/';
      SET soapEnvelope.(XML.Attribute)"xmlns:xsd" =
'http://www.w3.org/2001/XMLSchema' ;
      SET soapEnvelope.(XML.Attribute)"xmlns:xsi" =
'http://www.w3.org/2001/XMLSchema-instance' ;

      /* Add the application-specific data */
      CREATE FIELD soapBody.getCustomerReply;
      Declare getCustomerReply REFERENCE TO
soapBody.getCustomerReply;

      SET getCustomerReply = InputRoot.XMLNS.CustomerReply;
```

Although a bit more lengthy, this code is not very complicated. We will not explain it in detail, but it basically is adding the standard SOAP message headers to the application data.

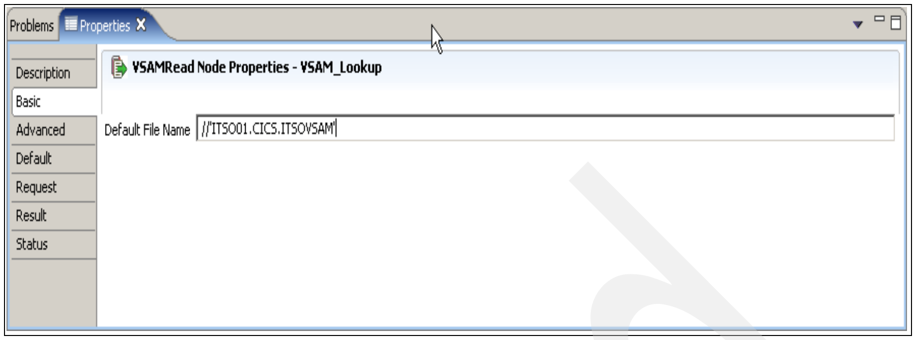
After the Compute node, the HTTPReply node is used to return the SOAP reply to the calling application. This node requires no customization at all.

- The Prepare_VSAM_query node: This is the case where the incoming client request specified a CustNo between 11 to 20 and the flow branches to the second Label node CustNo11-20. In this case, the Compute node labeled Prepare_VSAM_query is invoked, with the following ESQL code:

```
DECLARE Pattern CHARACTER '000';
      SET OutputRoot.XMLNS.VSAM.Request.Position.Key =
         CAST( CAST(InputRoot.XMLNS.CustomerReq.CustNo AS
INTEGER) AS CHARACTER FORMAT Pattern);
      RETURN TRUE;
```

This code picks out the incoming CustNo field, formats it with leading zeros, and places it in the field expected by the following VSAMRead node.

- The VSAM_Lookup node: For this node we need to specify where the VSAM data set is. In the lower Properties page, make sure that the **Basic** tab is selected as the left tab, as shown in Figure 7-11.



Default File Name //'ITSO01.CICS.ITSOVSAM'

Figure 7-11 Configure VSAM file

You must set the VSAM File name, which in our case is //'ITSO.CICS.ITSOVSAM' (single quotes included). Select the Default left tab now. You must indicate here the message format to allow the Broker to parse the VSAM data record.

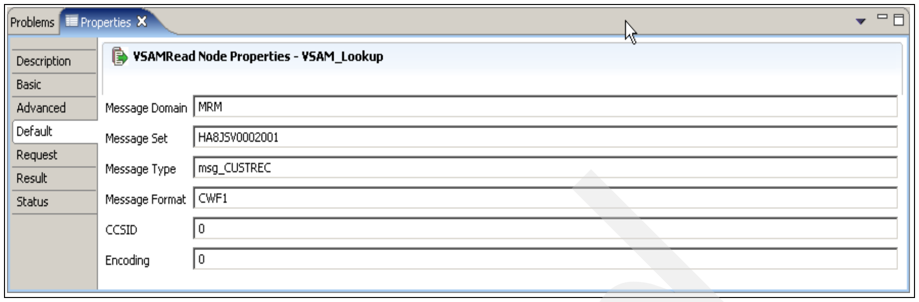
See Figure 7-12 on page 322, for defining the values.

– Set the Message Domain to MRM. This stands for the *Message Repository Manager* and refers to the library and format used to store WebSphere Message Broker message formats.

– Type the Message Set identifier. This is a unique value for every machine. Fill in the value of your message set as seen in Figure 7-6 on page 307 (or better yet, to avoid potential typing errors, re-open the message set definition and copy/paste this field).

– Type `msg_CUSTREC` for the Message Type. This is the name of the specific message in our message set. This was the name on the original COBOL COPYBOOK that was used by the importer.

– Type `CWF1` as the Message Format. This is the particular type of *physical binding* with the logical message that is to be used. The WebSphere Message Broker can associate multiple physical bindings (XML, C structure, delimited, and so on) with one logical message definition. In this case, only one physical binding was created at import time.


Problems | Properties

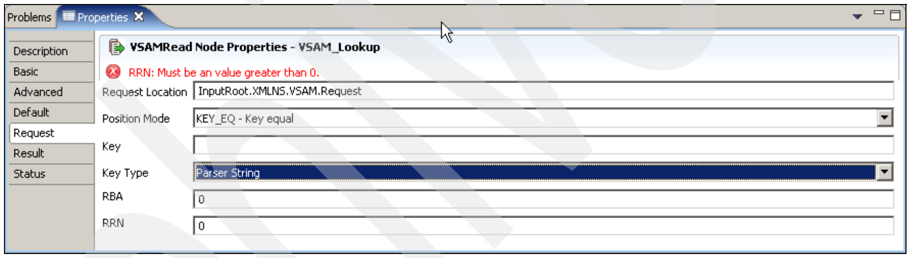
Description
Basic
Advanced
Default
Request
Result
Status

VSAMRead Node Properties - VSAM_Lookup

Message Domain	MRM
Message Set	HA8J5V0002001
Message Type	msg_CUSTREC
Message Format	CWF1
CCSID	0
Encoding	0

Figure 7-12   Configuring the VSAM data set

Select the **Request** left tab now. In this pane, as shown in Figure 7-13, you specify the default VSAM data search and access parameters here (note that all these parameters could equally be passed as dynamic parameters, but it is easier for this example to use static node parameters). Use the information shown in Figure 7-13 to fill in the values.


Problems | Properties

Description
Basic
Advanced
Default
Request
Result
Status

VSAMRead Node Properties - VSAM_Lookup

RRN: Must be an value greater than 0.

Request Location	InputRoot.XMLNS.VSAM.Request
Position Mode	KEY_EQ - Key equal
Key	
Key Type	Parser String
RBA	0
RRN	0

Figure 7-13   VSAM Search and access parameters

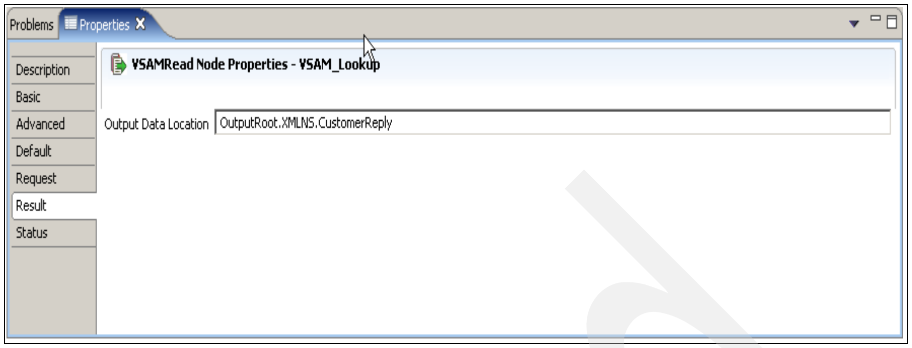
– Type `InputRoot.XMLNS.VSAM.Request` as the Request Location. This must be set to the name of the element where the VSAM search key has been placed.

– Select **KEY_EQ** as the Position Mode. This tells VSAM the type of key search to perform. We want to search for the record with the key equal to the parameter.

– Set the Key Type parameter to Parser String to indicate that the key is character.

  Note that an error message appears on this menu, `X RRN Must be....` This message can safely be ignored as we are not working with VSAM Relative Record Numbers.

Select the **Result** left tab now. You specify here where the VSAM data is to be returned. Use Figure 7-14 to fill in the values.



VSAMRead Node Properties - VSAM_Lookup

Output Data Location OutputRoot.XMLNS.CustomerReply

Figure 7-14 VSAM node result parameter

– For the Output Data Location, type in **`OutputRoot.XMLNS.CustomerReply`**. This is the position in the message where the parsed data should be placed after reading it from VSAM.

Select the **Status** left tab now. You specify here record report return. For the Include Report Record, select **Yes**.

► The Complete_reply node. Double-click the node to show its ESQL code. This node uses the following supplied code:

```
IF (InputLocalEnvironment.VSAM.Report.Length > 0 ) THEN
         SET OutputRoot.XMLNS.CustomerReply.RetCode = 0;
      ELSE
         SET OutputRoot.XMLNS.CustomerReply.RetCode = 8;
      END IF;
      RETURN TRUE;
```

This code performs a simple verification if the VSAM search returned valid data and sets a return code appropriately. After this Compute node, the code joins the same Filter node described above to determine if an MQ or HTTP message should be sent as a reply.

The third and final case is where the incoming client request specifies an invalid customer number. In this case, the label `CustNoInvalid` is entered.

► The Invalid_CustNo node: `SET OutputRoot.XMLNS.CustomerReply.RetCode = 12`; a simple return code is set.

The flow is now complete. Be sure to save the flow at this point by selecting the flow editor tab and pressing Ctrl-s. In the lower pane labeled Problems, several (9) warning messages concerning `Unresolvable database table reference` appear.

Deploying and testing the DB2 VSAM flow

In this section we deploy the application and test the following flow. We create a Broker Archive (.bar) file with the artifacts, deploy it to the Broker runtime and test it.

Note: A WebSphere Message Broker uses the .bar file in the same way that WebSphere Application Server uses .war or .ear files—it is a simple file that contains all the necessary components in a compressed, unified format for easy deploying to a runtime environment.

Select the **Broker Administration** perspective in the WMB Toolkit in the upper right part of the WMB Toolkit. In the Broker Administration pane found in the upper left, navigate to the **Servers** folder in the Broker Archives. Right-click it and select **New** → **Message Broker Archive**.

This should bring up the Broker Archive creation window. Any name can be used when filling in the name of a .bar file to create. The .bar file typically has a name that indicates the project. We suggest that you use the name `DB2VSAM.bar` here. The .bar files also have Deployment Descriptors and can be customized for an environment. Press **Finish.**

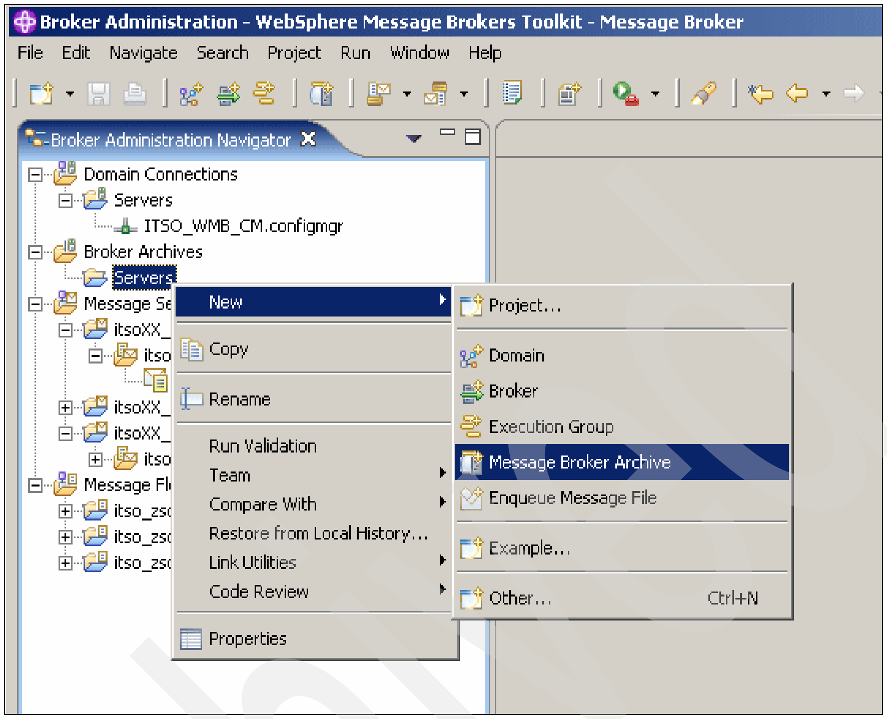
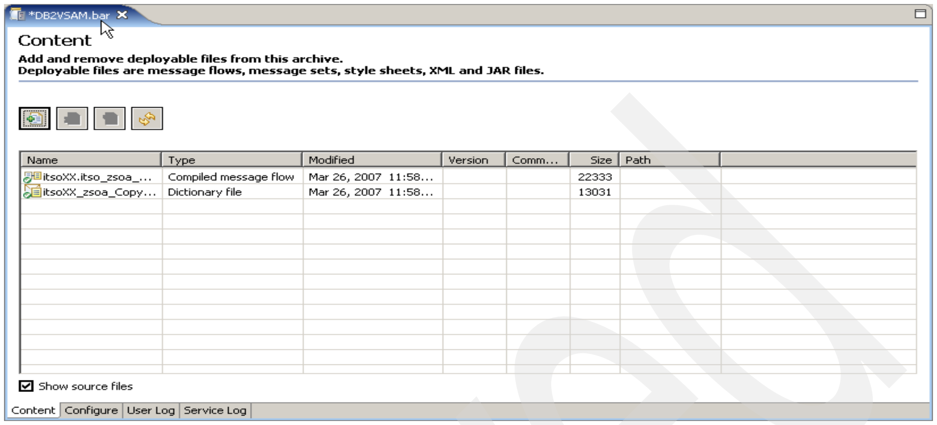


Figure 7-15 Start the making of the .bar file

In the .bar editor window, press the .bar Add button to bring up the Broker Archive selection window. A .bar file should contain any message sets and flows that need to be deployed to a runtime Broker in order for the flow to execute.

Select the project containing the message set and the project containing the flow we developed. Press **OK** to continue.

A confirmation window is displayed; press **OK** to dismiss the window, and the .bar editor should display the artifact names, as shown in Figure 7-16.


*DB2VSAM.bar
Content
Add and remove deployable files from this archive.
Deployable files are message flows, message sets, style sheets, XML and JAR files.

Name	Type	Modified	Version	Comm...	Size	Path
itsoXX.itso_zsoa_...	Compiled message flow	Mar 26, 2007 11:58...			22333	
itsoXX_zsoa_Copy...	Dictionary file	Mar 26, 2007 11:58...			13031	

☑ Show source files
Content
Configure
User Log
Service Log

Figure 7-16 Adding .bar files

Press Ctrl-s to save the .bar file.

You can now deploy the .bar to the Broker executing on z/OS. First, however, you need to connect the WMB Toolkit to the Configuration Manager on z/OS. This connection only needs to be done once each time you start the WMB Toolkit.

The Configuration Manager controls and filters access to the Broker. All artifacts are first verified by the Configuration Manager. The Configuration Manager domain has Broker name and Broker execution groups. We deploy the .bar file into the Broker execution group.

We can now test our new flow!

Refer to the Message flow in Figure 7-10 on page 313. The first message flow path we will test is: **Incoming HTTP request** → **B** → **D** → **E** → **F** → **I** → **N** → **O** → **P**.

Note: The alphabets in the above flow represent the alphabetical annotation in Figure 7-10 on page 313.

We will send the following XML message through HTTP through a standalone Java main program[1]. Recall that we configured the HTTPInput node A to listen on HTTP requests in our message flow. See Example 7-6 on page 327.

Example 7-6 Input soap message to the Java program

```
<!-- Example of SOAP request for testing -->
<!-- http://wtsc47.itso.ibm.com:7080/db2vsam/getCustomerInfo -->
<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">
   <soapenv:Body>
      <CustomerReq>
         <CustNo>8</CustNo>
      </CustomerReq>
   </soapenv:Body>
</soapenv:Envelope>
```

The Java program source is provided as additional material.

Example 7-7 Output from the JAVA program

```
HTTP/1.1 200 OK
Content-Type: text/xml;charset=utf-8
Content-Length: 537
Date: Tue, 02 Oct 2007 17:21:44 GMT
Server: Apache-Coyote/1.1
Connection: close

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/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"><soapenv:Body><Cu
stomerReply><CustNo>8</CustNo><LastName>Farkas</LastName><FirstName>Car
l</FirstName><Address1>1 rue de la
Paix</Address1><City>Paris</City><State>DB2</State><Country>FRANCE</Cou
ntry><RetCode>0</RetCode></CustomerReply></soapenv:Body></soapenv:Envel
ope>
```

Next we use the same program. But this time we send in a Customer number of 11. The flow this data will cause to execute is:

Incoming HTTP request → B → C → D → E → G → J → L → M → N → O → P

[1] We could have used a browser to test this but we chose to use a Java program to illustrate that browsers would not typically communicate directly with a WMB HTTPInputNode, but internal programs would be the source of the message.

In the same Java program we changed the tag to:

```
<CustNo>11</CustNo>
```

Example 7-8 Output Data from VSAM

```
HTTP/1.1 200 OK
Content-Type: text/xml;charset=utf-8
Content-Length: 535
Date: Tue, 02 Oct 2007 17:32:43 GMT
Server: Apache-Coyote/1.1
Connection: close

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/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"><soapenv:Body><Cu
stomerReply><CustNo>11</CustNo><LastName>Kho</LastName><FirstName>Wilbe
rt</FirstName><Address1>99 Ocean Drive</Address1><City>San
Jose</City><State>CA</State><Country>USA</Country><RetCode>0</RetCode><
/CustomerReply></soapenv:Body></soapenv:Envelope>
```

This illustrates the power of the WMB environment, wherein the same input message was routed to different interfaces (DB2 and VSAM) based on the input data.

The invalid customer number flow is fairly trivial. We send in a Customer number that is above 20 through the same Java program. We see the output shown in Example 7-9.

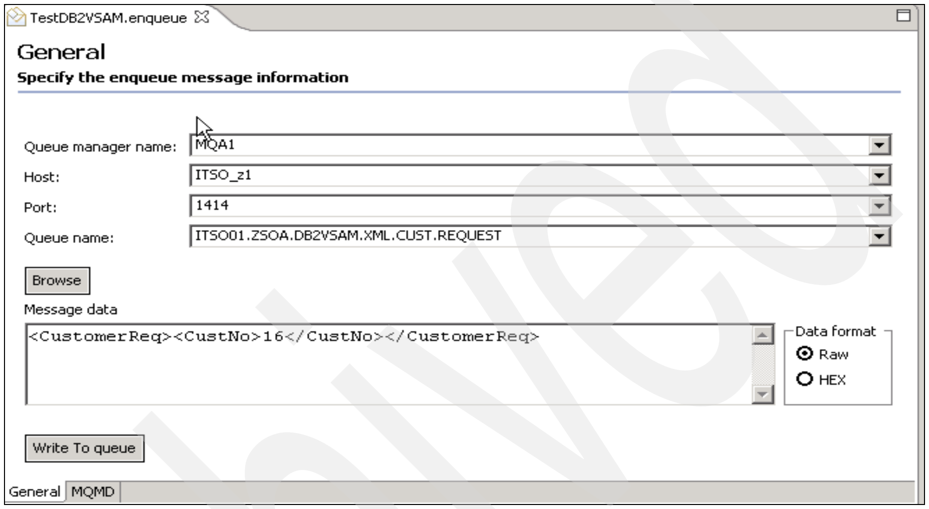
Example 7-9 Output from sending a Customer number greater than 20

```
HTTP/1.1 200 OK
Content-Type: text/xml;charset=utf-8
Content-Length: 368
Date: Tue, 02 Oct 2007 17:41:42 GMT
Server: Apache-Coyote/1.1
Connection: close

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/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"><soapenv:Body><Cu
```

```
stomerReply><RetCode>12</RetCode></CustomerReply></soapenv:Body></soape
nv:Envelope>
```

Enqueue and dequeue utility

WMB has facilities to put and get messages from a queue. The user interface is shown in Figure 7-17.



Queue manager name: MQA1
Host: ITSO_z1
Port: 1414
Queue name: ITSO01.ZSOA.DB2VSAM.XML.CUST.REQUEST
Message data
```
<CustomerReq><CustNo>16</CustNo></CustomerReq>
```
Data format
[✓ Raw]
HEX

Figure 7-17 Enqueue

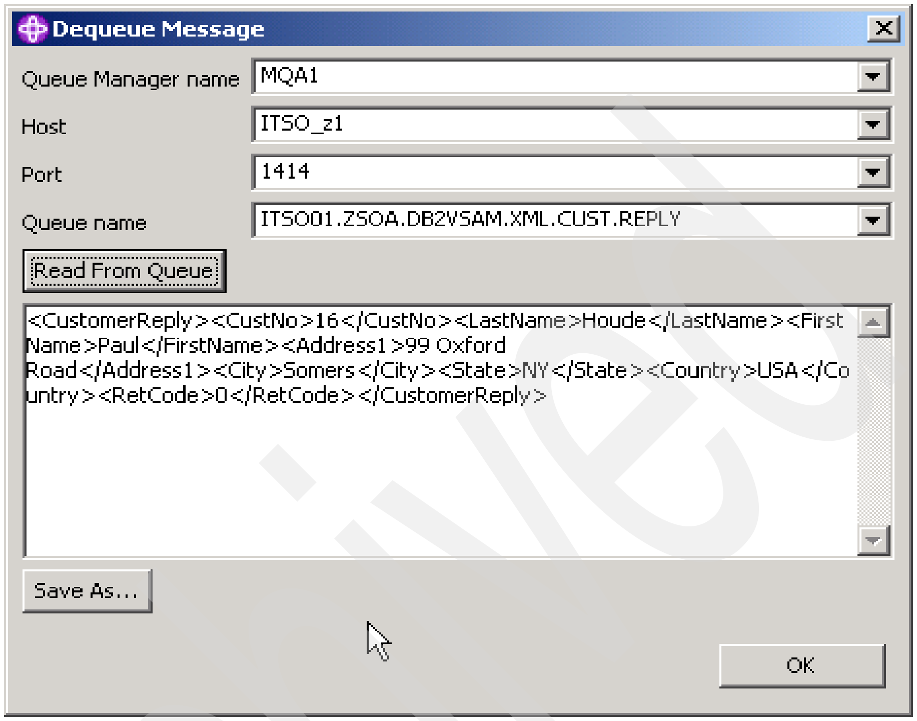
Note: We have supplied a sample enqueue file. You may skip these steps and directly import the file into the tool.

We will configure an enqueue file so that you can use it to send a test message:

1. Switch to the Broker Administration perspective.
2. On the workbench toolbar, click the arrow on the **Put a message onto a queue** icon.
3. On the drop-down menu, click **Put Message** to invoke the New Enqueue Message File wizard.
4. Select the message flow project containing the message flow that you are debugging.
5. In the File name field, enter a name for the file to create (the extension .enqueue is added automatically).
6. Click **Finish**. The enqueue file is created, and a view opens showing its details.

7. Enter the names for the queue manager and the queue for the input node for this flow. Queue manager names are case-sensitive; make sure that you enter the name correctly.
8. If you are putting a message on an input queue that is on a remote computer, ensure that the queue manager of the associated Broker has a server-connection channel called SYSTEM.BKR.CONFIG.
9. If you are putting a message on a remote queue, enter values to identify the host and port of the computer that is hosting the queue.
10. Click the **MQMD** tab to customize the fields of the MQMD header. Specify a value in the reply queue field.
11. Click **File** → **Save** to save the enqueue file.
12. Optional: To put the message to the queue immediately from this window, click **Write to queue**.
13. Click the arrow on the **Put a message onto a queue icon** to see your enqueue file listed on the drop-down menu.

Similarly, you can configure a dequeue message to see the message in the queue from the Broker Development perspective. See Figure 7-18.



Queue Manager name	MQA1
Host	ITSO_z1
Port	1414
Queue name	ITSO01.ZSOA.DB2VSAM.XML.CUST.REPLY

```
<CustomerReply><CustNo>16</CustNo><LastName>Houde</LastName><First
Name>Paul</FirstName><Address1>99 Oxford
Road</Address1><City>Somers</City><State>NY</State><Country>USA</Co
untry><RetCode>0</RetCode></CustomerReply>
```

Figure 7-18 Dequeue message

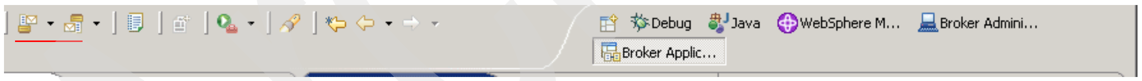


Figure 7-19 Put and get icons in the Broker Development perspective

Alternatively you may also use the RFHUTIL utility. This is a free utility that is available as a WebSphere MQ support pack and can be downloaded from the IBM support software download site at:

http://www-1.ibm.com/support/docview.wss?rs=203&uid=swg24000637&loc=en_US&cs=utf-8&lang=en

7.2.2 Protocol transformation and data transformation

While the goal of the scenario discussed in 7.2.1, “Routing” on page 300 was to demonstrate routing, the scenario discussed in this section focuses on protocol transformation and data transformation, both two key areas of functionality in WebSphere Message Broker.

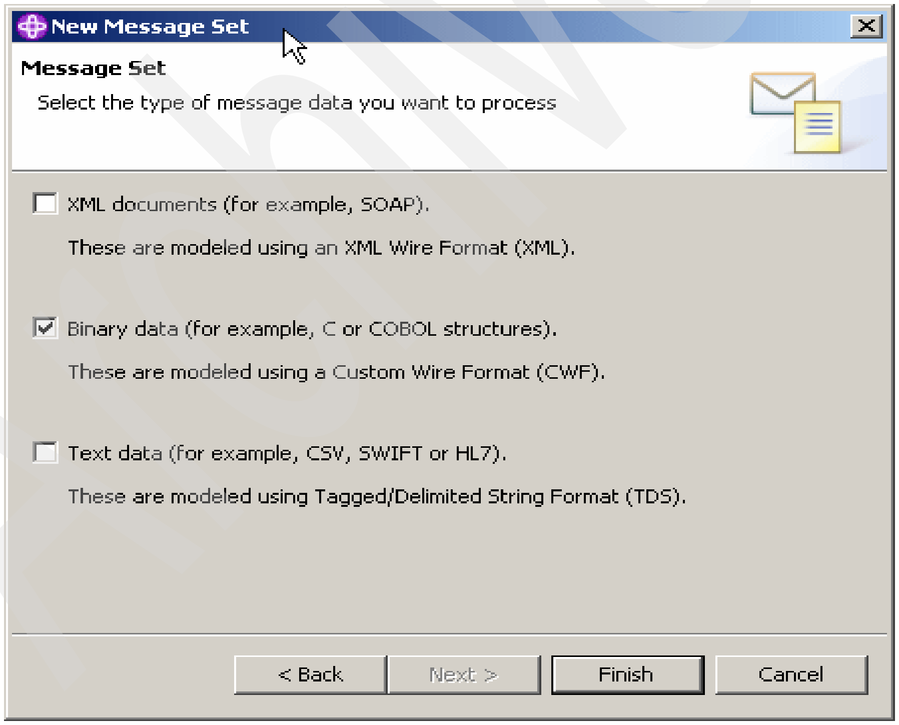
We import a couple of message sets and build and design a message flow to illustrate this scenario. WMB will interface with a CICS back-end program using the EXCI protocol. A specific node type exists in WMB for EXCI.

Building two new message sets

As in "Defining the format of the message" on page 303, we start with defining two new message sets from the following sources (included in the additional material):

- COBOL file: COMAREA.cpy
- XML files: CustomerInfoRequest.xsd and CustomerInfoReply.xsd

Import these files into two separate new message set projects. For the COBOL file make sure you choose **Binary data** in the New Message Set pane, as shown in Figure 7-20.


New Message Set

Message Set

Select the type of message data you want to process

[ ] XML documents (for example, SOAP).

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

[✓] Binary data (for example, C or COBOL structures).

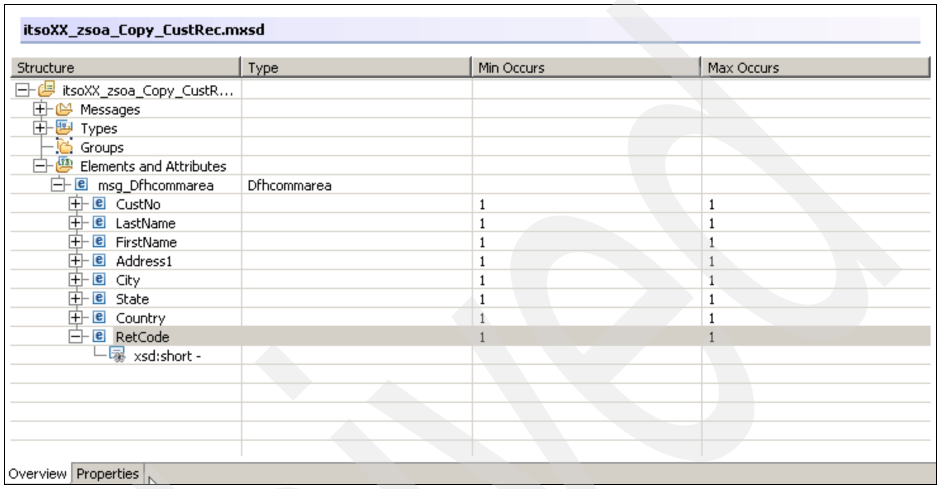
These are modeled using a Custom Wire Format (CWF).

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

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

Figure 7-20 Binary data choice for new message set

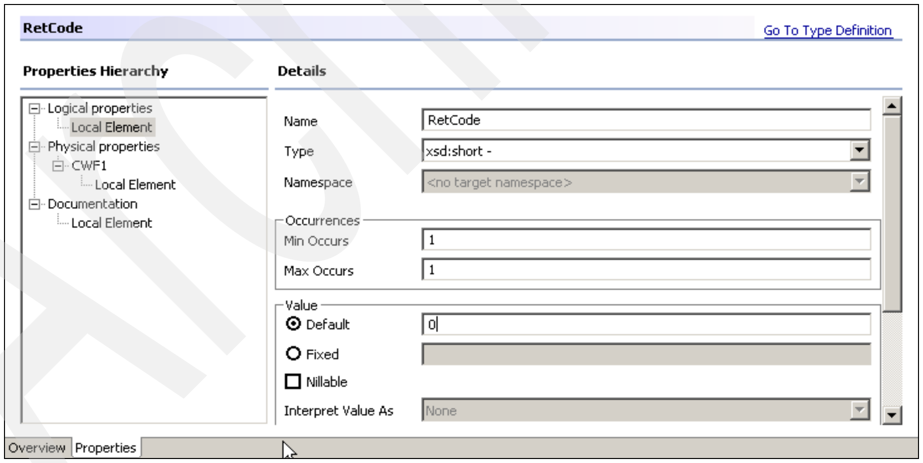
Also note down the message set ID as before. Import the COBOL structure just like we did in “Defining the format of the message” on page 303. There is one unique customization we have to do here. As we will be invoking a CICS program through EXCI, this message structure is used as a COMMAREA passed to this CICS program. It requires a return code to be initialized. See Figure 7-21.



itsoXX_zsoa_Copy_CustRec.mxsd

Structure	Type	Min Occurs	Max Occurs
itsoXX_zsoa_Copy_CustR...			
Messages			
Types			
Groups			
Elements and Attributes			
msg_Dfhcommarea	Dfhcommarea		
CustNo		1	1
LastName		1	1
FirstName		1	1
Address1		1	1
City		1	1
State		1	1
Country		1	1
RetCode		1	1
xsd:short -			

Figure 7-21 Change property of the return code



RetCode

Name: RetCode
Type: xsd:short -
Namespace: <no target namespace>
Occurrences
Min Occurs: 1
Max Occurs: 1
Value
[✓ Default]: 0
Fixed
Nillable
Interpret Value As: None

Figure 7-22 Set default value

Following steps similar to the ones in “Defining the format of the message” on page 303, create a new message set project for importing the XML files CustomerInfoRequest.xsd and CustomerInfoReply.xsd.

Make sure you choose XML documents in the “New Message set” window. You will have two new message set projects.

Building a new message flow

We will design a message flow that handles two different consumers. We chose XML/MQ and SOAP/HTTP as the consumer data format/protocol. The flow demonstrates protocol conversion, data transformation, and content-based routing.

The flow accepts the same inputs as the scenario in the previous section (MQInput or SOAP over HTTP).

Depending upon the nature of the input data:

- Two Compute nodes perform data transformation to a COBOL COPYBOOK format for executing the CICS transaction.
- The CICS transaction is executed using the EXCI interface. It sends the response in a COBOL COPYBOOK format (this is an example of protocol conversion).
- The next node is a Filter node that decides on the original consumer (MQ or HTTP). Based on the requester, the Filter node routes the response (this is an example of content-based routing[2]).
- The two compute nodes perform a reverse transformation from COBOL COPYBOOK to either XML or SOAP based on the path it takes.
- The response message is either written to ReplyToQueue or HTTPReply.

An example of the XML reply message written to MQ is shown here:

```
<CustomerReply>
   <CustNo>9</CustNo>
   <LastName>Maitra</LastName>
   <FirstName>Subhajit</FirstName>
   <Address1>1 Financial Plaza</Address1>
   <City>Hartford</City><State>CT</State>
   <Country>USA</Country>
   <RetCode>0</RetCode>
</CustomerReply>
```

An example of the SOAP/HTTP reply message is shown here:

```
<?xml version="1.0" encoding="UTF-8" ?>
 <soapenv:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
```

[2] This concept was illustrated in the “routing” scenario also.

```
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
       <soapenv:Body>
         <CustomerReply>
            <CustNo>9</CustNo>
            <LastName>Maitra</LastName>
            <FirstName>Subhajit</FirstName>
            <Address1>1 Financial Plaza</Address1>
            <City>Hartford</City>
            <State>CT</State>
            <Country>USA</Country>
            <RetCode>0</RetCode>
        </CustomerReply>
       </soapenv:Body>
     </soapenv:Envelope>
```

The entire flow, once it is ready, is shown in Figure 7-23.

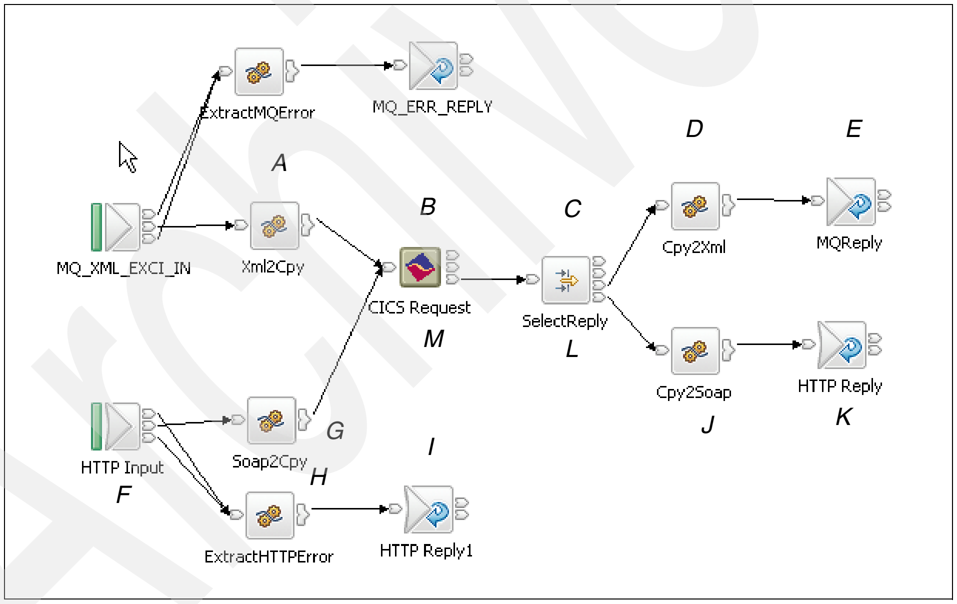


Figure 7-23 Message flow for invoking a CICS program through EXCI

This flow is similar to the earlier flow with the DB2 and VSAM calls, but notice that both the starting MQInput node (labeled MQ_XML_EXCI_IN) and the HTTPInput node now have their upper Failure and lower Catch terminals wired. This flow demonstrates the ability to “trap” potential errors in a flow and handle them more elegantly by having the flow send back the cause of an error message to the calling application. Build the flow as in the previous example and rename the nodes as shown in the figure. We have also included the project interchange file in the supplied material. The MQ flow is **A** → **B** → **C** → **D** → **E** and the HTTP flow is **F** → **G** → **M** → **L** → **J** → **K**. We list the customizations for each node here:

- MQ_XML_EXCI_IN node**:** Select **Basic** on the Properties page and enter the Input Queue name. This is the queue where WMB will be listening. In our example we entered `ITSO01.ZSOA.EXCI.XML.CUST.REQUEST`.

  We set the parsing attributes for the MQ_XML_EXCI_IN node, as follows:

  – Select **Input Message Parsing** on Properties.

  – Set Message Domain to MRM.

  – Set Message Set to the first message set we created in the section: “Building two new message sets” on page 332.

  – Set Message Type to CustomerReq.

  – Set Message Format to XML1.

  Connections to/from this node can be created using the following procedure:

  – Right-click the **Origin** node and select **Create Connection**.

  – Select the output terminal (we will supply which terminal to use here).

  – Click the target node to anchor the connection to the node.

  – Hover the mouse over the connection line to verify to/from terminals.

  Use the information in Table 7-2 to create the required connections.

Table 7-2 Connectors for CICSEXCI MQ

From/To	**Output terminal to use**
MQ_XML_EXCI_IN to A	Out
A to B	Out
B to C	Out
Third terminal on C to D	False
D to E	Out

- CICSRequest node: Select **Basic** on the Properties page. In our example we specified the following values:

  – `CITSO01` for the CICS Network Applid

  – `CUSTINQ` for then CICS Program Name

  – `ITSO01` for the CICS userid

- Check box **Use Message Location for Program Name**
- Check box **Use Message Location for CICS Userid**

Note: The CUSTINQ CICS program is the same program introduced in Chapter 2, “Accessing CICS from WebSphere using Web Services over HTTP” on page 3.

Connections to/from this node can be created using the following procedure:

- Right-click the **Origin** node and select **Create Connection**.
- Select the output terminal (we will supply which terminal to use here).
- Click the target node to anchor the connection to the node.
- Hover mouse over the connection line to verify to/from terminals.

Use the information in Table 7-3 to create the requited connections.

Table 7-3 Connectors for CICSEXCI HTTP

From/To	Output Terminal to use
First terminal on F to H connection	Failure
Second terminal on F to G connections	Out
Third terminal on F to H connections	Catch
H to I	Out
G to M	Out
Fourth terminal on L to J	True
J to K	Out

- The HTTP Input node: Select **Basic** on the Properties page and specify the URL. We used `/exci/getCustomerInfo` in our example.

Note: The full URL is composed of a host name and port number, followed by the value specified here.

- Compute nodes: The ESQLs for the compute nodes are provided in esql.txt. Paste this into the nodes. Because this is custom code, we need to tell each compute node which module to use by setting a property value. Refer to Table 7-4 for details. Click **Basic Properties** and **Browse** to select the ESQL module.

Table 7-4 Compute node ESQL module selection

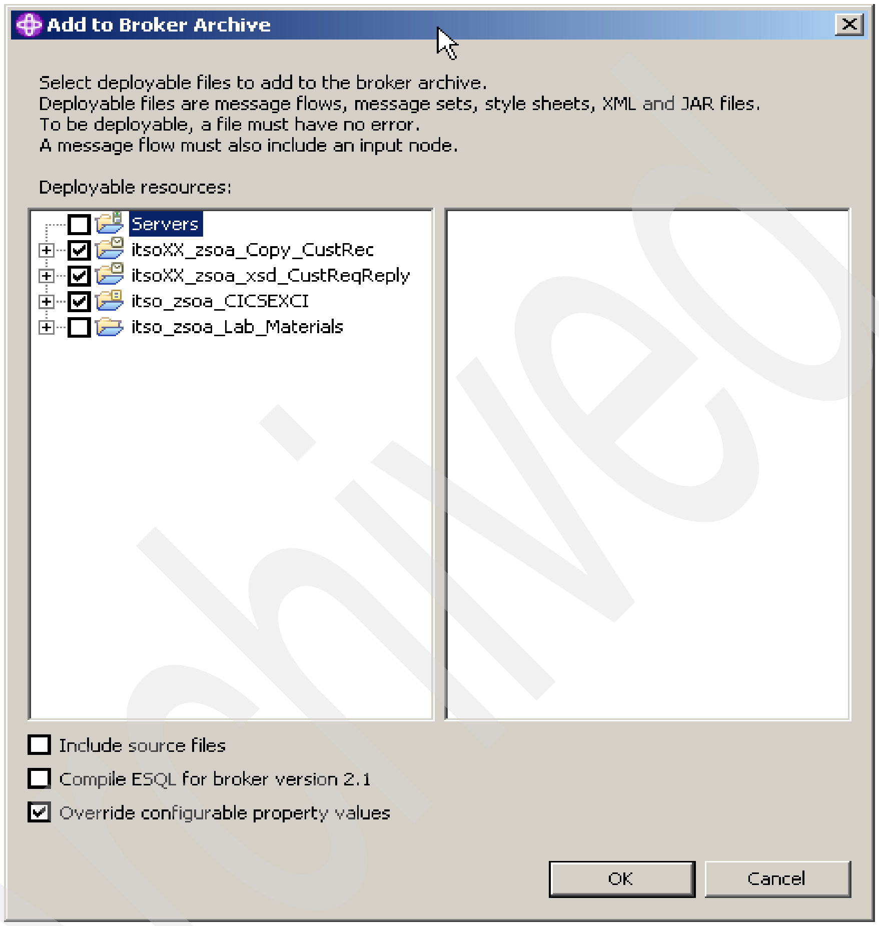
Compute Node	Module Name
xml2Cpy	itsoXX.itso_zsoa_CicsExci_xml2Cpy
Soap2Cpy	itsoXX.itso_zsoa_CicsExci_soap2Cpy
ExtractHTTPError	itsoXX.itso_zsoa_CicsExci_ExtractError
Cpy2Xml	itsoXX.itso_zsoa_CicsExci_cpy2Xml
Cpy2Soap	itsoXX.itso_zsoa_CicsExci_cpy2Soap
SelectReply	itsoXX.itso_zsoa_CicsExci_Filter_Reply

When the flow definitions are complete, the flow can be deployed and tested.

Note: CICS Request is the node where the protocol conversion takes place. The protocol conversion is transparent. Most other compute nodes perform message transformations from XML to COMMAREA format.

Deploying and testing the CICS EXCI flow

We follow the same process as in “Deploying and testing the DB2 VSAM flow” on page 324. The artifacts for the bar file are different, as shown in Figure 7-24.


Add to Broker Archive

Select deployable files to add to the broker archive.
Deployable files are message flows, message sets, style sheets, XML and JAR files.
To be deployable, a file must have no error.
A message flow must also include an input node.

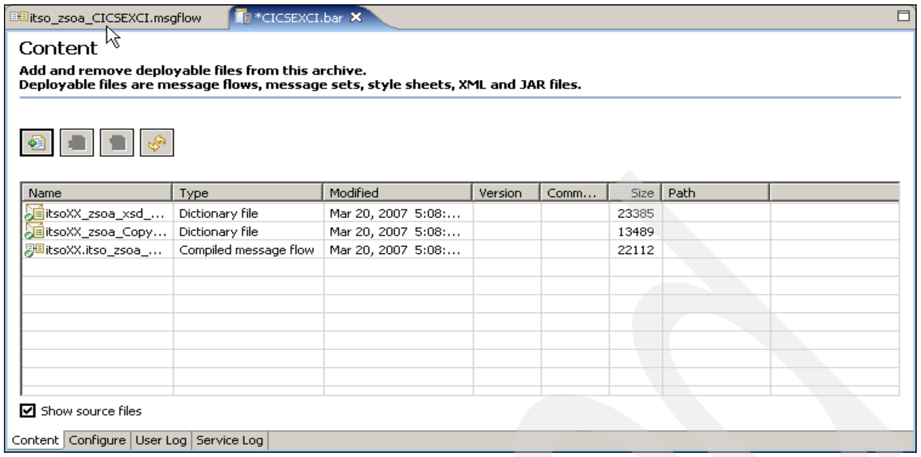
Deployable resources:

- ☐ Servers
- ☑ itsoXX_zsoa_Copy_CustRec
- ☑ itsoXX_zsoa_xsd_CustReqReply
- ☑ itso_zsoa_CICSEXCI
- ☐ itso_zsoa_Lab_Materials

☐ Include source files
☐ Compile ESQL for broker version 2.1
☑ Override configurable property values

Figure 7-24 Artifacts to include in the .bar file

Save the .bar file and deploy it to the Broker environment executing in the z/OS environment. First, you need to connect the WMB Toolkit to the Configuration Manager. The Configuration Manager controls and filters access to the Broker. All artifacts are first verified by the Configuration Manager. We connected to our Broker execution environment first and then used the Deploy menu to deploy the .bar file.


itso_zsoa_CICSEXCI.msgflow | *CICSEXCI.bar

Content

Add and remove deployable files from this archive.
Deployable files are message flows, message sets, style sheets, XML and JAR files.

Name	Type	Modified	Version	Comm...	Size	Path
itsoXX_zsoa_xsd_...	Dictionary file	Mar 20, 2007 5:08:...			23385	
itsoXX_zsoa_Copy...	Dictionary file	Mar 20, 2007 5:08:...			13489	
itsoXX.itso_zsoa_...	Compiled message flow	Mar 20, 2007 5:08:...			22112	

☑ Show source files

Figure 7-25 .bar file contents for the CICS EXCI flow

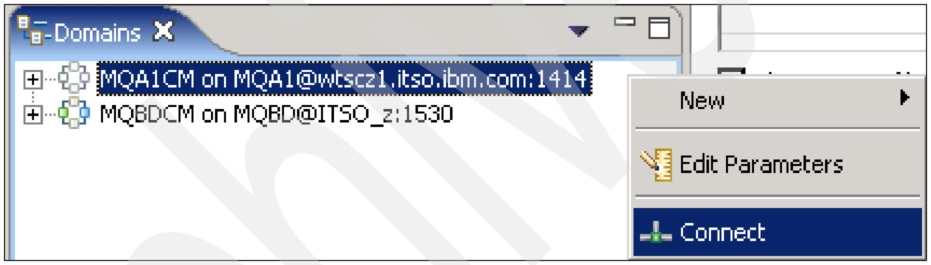

Domains
MQA1CM on MQA1@wtscz1.itso.ibm.com:1414
MQBDCM on MQBD@ITSO_z:1530
New
Edit Parameters
Connect

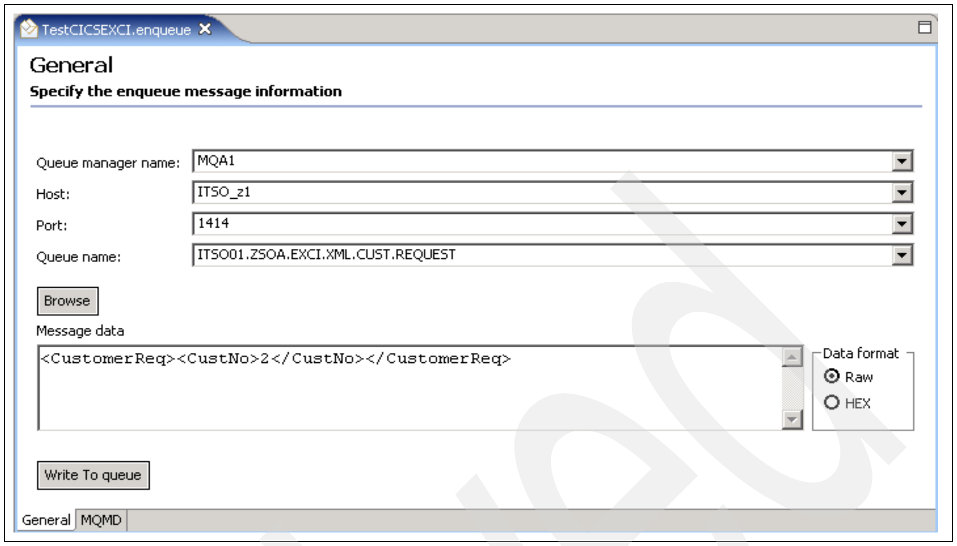
Figure 7-26 Connecting to the Broker domain

For testing, we chose the following path first:

XML Message is put on a Queue → MQ_XML_EXCI_IN → A → B → C → D → E

We used the toolkit MQ put functionality, as shown in Figure 7-27 on page 341.

Note: Make sure you specify the reply queue name in the MQMD tab in the toolkit.


TestCICSEXCI.enqueue

General

Specify the enqueue message information

Queue manager name: MQA1

Host: ITSO_z1

Port: 1414

Queue name: ITSO01.ZSOA.EXCI.XML.CUST.REQUEST

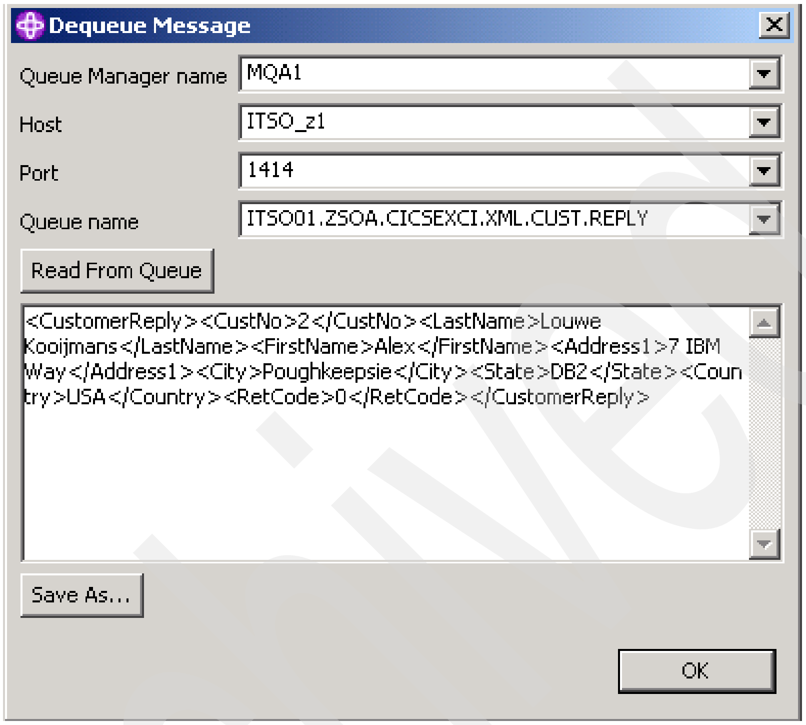
Message data

```
<CustomerReq><CustNo>2</CustNo></CustomerReq>
```

Data format: [✓ Raw] HEX

Figure 7-27 Putting a message onto the queue

Read the response from the reply through the dequeue tool, as shown in Figure 7-28.



Queue Manager name: MQA1
Host: ITSO_z1
Port: 1414
Queue name: ITSO01.ZSOA.CICSEXCI.XML.CUST.REPLY

```
<CustomerReply><CustNo>2</CustNo><LastName>Louwe Kooijmans</LastName><FirstName>Alex</FirstName><Address1>7 IBM Way</Address1><City>Poughkeepsie</City><State>DB2</State><Country>USA</Country><RetCode>0</RetCode></CustomerReply>
```

Figure 7-28 Message read from the reply queue

The next path of the flow we will test is:

SOAP HHTP Message → **HTTP Input** → **Soap2CPy** → **CICS request** → **Select Reply** → **Cpy2Soap** → **HTTP Reply**

We use the simple Java program that we introduced when testing the router scenario. We change the path variable in the Java program from:

```
String path = "/itso01/db2vsam/getCustomerInfo
```

to

```
String path = "/exci/getCustomerInfo
```

We send the customer number as 4. Refer to Example 7-6 on page 327 for the complete SOAP message. We see the following for the output:

```
HTTP/1.1 200 OK
Content-Type: text/xml;charset=utf-8
Content-Length: 542
Date: Wed, 03 Oct 2007 13:20:45 GMT
Server: Apache-Coyote/1.1
Connection: close

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/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"><soapenv:Body><Custom
erReply><CustNo>4</CustNo><LastName>Connolly</LastName><FirstName>Michael</
FirstName><Address1>22 Enterprise
St</Address1><City>Woodstock</City><State>NY</State><Country>USA</Country><
RetCode>0</RetCode></CustomerReply></soapenv:Body></soapenv:Envelope>
```

7.2.3 Calling out to a Web Service from WMB

In this section, we illustrate how to call a Web Service from WMB. We use the same flow from the previous section with a minor change to a single node, the CICS Request node.

Once the flow is complete, it looks as shown in Figure 7-29 on page 344.

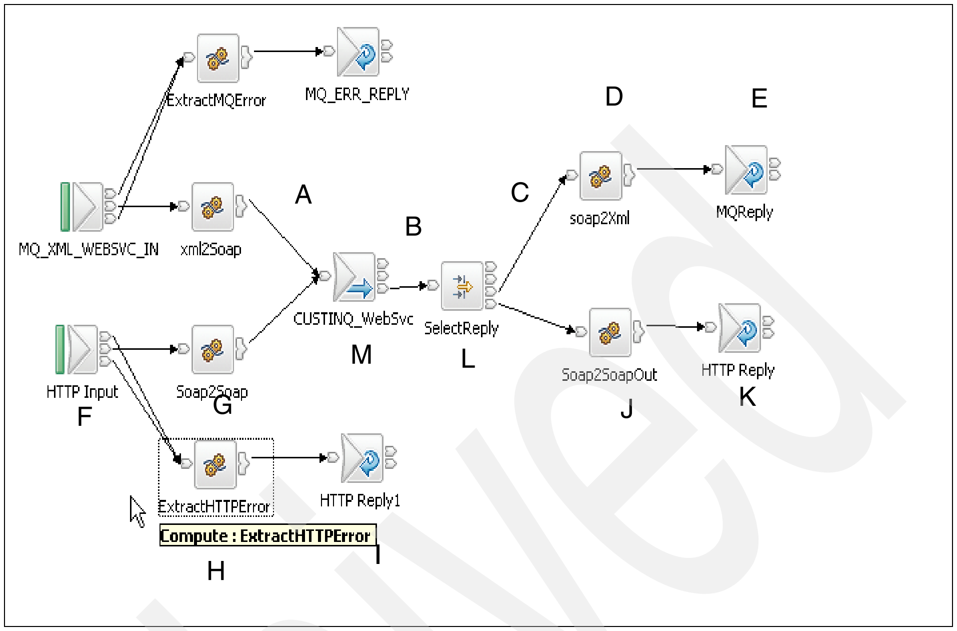


Figure 7-29 Web Service flow

In Chapter 2, “Accessing CICS from WebSphere using Web Services over HTTP” on page 3, the CUSTINQ CICS program was exposed as a Web Service. The CUSTInq_WebSvc node will call out to this Web Service from WMB. Table 7-5 shows the important connections for the MQ input leg.

Table 7-5 Terminal connections for WEBSVC (mq leg)

From/To	**Output terminal to use**
MQ_XML_WEBSVC_IN to A	Out
A to B	Out
B to C	Out
Third terminal on C to D	False
D to E	Out

Table 7-6 shows the important connections for the HTTP input leg.

Table 7-6 Terminal connections WEBSVC flow(http leg)

From/To	**Output Terminal to use**
First terminal on F to H connection	Failure
Second terminal on F to G connections	Out
Third terminal on F to H connections	Catch
H to I	Out
G to M	Out
Fourth terminal on L to J	True
J to K	Out

The following node customizations are needed:

- HTTP request node (HTTP Input (letter block F)).
  Click **HTTP Input**. Select **Basic** on the Properties page. Type `/webSvc/getCustomerInfo` in the path suffix for the URL.
- HTTP request node (CUSTINQ_WebSvc (letter block M)).
  The URL is the same as the URL used for accessing the CICS Web Service introduced in Chapter 2, "Accessing CICS from WebSphere using Web Services over HTTP" on page 3.

Paste the supplied custom code into the Compute node, set properties for each Compute node. The file name for this code is WEBSVC_flow_esql.txt and is also supplied in the additional material.

Note: The code you will paste into the compute node is custom code, provided for this scenario. The custom code contains multiple modules; when you paste the block into one compute node, it is made available to all of the compute nodes. We do need to instruct each compute node which module to use. We do that by setting a property value.

Use the information in Table 7-7 to select the esql module for each Compute node.

Table 7-7 Select the esql module for compute nodes (WEBSVC)

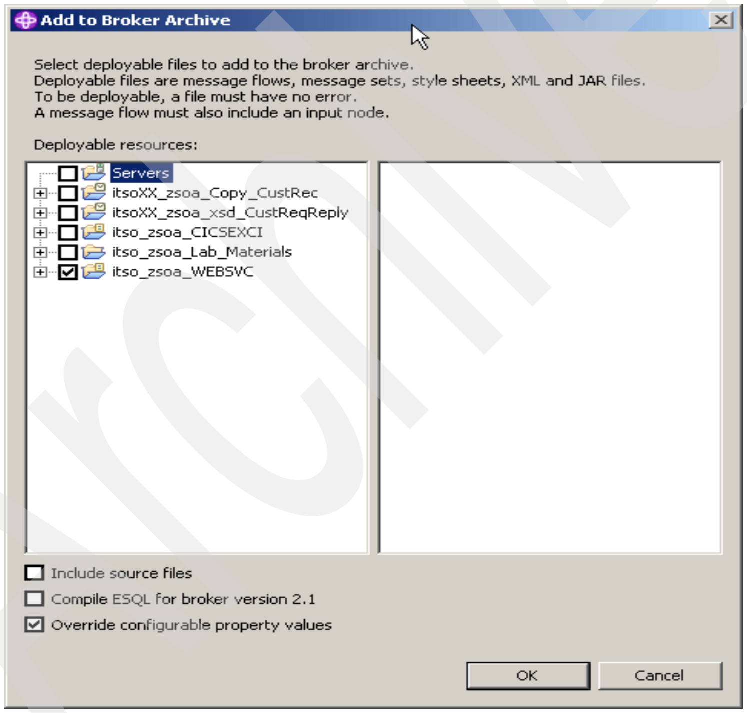
Compute Node	**Module Name**
xml2Soap	itsoXX.itso_zsoa_WebSvc_xml2Soap

Compute Node	Module Name
Soap2Soap	itsoXX.itso_zsoa_WebSvc_soap2soap
ExtractHTTPError	itsoXX.itso_zsoa_WebSvc_flow_ExtractError
soap2Xml	itsoXX.itso_zsoa_WebSvc_soapXml
Soap2SoapOut	itsoXX.itso_zsoa_WebSvc_soap2soapOut
SelectReply	itsoXX.itso_zsoa_WebSvc_filterReply

Deploying and testing the CICS Web Service flow

This time we give this bar file the name WEBSVC.

Include the artifacts shown in Figure 7-30.


Add to Broker Archive

Select deployable files to add to the broker archive.
Deployable files are message flows, message sets, style sheets, XML and JAR files.
To be deployable, a file must have no error.
A message flow must also include an input node.

Deployable resources:

- [ ] Servers
- [ ] itsoXX_zsoa_Copy_CustRec
- [ ] itsoXX_zsoa_xsd_CustReqReply
- [ ] itso_zsoa_CICSEXCI
- [ ] itso_zsoa_Lab_Materials
- [✓] itso_zsoa_WEBSVC

- [ ] Include source files
- [ ] Compile ESQL for broker version 2.1
- [✓] Override configurable property values

Figure 7-30 Selecting the .bar artifacts for the CICS Web Service flow

Press Ctrl-s to save the WEBSVC.bar file. Before deploying the file to the Broker on z/OS, connect the WMB Toolkit to the Configuration Manager. The

Configuration Manager controls and filters access to the Broker. All artifacts are first verified by the Configuration Manager.

The testing steps are exactly the same as outlined for the earlier scenarios except for the message content. The following examples show the source message formats.

Example 7-10 Input to MQ

```
<CustomerReq>
   <CustNo>9</CustNo>
</CustomerReq>
```

Example 7-11 Input to HTTP

```
<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">
   <soapenv:Body>
      <CustomerReq>
         <CustNo>6</CustNo>
      </CustomerReq>
   </soapenv:Body>
</soapenv:Envelope>
```

7.3 Additional material

The following is a description of the material that has been supplied with this chapter. Refer to Appendix B, “Additional material” on page 397 for download instructions.

Table 7-8 Supplied material

Name	Purpose
VSAMREC.cpy	COBOL COPYBOOK definition of the VSAM file structure.
Define_Queue_Script.txt	Script for creating all the MQ objects needed for this chapter
DB2VSAM_flow.esql.txt	ESQL programs for DB2 VSAM flow

Name	Purpose
customer_table_create.sql	Definition of the CUSTOMER DB2 table
testDB2VSAM.enqueue	This file can be imported into WMB and used for putting sample message in the queue.
SendCustomerAndPrintResponse .java	Java source file for testing out Customer send
DB2VSAM_PI.zip	The Project interchange for routing scenario
COMMAREA.cpy	COBOL Message definition for protocol conversion
CustomerInfoRequest.xsd and CustomerInfoReply.xsd.	XML message definitions for protocol conversion scenario
CICSEXCI_PI.zip	The Project interchange file for the Protocol transformation scenario
CICSEXCI_flow_esql.txt	ESQL code for protocol transformation scenario
WEBSVC_flow_esql.txt	ESQL code for Web Service call out scenario
WEBSVC_PI.zip	The Project interchange file for the Web Service scenario

8

Using WebSphere Enterprise Service Bus for connectivity

This chapter provides information on how WebSphere Enterprise Service Bus (WESB) could be used as an integration layer between WebSphere Application Server and back-end systems.

We are using an example in which we access a CICS back-end application through WESB and discuss the architecture, requisites and implementation of the WESB-related components.

8.1 WebSphere ESB introduction

WebSphere Enterprise Service Bus (WESB) supports the integration of service-oriented, message-oriented, and event-driven technologies to provide a standards-based messaging infrastructure to companies wanting a fast start to an Enterprise Service Bus.

An Enterprise Service Bus (ESB) is a logical architectural component that provides an integration infrastructure consistent with the principles of Service-Oriented Architecture (SOA); see Figure 8-1.

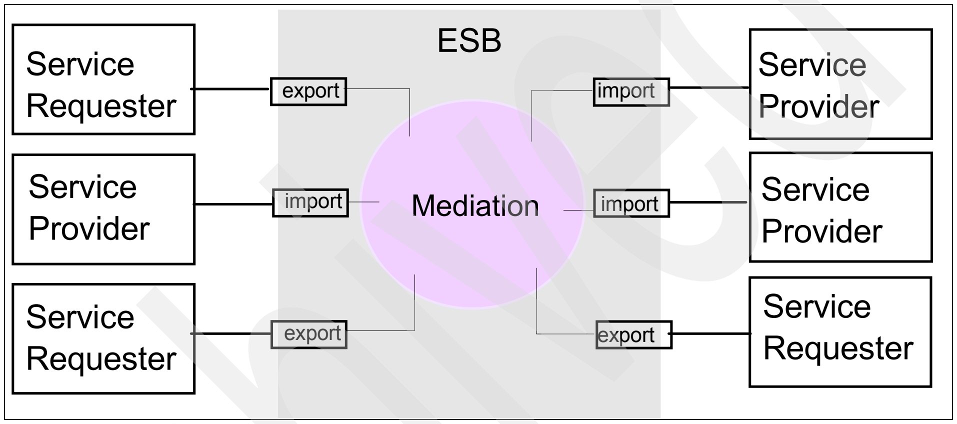


Figure 8-1 Enterprise Service Bus (ESB) - simplified architecture

You can develop service applications that, when deployed to WebSphere Enterprise Service Bus (WESB), enable message-based communication between services and can operate on and manipulate messages in flight between interaction endpoints (*mediation*).

A service application has an associated Service Component Architecture (SCA) module. You deploy service applications to WebSphere ESB in EAR (enterprise archive) files.

The kind of SCA modules that WebSphere ESB supports are called *mediation modules*. Mediation modules allow you to change the format, content, or target of service requests.

WebSphere ESB provides *message service clients* that extend the connectivity of the Enterprise Service Bus.

8.2 Infrastructure setup

In the following sections we provide information about what you need to execute a scenario such as the one we discuss in the chapter.

8.2.1 Components used

WebSphere Enterprise Service Bus (WESB)

WebSphere ESB is based on the J2EE 1.4 infrastructure and associated platform services provided by WebSphere Application Server for z/OS Version 6.0.2.

WebSphere Integration Developer (WID)

WebSphere Integration Developer (WID) is the development environment for WebSphere ESB. You can use WebSphere Integration Developer (WID) to graphically model and assemble *mediation components* from *mediation primitives*, and assemble *mediation modules* from mediation components.

Back-end systems

Services implemented in back-end systems can be accessed from WebSphere Enterprise Service Bus (WESB). In our example we use a Web Service implemented in CICS.

8.2.2 Prerequisites

WebSphere Enterprise Service Bus (WESB)

For WebSphere Enterprise Service Bus (WESB) for z/OS, Version 6.02, the requirements are as follows:

- Operating systems
  - z/OS v 1.6
  - z/OS v 1.7
- Application server environment
  - WebSphere Application Server for z/OS Version 6.0.1 with the latest service level
- Databases
  - DB2 V8 (recommended)
  - DB2 V7

  - Cloudscape® 5.1
- Back-end Transaction Managers
  - CICS Transaction Server for OS/390 1.3
  - CICS Transaction Server for z/OS 2.2, 2.3, and 3.1
  - IMS V8 or V9
- Web Server
  - IBM HTTP Server for z/OS
- Java
  - IBM SDK v1.4.2 SR6 i-fix 110979, 111872, and 112270
- LDAP
  - IBM z/OS Security Server
  - IBM Lotus® Domino® Enterprise Server 6.0.3, 6.0.5, 6.5.1 or 6.5.4
- WebSphere MQ
  - WebSphere MQ for z/OS 5.3.1
  - WebSphere MQ for z/OS 6.0

The WESB detailed system requirements Web page has more information:

http://www.ibm.com/support/docview.wss?uid=swg27006912

WebSphere Integration Developer (WID) prerequisites

For our scenario we used WebSphere Integration Developer (WID) Version 6.0.2.2ifix1. The prerequisites are:

- Operating systems
  - Windows 2000 Advanced Server SP3 or SP4
  - Windows 2000 Server SP3 or SP4
  - Windows 2000 professional SP3 or SP4
  - Windows 2003 Enterprise Edition
  - Windows 2003 Standard Edition
  - Windows XP Professional with SP1 or SP2
  - Red Hat Enterprise Linux 3.0 WS Update 2
  - SuSE Linux Enterprise Server 9
- Hardware requirements
  - Intel® Pentium III Processor at 1GHz or faster (32-bit kernel support only)

- Minimum 1 GB physical memory (1 to 2 GB physical memory is recommended)
- Minimum 5.8 GB available disk space for a full installation
- Minimum 1024 x 768 display resolution (1280 x 1024 is recommended)

8.2.3 WESB configuration on z/OS

WebSphere ESB for z/OS installation and configuration is tightly integrated with and dependent on the installation and configuration of WebSphere Application Server for z/OS. Before you can create a WESB server you need to have customized the WebSphere Application Server for z/OS. This results in the default profile for that standalone application server being "augmented" with WebSphere ESB for z/OS standalone server configuration data, configuring that standalone application server into a standalone server.

In our scenario we used a standalone WebSphere Application Server and WESB V6.0.2 for z/OS. Refer to the following online documentation regarding installation and configuration of WESB on z/OS:

```
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp
http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp
http://www.redbooks.ibm.com/abstracts/redp4349.html?Open
http://www.redbooks.ibm.com/abstracts/redp4388.html?Open
http://www.redbooks.ibm.com/abstracts/redp4196.html?Open
```

8.2.4 WID configuration

WID is the standard development environment for WESB. WID Version 6.0.2 is compatible with RSDP level 6.0.1.x, which means that if you have other Rational products installed on your machine, be sure they are at level 6.0.1.x in order to install the WID.

We also applied the latest fix available at the product support page:

```
http://www-306.ibm.com/software/integration/wid/support/
```

8.3 Development approach

We use WebSphere Integration Developer Version 6.02 as the development tool to create all the elements necessary for the integration between WebSphere Application Server and the back-end system.

We will show how to develop the interfaces, data objects, mediation flows, bindings, filters, mappings, and finally test the application by simulating requests to Web Services implemented in back-end systems. This WID project can be deployed to WESB or to the WebSphere Process Server (WPS).

We create a flow as shown in Figure 8-2.

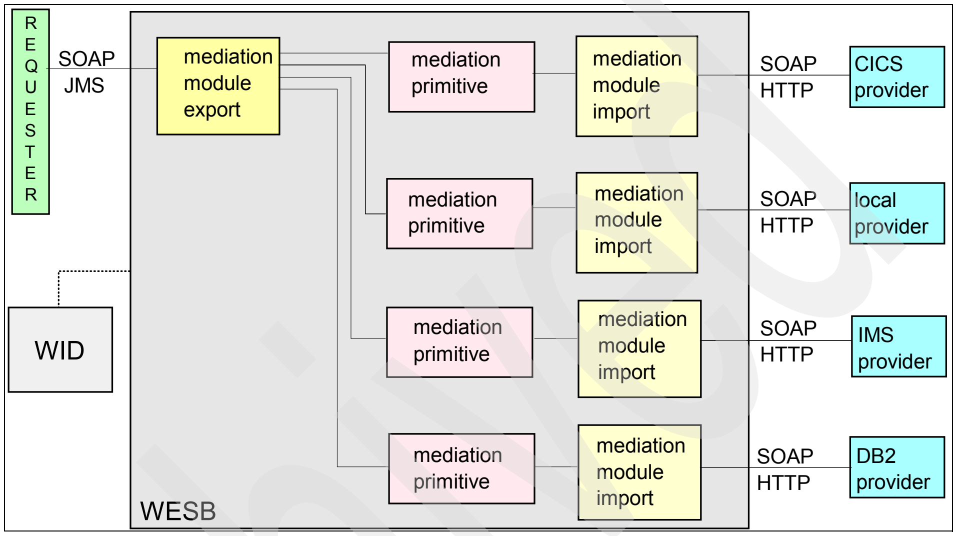


Figure 8-2 Our scenario

8.3.1 Getting started

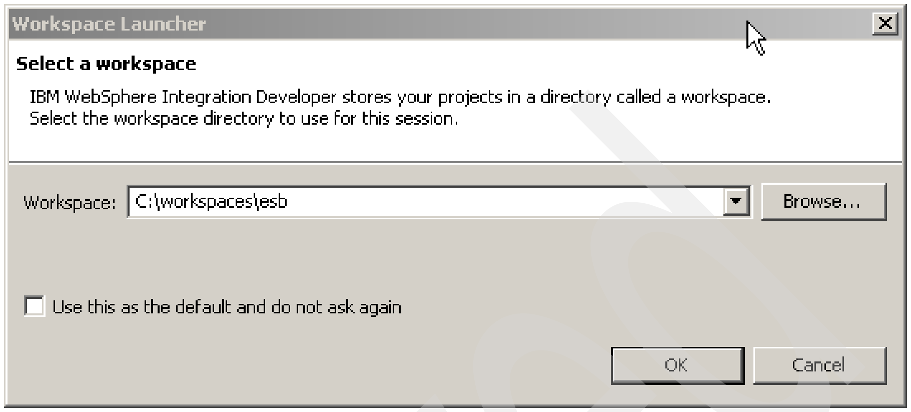
There are a few things you need to do to get ready for the actual development steps:

- Starting WID on your workstation
- Bringing up the Business Integration perspective
- Starting the WESB server

Starting WebSphere Integration Developer

1. Start WebSphere Integration Edition. From the Windows Start Menu, select **Programs** → **IBM WebSphere** → **Integration Developer V6.0.2** → **WebSphere Integration Developer V6.0.2**.

2. In the Select a Workspace pane enter the path to the workspace you use for this project (use **Browse** if necessary); see Figure 8-3.


Workspace Launcher

Select a workspace

IBM WebSphere Integration Developer stores your projects in a directory called a workspace.
Select the workspace directory to use for this session.

Workspace: C:\workspaces\esb

Figure 8-3 Select a workspace window

Open the Business Integration perspective

If a Welcome page is displayed, click the ☒ to close it. WebSphere Integration Developer should open up in the Business Integration perspective. If it does not, navigate to the process of opening the Business Integration perspective, as follows:

1. In WID, select **Window** → **Open Perspective** → **Other**.
2. Check **Show All**, as shown in Figure 8-4.
3. Choose **Business Integration (default)**.

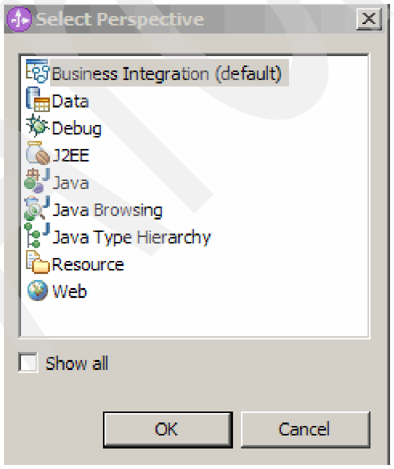
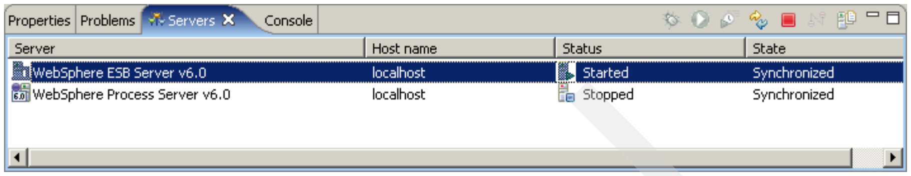


Figure 8-4 Select the Business Integration perspective window

Starting the WebSphere ESB Server

1. On the bottom window pane, select the **Servers** tab as shown in Figure 8-5.



Server	Host name	Status	State
WebSphere ESB Server v6.0	localhost	Started	Synchronized
WebSphere Process Server v6.0	localhost	Stopped	Synchronized

Figure 8-5 Servers tab

2. Right-click the **WebSphere ESB server** and select **Start**.
3. Wait until the server Status is Started.
4. If you happen to have a WebSphere Administrative console open in any browser window, be sure to close it.

8.3.2 Creating components for the flow

We now create some modules for our flow.

Creating a new mediation module

This task creates a new mediation module. This is a project for designing ESB mediation.

1. Right-click in the Business Integration pane, select **New** → **Mediation Module**, as shown in Figure 8-6.

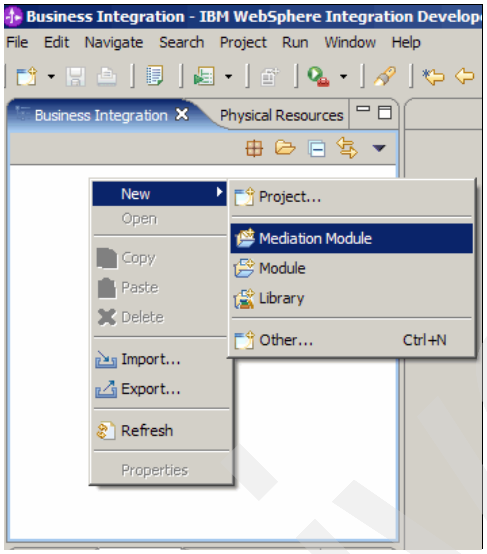
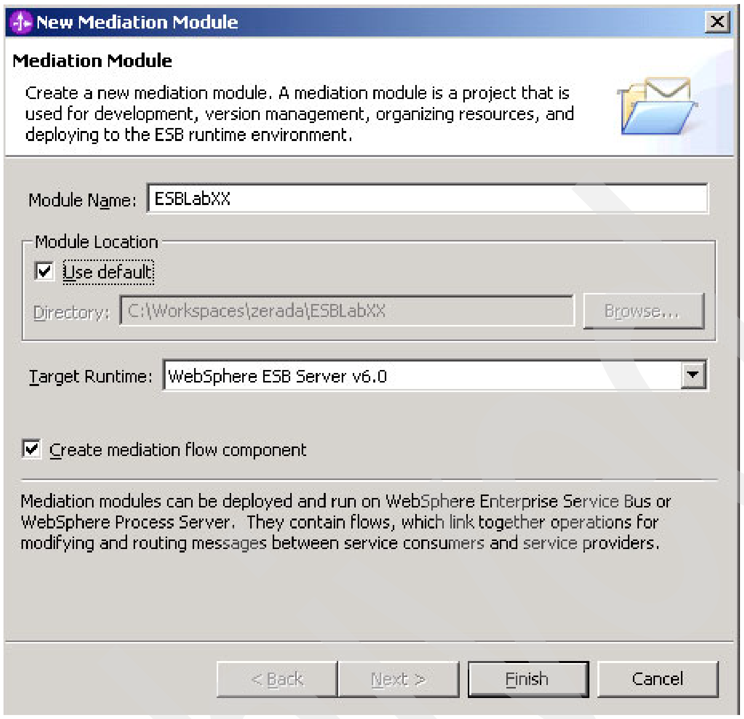


Figure 8-6 New Mediation Module window

You will then see the New Mediation Module window, as shown in Figure 8-7 on page 358.


New Mediation Module

Mediation Module

Create a new mediation module. A mediation module is a project that is used for development, version management, organizing resources, and deploying to the ESB runtime environment.

Module Name: ESBLabXX

Module Location

[✓] Use default

Directory: C:\Workspaces\zerada\ESBLabXX

Target Runtime: WebSphere ESB Server v6.0

[✓] Create mediation flow component

Mediation modules can be deployed and run on WebSphere Enterprise Service Bus or WebSphere Process Server. They contain flows, which link together operations for modifying and routing messages between service consumers and service providers.

Figure 8-7 New Mediation Module dialog

2. In Module Name you enter the name of the mediation module. In our scenario, we used `ESBLabXX`.
3. In Target Runtime you select the WebSphere ESB version you plan to use for deployment. In our scenario we are using WebSphere ESB Server v6.0.
4. Click **Finish**. The new mediation module ESBLabXX will appear as shown in Figure 8-8.

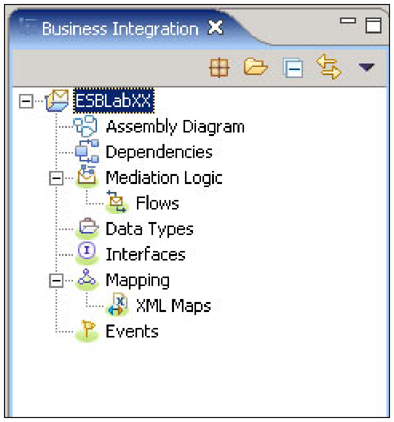


Figure 8-8 New mediation module created

Adding WSDL with Business Objects to the project

In the initial workspace there are two Web Service applications and a test client application. In the following steps, you will copy the required WSDL files into the mediation module. These WSDL files are included as additional material in this book.

1. In WebSphere Integration Developer, right-click **Interfaces** and select **Import**, as shown in Figure 8-9.

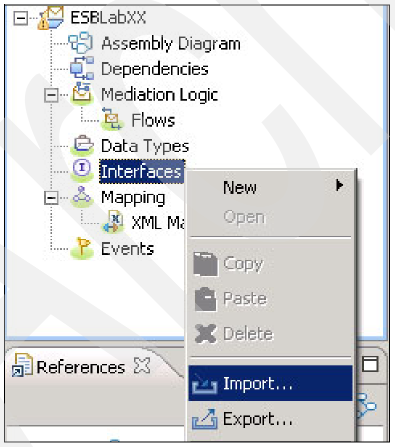


Figure 8-9 Importing an interface

2. Select the WSDL file as shown in Figure 8-10 and click **Finish**.

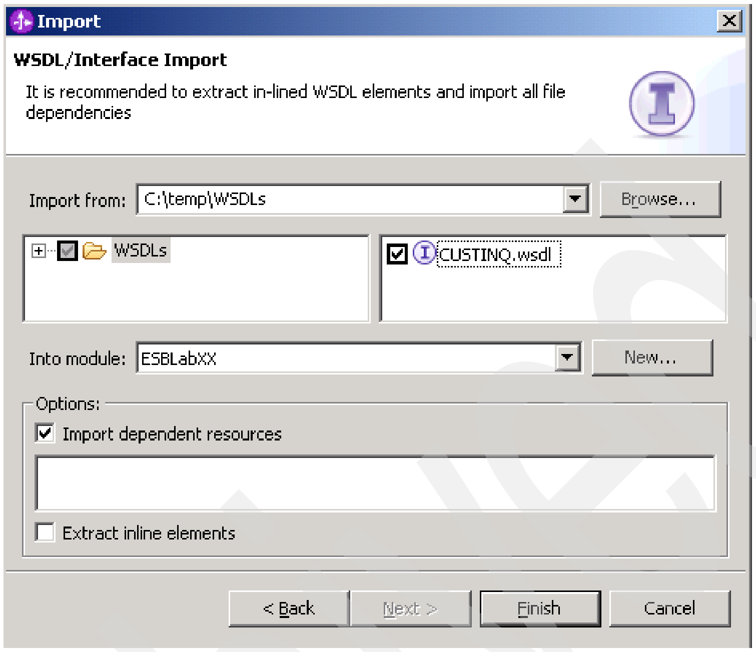

Import
WSDL/Interface Import
It is recommended to extract in-lined WSDL elements and import all file dependencies
Import from: C:\temp\WSDLs
Browse...
WSDLs
CUSTINQ.wsdl
Into module: ESBLabXX
New...
Options:
Import dependent resources
Extract inline elements
< Back
Next >
Finish
Cancel

Figure 8-10   Importing WSDLs files

As a result, several new objects are created in the Interfaces, Data Types and Web Service Ports folders, as shown in Figure 8-11.

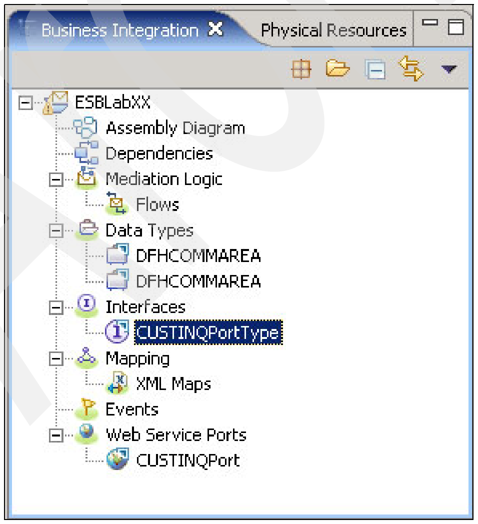
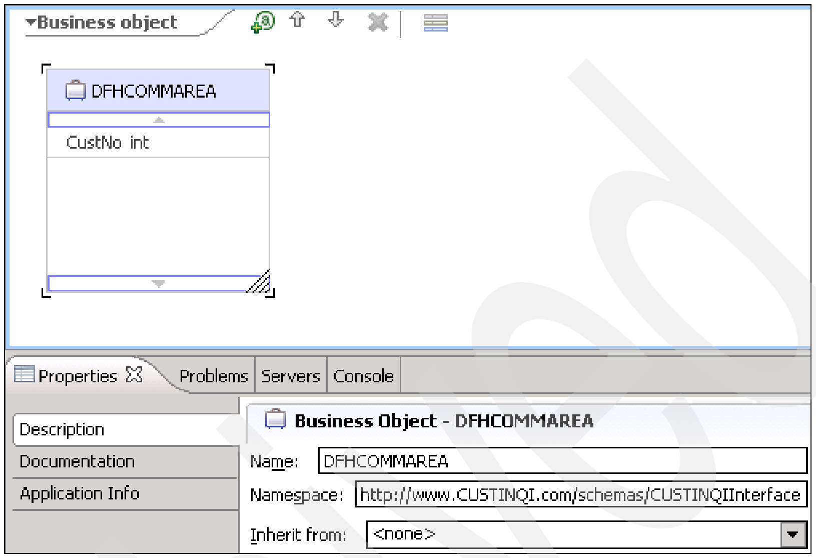


Figure 8-11   Workspace with new interface

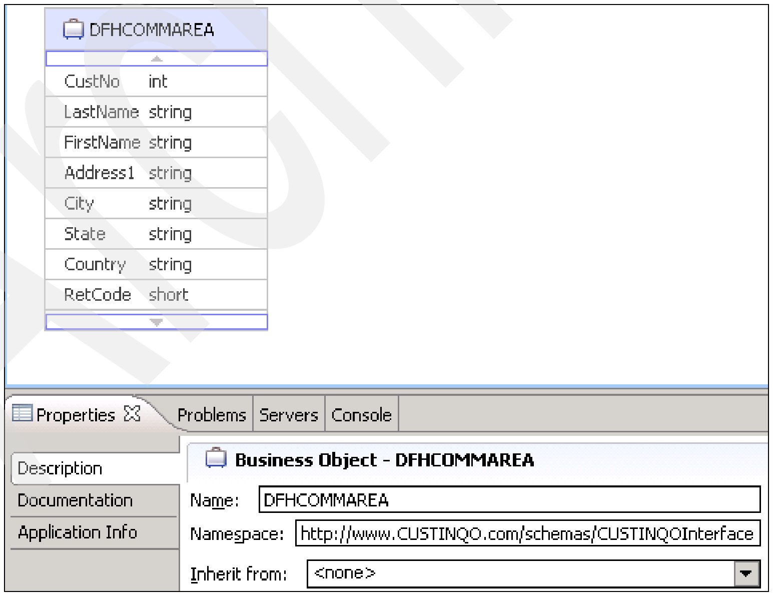
We can now see the Data Types structures. Double-click each of the DFHCOMMAREA Data Types. This will open another window, as shown in Figure 8-12 for the request and Figure 8-13 for the response.



CustNo int

Name: DFHCOMMAREA
Namespace: http://www.CUSTINQI.com/schemas/CUSTINQIInterface
Inherit from: <none>

Figure 8-12   DFHCOMMAREA Data Type - Input



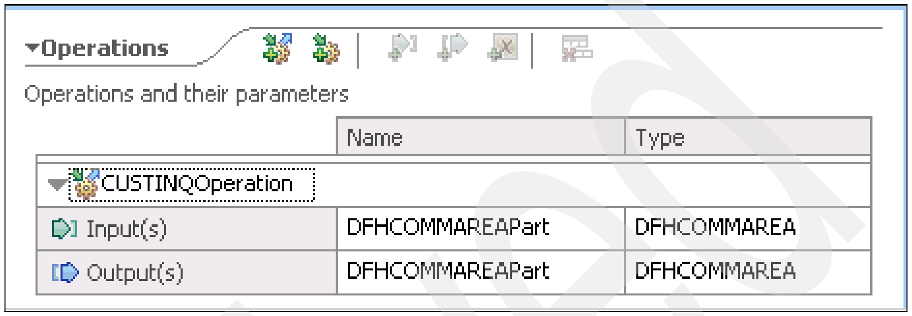
CustNo	int
LastName	string
FirstName	string
Address1	string
City	string
State	string
Country	string
RetCode	short

Name: DFHCOMMAREA
Namespace: http://www.CUSTINQO.com/schemas/CUSTINQOInterface
Inherit from: <none>

Figure 8-13   DFHCOMMAREA Data Type - Output

Note: Although the Data Types have the same name, the Name Spaces are different, as we can see in the images above.

To see the CUSTINQPortType interface structure, double-click it, which will result in a window as shown in Figure 8-14.



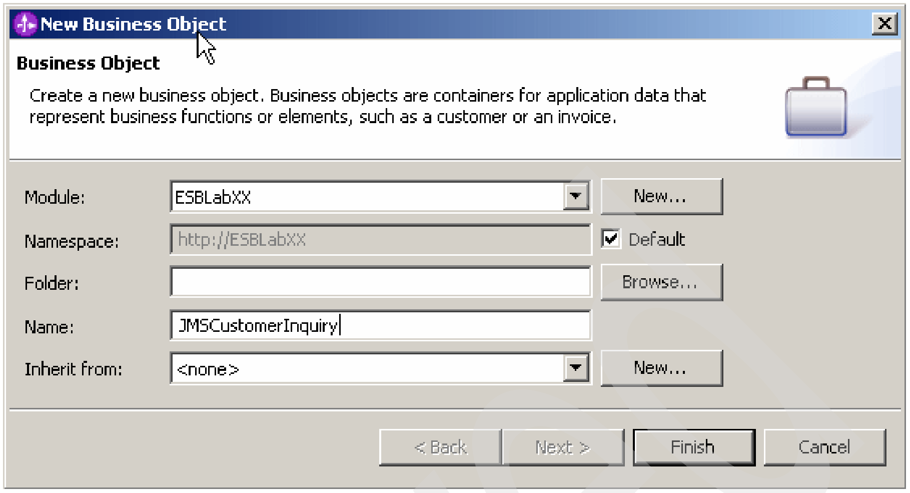
	Name	Type
CUSTINQOperation		
Input(s)	DFHCOMMAREAPart	DFHCOMMAREA
Output(s)	DFHCOMMAREAPart	DFHCOMMAREA

Figure 8-14 CUSTINQPortType interface

8.3.3 Creating the Business Objects

In this section, we create Business Objects for the JMS service, the request and response objects. The request object will include a customer number. The response object will contain the customer information.

1. In the Business Integration view, right-click **Data Types** and select **New** → **Business Object**. The dialog shown in Figure 8-15 on page 363 is displayed. For **Name** we entered `JMSCustomerInquiry` and clicked **Finish.**


New Business Object

Business Object

Create a new business object. Business objects are containers for application data that represent business functions or elements, such as a customer or an invoice.

Module: ESBLabXX

Namespace: http://ESBLabXX ☑ Default

Folder:

Name: JMSCustomerInquiry

Inherit from: <none>

Figure 8-15   New Business Object window

The Business Object editor window pane is displayed. Click **Add Attribute** to add attributes, as highlighted in Figure 8-16.

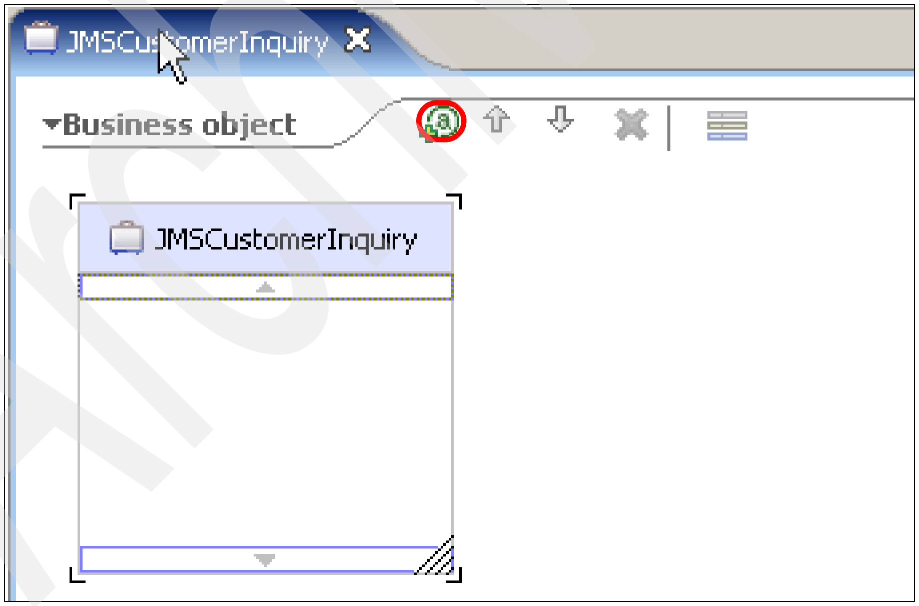


Figure 8-16   Business Object definition for JMSCustomerInquiry

2. For attribute name, change the default value, attribute1, to `CustNo`. You can make the change in the bottom properties pane or just type in the field. For easy switching between fields, you can use the Tab key.
3. For changing the attribute type, change the default value, string, to `int`.

   The result should look like in Figure 8-17.

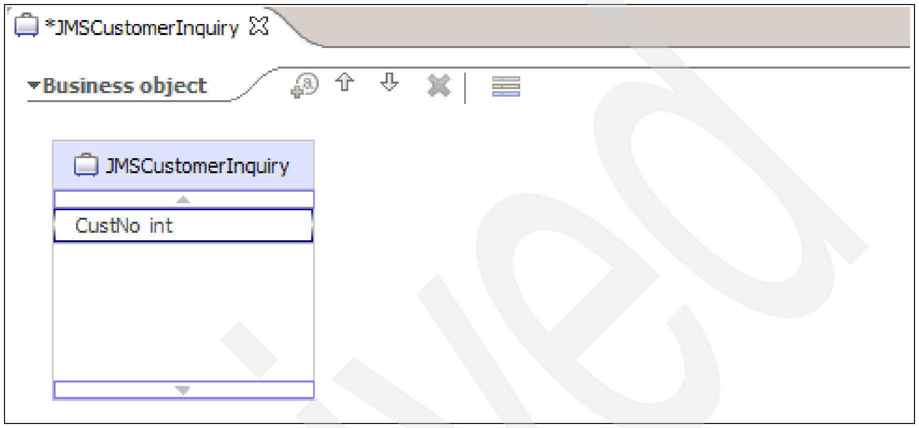
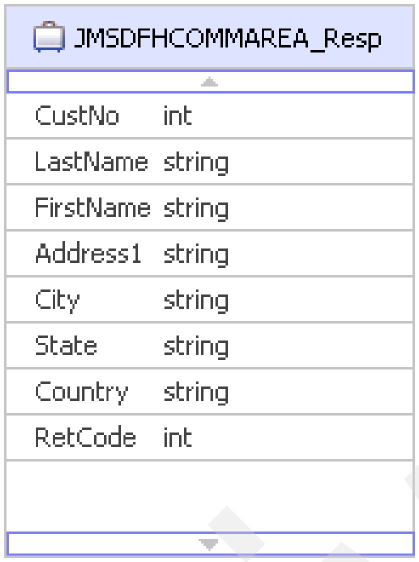


Figure 8-17 Completed Business Object definition for JMSCustomerInquiry

4. Press Ctrl+s to save the Business Object.
5. Click the ☒ on the editor tab to close the Business Object editor.

Now we create the JMSCustomer object. We perform the same steps as for the previous object. This object will contain the response.

1. Right-click **Data Types** and select **New** → **Business Object**.
2. On the Create a new Interface screen, for Name we entered `JMSCustomer`. After that, we clicked **Finish**.
3. Add the following attributes and data types, as shown in Figure 8-18 on page 365. Refer to the previous steps for adding attributes.



JMSDFHCOMMAREA_Resp

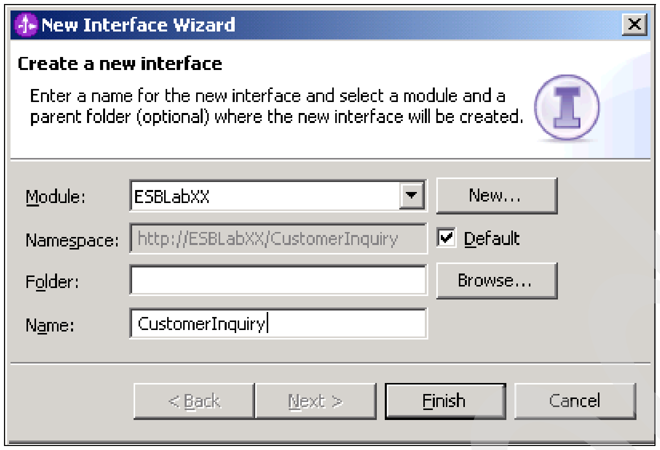
CustNo	int
LastName	string
FirstName	string
Address1	string
City	string
State	string
Country	string
RetCode	int

Figure 8-18 Completed Business Object definition for JMSDFHCOMMAREA

4. Press Ctrl+s to save the Business Object.
5. Click the ☒ on the editor tab to close the business object editor.

Creating the JMS interface

1. In the Business Integration view, select **Interfaces** as shown in Figure 8-11 on page 360.
2. Right-click **Interfaces** and select **New** → **Interface**. The New Interface wizard is shown in Figure 8-19 on page 366.
3. For name we entered `CustomerInquiry` and clicked **Finish**.


New Interface Wizard

Create a new interface

Enter a name for the new interface and select a module and a parent folder (optional) where the new interface will be created.

Module: ESBLabXX

Namespace: http://ESBLabXX/CustomerInquiry ✓ Default

Folder:

Name: CustomerInquiry

Figure 8-19 New Interface Wizard

The Interface editor is displayed as shown in Figure 8-20. Figure 8-21 on page 367 shows the completed interface.

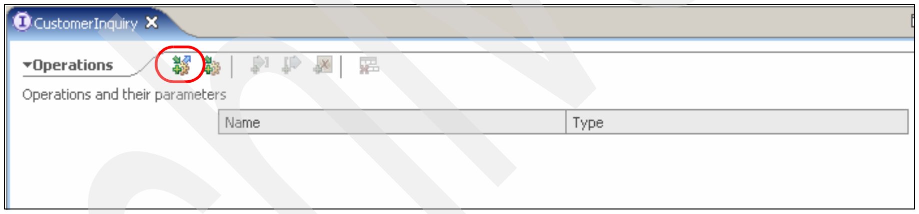
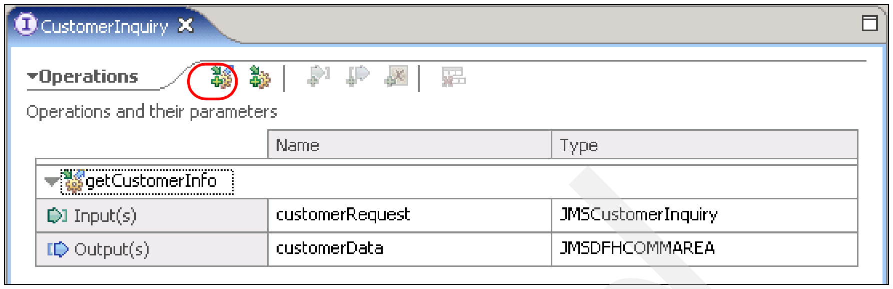


Figure 8-20 CustomerInquiry Interface in editor

4. Click **Add Request Response Operation**, which is shown circled in Figure 8-20 and Figure 8-21 on page 367.
5. Rename operation1 to `getCustomerInfo`. This can be done in the top pane but you can also do it in the bottom properties - description pane.
6. Rename input1 to `customerRequest`.
7. Rename output1 to `customerData`.
8. Change data type of customerRequest input from `string` to `JMSCustomerInquiry`.
9. Change data type of customerData output from `string` to `JMSDFHCOMMAREA`.


CustomerInquiry

Operations

Operations and their parameters

	Name	Type
getCustomerInfo		
Input(s)	customerRequest	JMSCustomerInquiry
Output(s)	customerData	JMSDFHCOMMAREA

Figure 8-21 Completed CustomerInquiry Interface in editor

10.Press Ctrl+s to save the business object.

11.Click ☒ on the editor tab to close the business object editor.

8.3.4 Creating an Assembly Diagram

The Web service and the JMS objects and interfaces are defined. The next step is to build an *Assembly Diagram*. The Assembly Diagram links the JMS service to the Web Service through mediation. The completed assembly diagram is shown in Figure 8-30 on page 373.

1. Double-click the **Assembly Diagram** for the ESBLabXX project in the Business Integration view.

   The Assembly Diagram is opened in the center window containing a mediation object as shown in Figure 8-22.

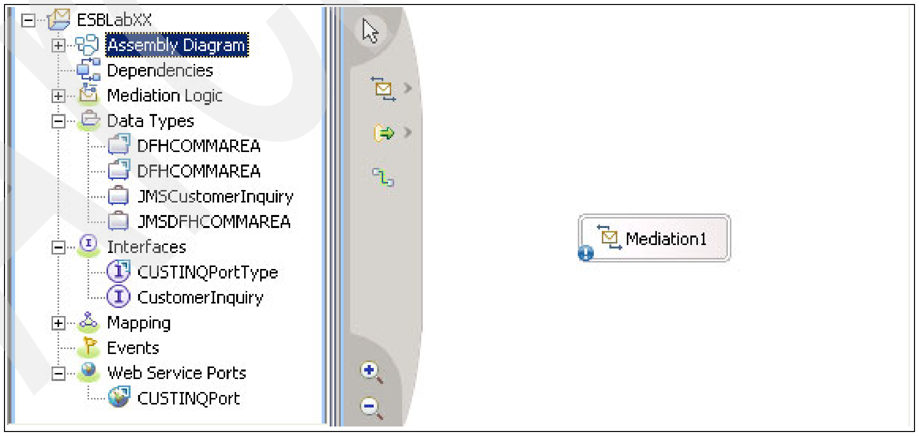


Figure 8-22 Assembly Diagram editor window

Adding an Export Interface

First we add the CustomerInquiry Interface. This is our Export part.

1. Drag the CustomerInquiry interface from the Interfaces folder and place it to the left of the Mediation1 object.

Attention: Don't drag the interface inside the Mediation1 box!

2. A Component Creation dialog box is opened. Select **Export with no Binding**, as shown in Figure 8-23.

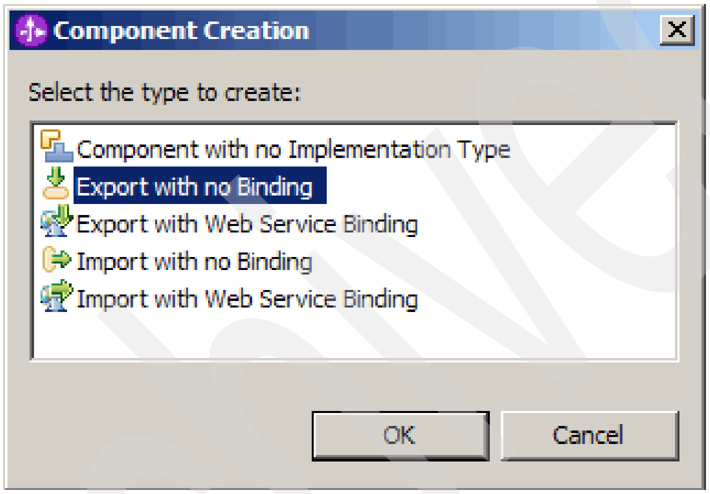

Component Creation
Select the type to create:
Component with no Implementation Type
Export with no Binding
Export with Web Service Binding
Import with no Binding
Import with Web Service Binding
OK
Cancel

Figure 8-23 Component Creation dialog

3. Select the newly created **CustomerInquiryExport1** object, right-click it and select **Generate Binding** → **Message Binding** → **JMS Binding**, as shown in Figure 8-24.

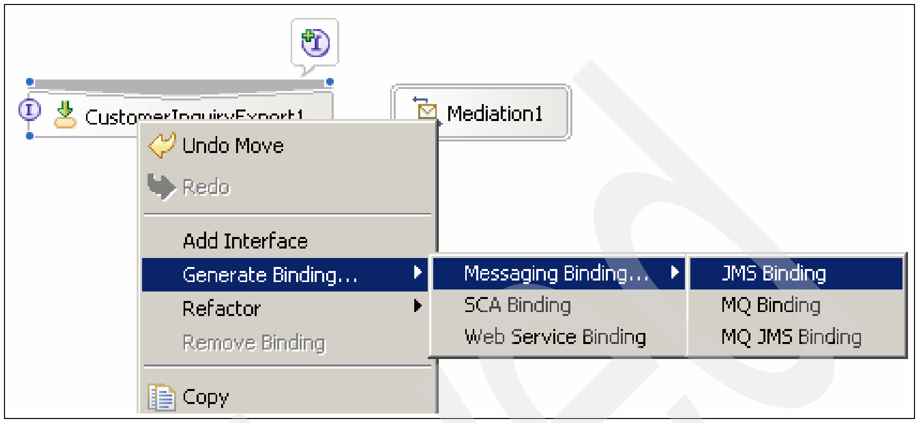
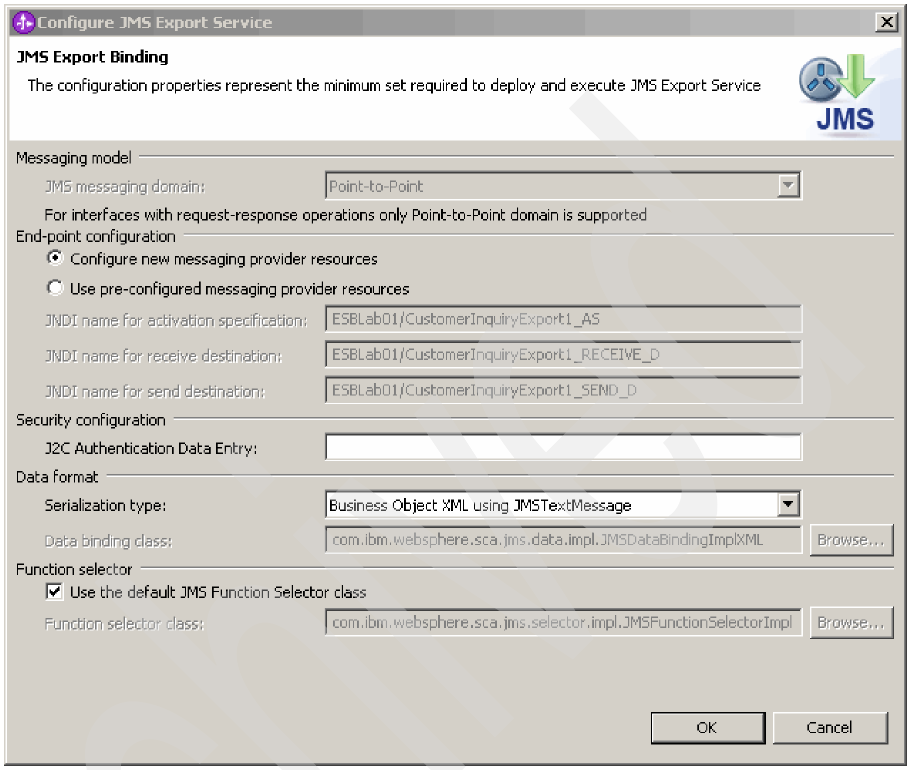


Figure 8-24 Interface context menu: generating JMS binding

4. A JMS Export Bindings dialog is displayed as shown in Figure 8-25. For Serialization type select **Business Object XML using JMSTextMessage**.



Configure JMS Export Service

JMS Export Binding

The configuration properties represent the minimum set required to deploy and execute JMS Export Service

Messaging model

JMS messaging domain: Point-to-Point

For interfaces with request-response operations only Point-to-Point domain is supported

End-point configuration

● Configure new messaging provider resources

○ Use pre-configured messaging provider resources

JNDI name for activation specification: ESBLab01/CustomerInquiryExport1_AS

JNDI name for receive destination: ESBLab01/CustomerInquiryExport1_RECEIVE_D

JNDI name for send destination: ESBLab01/CustomerInquiryExport1_SEND_D

Security configuration

J2C Authentication Data Entry:

Data format

Serialization type: Business Object XML using JMSTextMessage

Data binding class: com.ibm.websphere.sca.jms.data.impl.JMSDataBindingImplXML

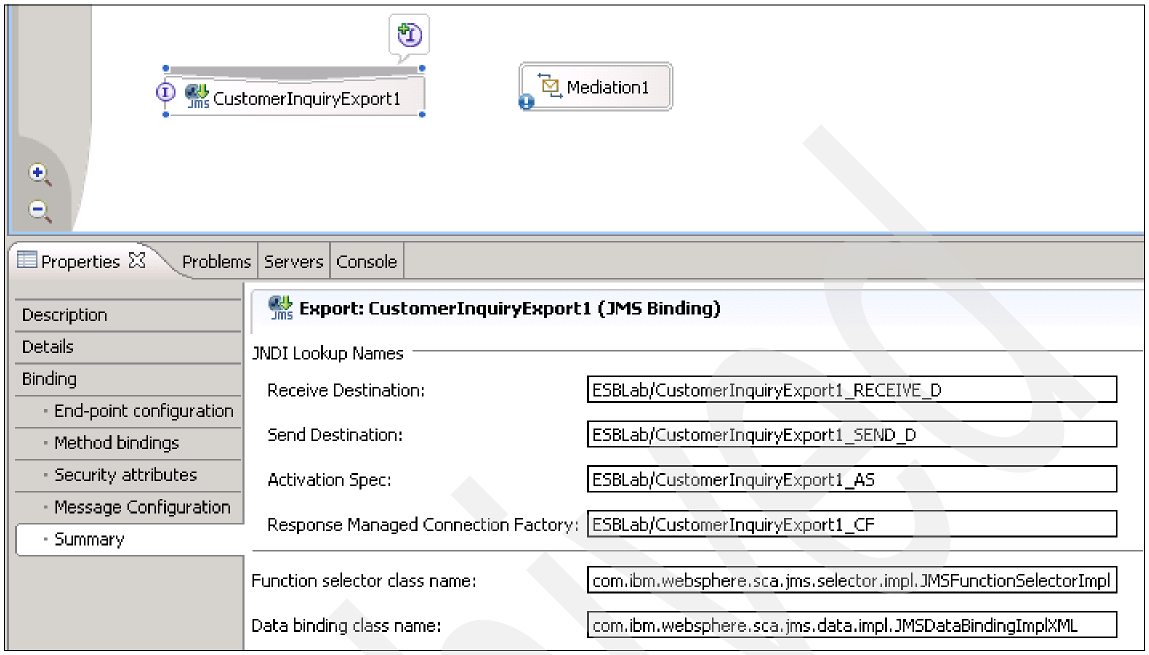
Function selector

☑ Use the default JMS Function Selector class

Function selector class: com.ibm.websphere.sca.jms.selector.impl.JMSFunctionSelectorImpl

Figure 8-25 JMS Export Bindings

5. Click **OK**. We can see the result of the Properties tab in Figure 8-26.



Export: CustomerInquiryExport1 (JMS Binding)

JNDI Lookup Names

Receive Destination:	ESBLab/CustomerInquiryExport1_RECEIVE_D
Send Destination:	ESBLab/CustomerInquiryExport1_SEND_D
Activation Spec:	ESBLab/CustomerInquiryExport1_AS
Response Managed Connection Factory:	ESBLab/CustomerInquiryExport1_CF
Function selector class name:	com.ibm.websphere.sca.jms.selector.impl.JMSFunctionSelectorImpl
Data binding class name:	com.ibm.websphere.sca.jms.data.impl.JMSDataBindingImplXML

Figure 8-26 CustomerInquiryExport1 - Summary

The objects in Figure 8-26 and Table 8-1 are JMS resources that are needed in WebSphere ESB for this application.

Table 8-1 JMS objects

JMS Resource	Function
Receive Destination	Read queue for the ESB to receive requests
Send Destination	Write queue for the ESB to write responses
Activation Spec	Provides configuration information for MDBs
Response Managed Connection Factory	Used by WebSphere ESB to construct a queue

Adding an Import Interface

Now we add the Import part of our diagram. Let us add the RegularService Interface.

1. Drag the **CUSTINQPortType** interface from the Interfaces folder and place it to the right of the Mediation1 object. A Component Creation dialog box is opened. Select **Import with Web Service Binding**, as shown in Figure 8-27.

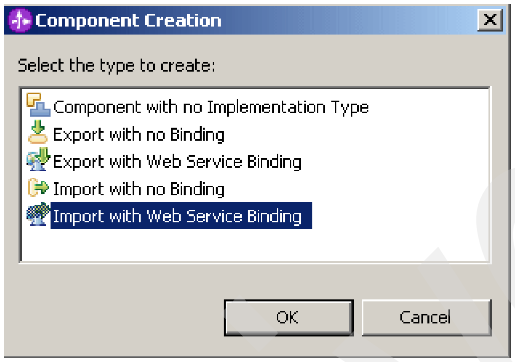
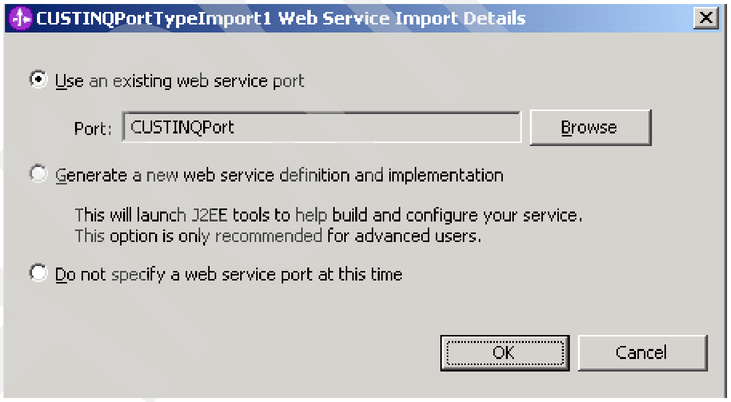

Component Creation
Select the type to create:
Component with no Implementation Type
Export with no Binding
Export with Web Service Binding
Import with no Binding
Import with Web Service Binding
OK
Cancel

Figure 8-27 Component Creation window

A dialog box is displayed as shown in Figure 8-28. Select **Use an existing web service port**. Click **Browse** and select **CUSTINQPort** and click **OK.**



(●) Use an existing web service port

Port: CUSTINQPort

( ) Generate a new web service definition and implementation

This will launch J2EE tools to help build and configure your service.
This option is only recommended for advanced users.

( ) Do not specify a web service port at this time

Figure 8-28 Select Port for a interface.

2. Draw a wire from the CustomerInquiryExport1 to the Mediation1 object. Move the mouse pointer to the right side of the CustomerInquiryExport1 object. The box becomes highlighted in orange. Hold the left mouse button down and drag the mouse over the Mediation1 object. An Add wire dialog box is presented, which asks you if you wish to continue. Check **Always create without prompt** and then click **OK**. See Figure 8-29.

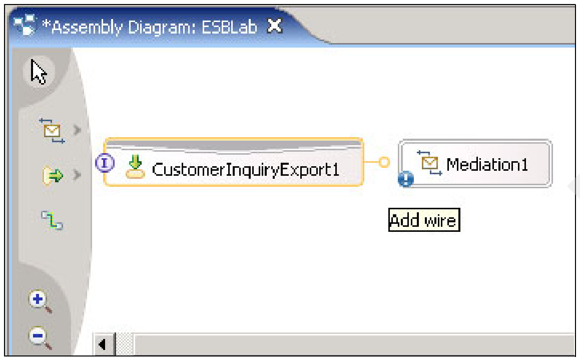


Figure 8-29 Adding a wire

3. Draw a wire from the Mediation1 object to the CUSTINQPortType object.

   The completed Assembly Diagram should appear as shown in Figure 8-30.

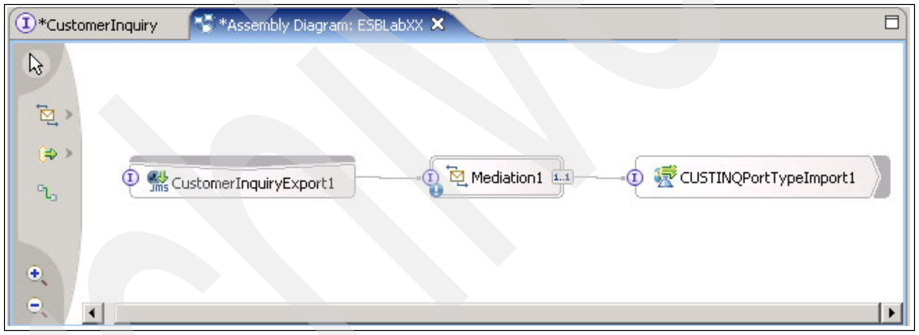


Figure 8-30 Completed Assembly Diagram window

4. Press Ctrl+s to save the Assembly Diagram.
5. Click ☒ on the editor tab to close the Assembly Diagram editor.

8.3.5 Creating a Mediation Flow

Proceed as follows:

1. Open the Assembly Diagram. This open/close procedure seems to be needed to ensure that the required menu option is available.

2. Right-click the **Mediation1** object and select **Generate Implementation**, as shown in Figure 8-31.

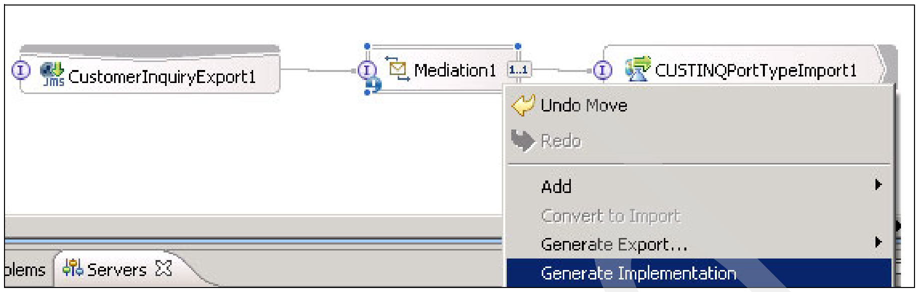


Figure 8-31 Generate implementation

3. A Generate Implementation dialog is displayed. Select project **ESBLabXX** and click **OK**.

The Mediation Flow is divided into two panes. The top pane is labeled Operations connections, and the bottom pane is labeled Mediation flow. First, the operation connections must be made. See Figure 8-32.

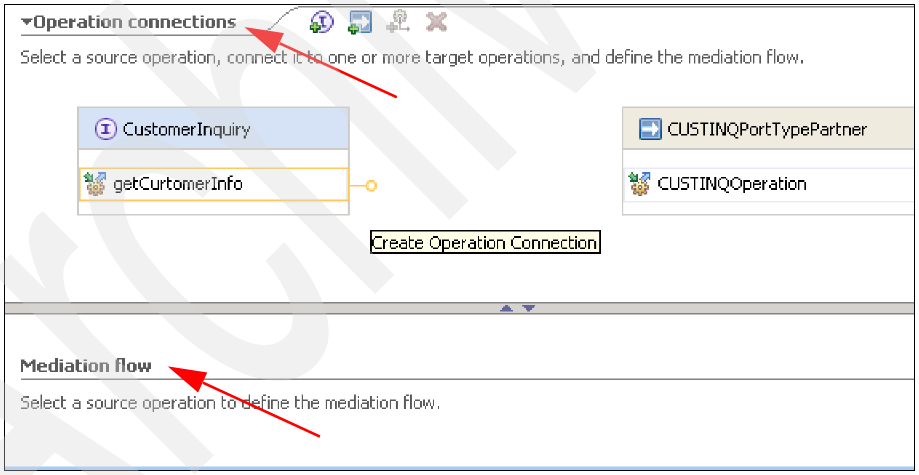

Operation connections
Select a source operation, connect it to one or more target operations, and define the mediation flow.
CustomerInquiry
getCurtomerInfo
CUSTINQPortTypePartner
CUSTINQOperation
Create Operation Connection
Mediation flow
Select a source operation to define the mediation flow.

Figure 8-32 Mediation Flow Editor - Operation connections

To create an operation connection:

a. Click in the blue area around getCustomerInfo. An orange anchor will appear. Draw a line between the getCustomerInfo method on the CustomerInquiry interface and the CUSTINQOperation method on the CUSTINQPortTypePartner. Drag the mouse pointer over to the CustReqReply method.

b. After that we can see the mediation flow on the Mediation flow panel, as shown in Figure 8-33.

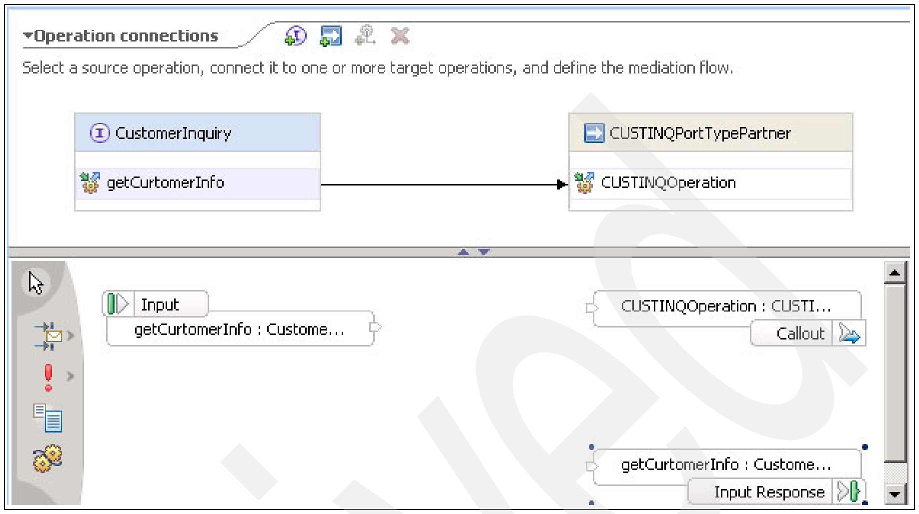

Operation connections
Select a source operation, connect it to one or more target operations, and define the mediation flow.
CustomerInquiry
getCurtomerInfo
CUSTINQPortTypePartner
CUSTINQOperation
Input
getCurtomerInfo : Custome...
CUSTINQOperation : CUSTI...
Callout
getCurtomerInfo : Custome...
Input Response

Figure 8-33 Added mediation flow

Request mediation

The request mediation flow manages the information flow from the JMS export to the Web Service import. Below the bar on the Mediation Flow editor is where the transformation is designed.

1. Click an **XML Transformation** object from the left pallet and place it on the canvas to the right of the Input terminal. Connect the terminals as shown in Figure 8-34.

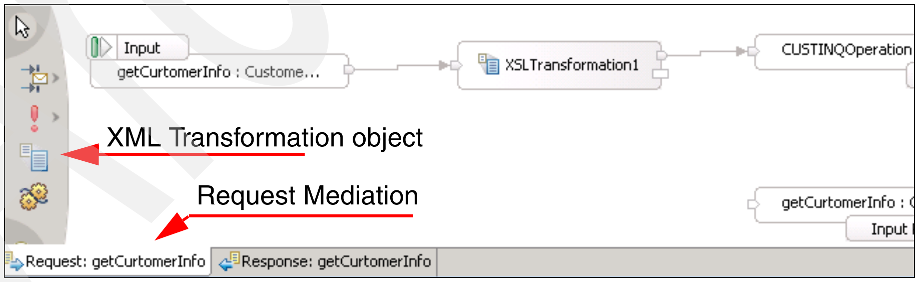
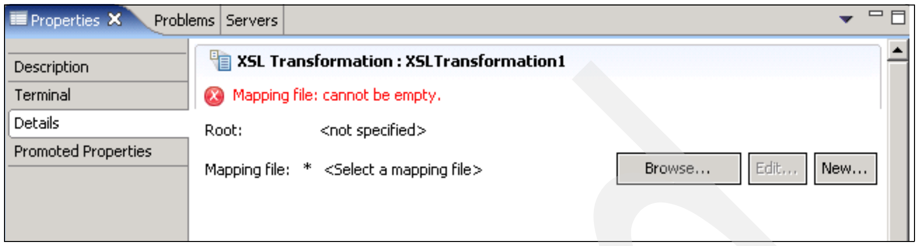


Figure 8-34 Request getCustomerinfo mediation flow

XSLT Request Transformation

The XSLT transformation takes the JMS message and transforms it to a Web Services SOAP message.

1. Select the **XSLTransformation1** object. On the bottom pane, select the **Properties** tab and then the **Details** subtab. We can see that we have to make a mapping. So, click **New**. See Figure 8-35.



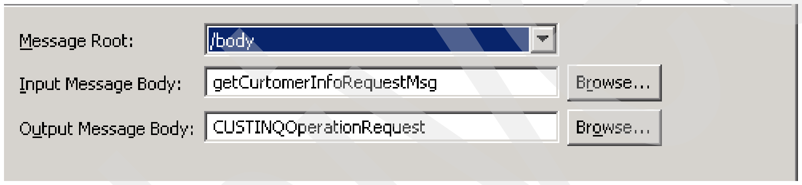
Mapping file: cannot be empty.

Root: <not specified>

Mapping file: * <Select a mapping file>

Figure 8-35 Transformation Properties - details

A New XSLT Mapping dialog is displayed, as partly shown in Figure 8-36.


Message Root: /body

Input Message Body: getCurtomerInfoRequestMsg

Output Message Body: CUSTINQOperationRequest

Figure 8-36 New XSLT Mapping

2. Click **Finish**. The XSLT editor is opened. Completely expand the **Source** and **Target** XML trees. Select **smo** in the source and **smo** in the target tree. Right-click **smo** in the target tree and select **Match Mapping**, as shown in Figure 8-37.

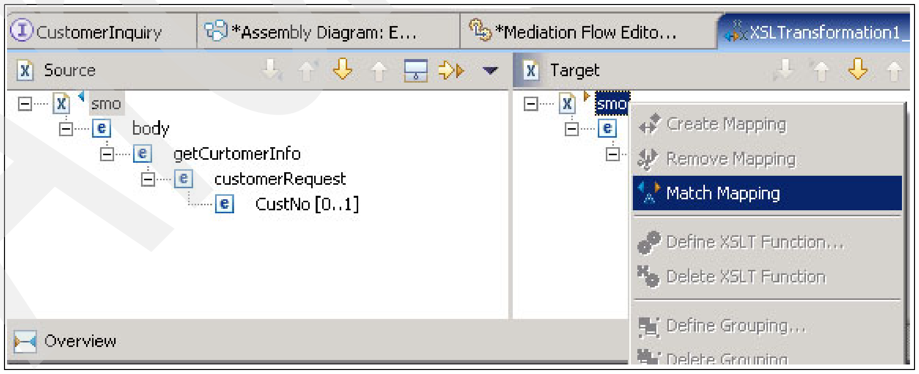
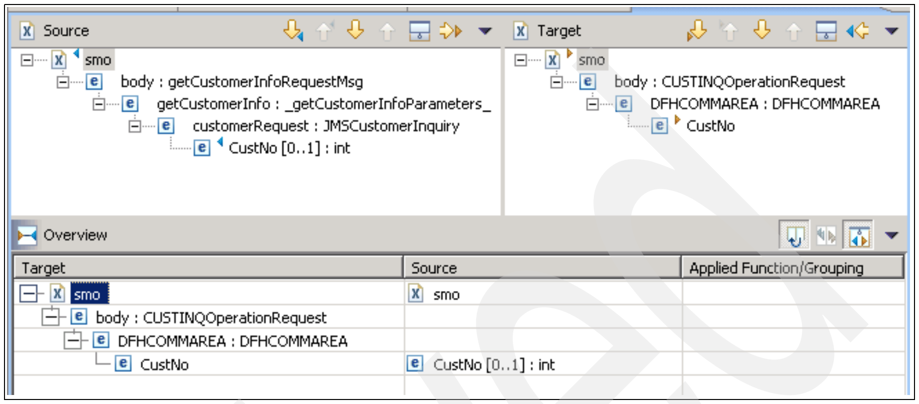


Figure 8-37 XSL Request Editor

3. Select **CustNo[0..1]** in the source tree and **CustNo** in the target tree.

4. Right-click the **CustNo** element of the target tree and select **Create Mapping** (you can use drag and drop).

   The results are shown in Figure 8-38.



Source
smo
body : getCustomerInfoRequestMsg
getCustomerInfo : _getCustomerInfoParameters_
customerRequest : JMSCustomerInquiry
CustNo [0..1] : int

Target
smo
body : CUSTINQOperationRequest
DFHCOMMAREA : DFHCOMMAREA
CustNo

Overview

Target	Source	Applied Function/Grouping
smo	smo	
body : CUSTINQOperationRequest		
DFHCOMMAREA : DFHCOMMAREA		
CustNo	CustNo [0..1] : int	

Figure 8-38 Mapping - Request transformation

5. Press Ctrl+s to save the generated XSL file.
6. Press the ☒ on the editor tab to close the XSL editor.

Response mediation

The response mediation flow manages the information flow from the Web Service import back to the JMS export. An XML transformation must be done to map the Web Service response back to the JMS XML format.

1. Click an **XML Transformation** object from the left pallet and place it on the canvas to the right of the Input terminal. Connect the terminals as shown in Figure 8-39.

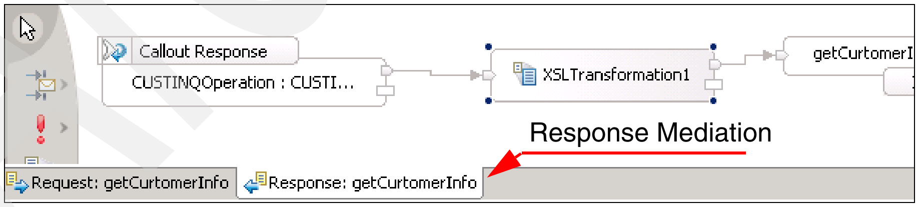
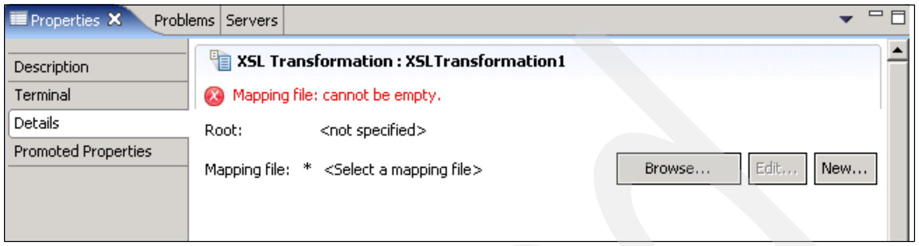


Figure 8-39 Response getCustomerinfo mediation flow

2. Click **File** → **Save All**.

XSLT response transformation

1. Select the **XSLTransformation1** object. On the bottom pane, select the **Properties** tab and then the **Details** subtab. We can see that we have to make a mapping. So, click **New**. See Figure 8-40.



XSL Transformation : XSLTransformation1

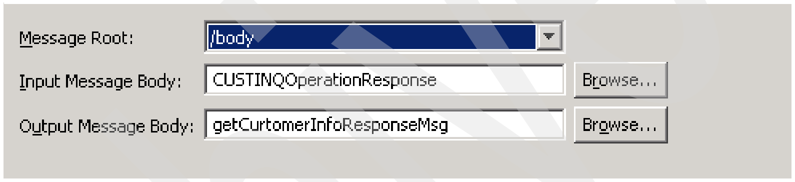
Mapping file: cannot be empty.

Root: <not specified>

Mapping file: * <Select a mapping file>

Figure 8-40 Transformation Properties - details

A New XSLT Mapping dialog is displayed, as partly shown in Figure 8-41.



Message Root: /body

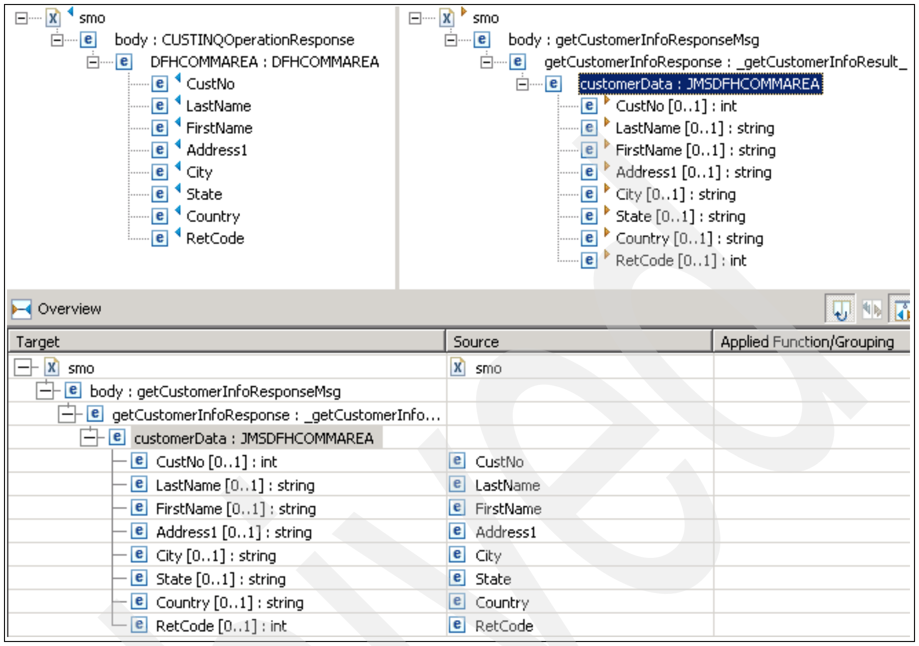
Input Message Body: CUSTINQOperationResponse

Output Message Body: getCurtomerInfoResponseMsg

Figure 8-41 New XSLT mapping

2. Click **Finish**. The XSLT editor is opened. Completely expand the **Source** and **Target** XML trees. Select **smo** in the source and **smo** in the target tree, and right-click **smo** in the target tree and select **Match Mapping**.

3. Use drag and drop objects to create the mappings as shown in Figure 8-42.



- smo
  - body : CUSTINQOperationResponse
    - DFHCOMMAREA : DFHCOMMAREA
      - CustNo
      - LastName
      - FirstName
      - Address1
      - City
      - State
      - Country
      - RetCode

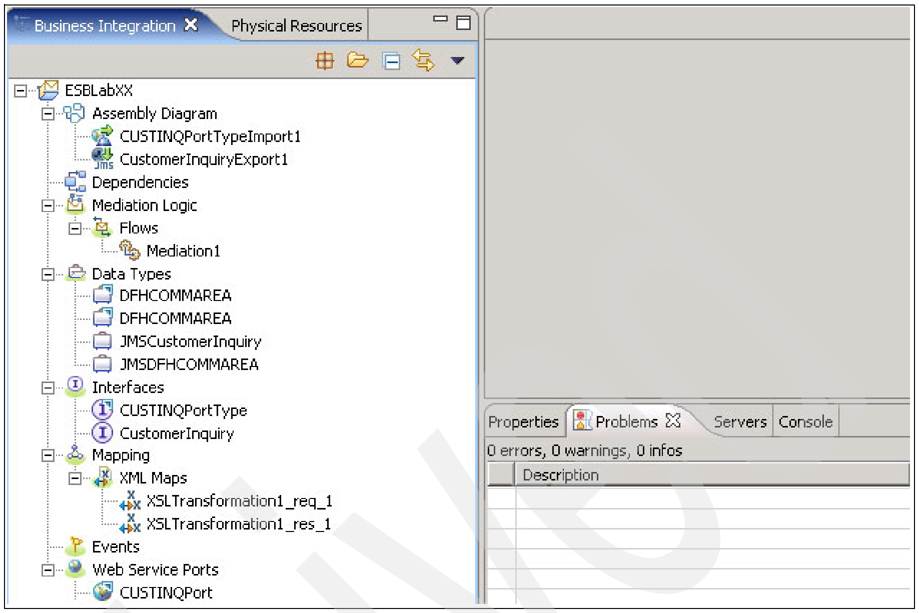
- smo
  - body : getCustomerInfoResponseMsg
    - getCustomerInfoResponse : _getCustomerInfoResult_
      - customerData : JMSDFHCOMMAREA
        - CustNo [0..1] : int
        - LastName [0..1] : string
        - FirstName [0..1] : string
        - Address1 [0..1] : string
        - City [0..1] : string
        - State [0..1] : string
        - Country [0..1] : string
        - RetCode [0..1] : int

Target	Source	Applied Function/Grouping
smo	smo	
body : getCustomerInfoResponseMsg		
getCustomerInfoResponse : _getCustomerInfo...		
customerData : JMSDFHCOMMAREA		
CustNo [0..1] : int	CustNo	
LastName [0..1] : string	LastName	
FirstName [0..1] : string	FirstName	
Address1 [0..1] : string	Address1	
City [0..1] : string	City	
State [0..1] : string	State	
Country [0..1] : string	Country	
RetCode [0..1] : int	RetCode	

Figure 8-42 Mapping - Response transformation

4. Press Ctrl+s to save generated XSL file.
5. Press the ☒ on the editor tab to close the XSL editor.
6. Save and close the Mediation1 flow.
7. Save and close the Assembly Diagram.

8. Be sure that there are no errors on Problems Tab, as shown in Figure 8-43.



0 errors, 0 warnings, 0 infos

Figure 8-43 Project finished

8.4 Deployment and test

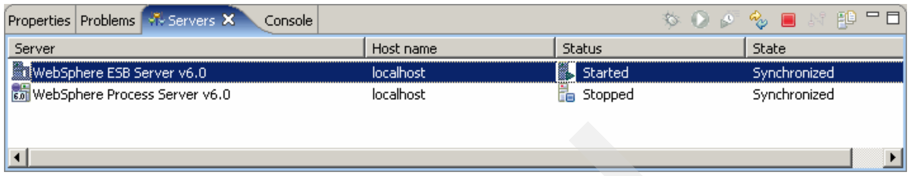
This section describes how to deploy and test the service. Normally, when deploying a service to WebSphere ESB, you must define connection factories, queues, and other JMS assets. When the service is deployed from WebSphere Integration Developer for testing, these assets are created for you temporarily. You will not see the queues defined under the JMS Queue list in the WebSphere Administrative console.

8.4.1 Deploying to the server

The Application Module and the Test Web Services must be deployed to the server. The deployment process will automatically generate the JMS resources needed.

1. Save and close any editor windows that may be open in WebSphere Integration Developer.

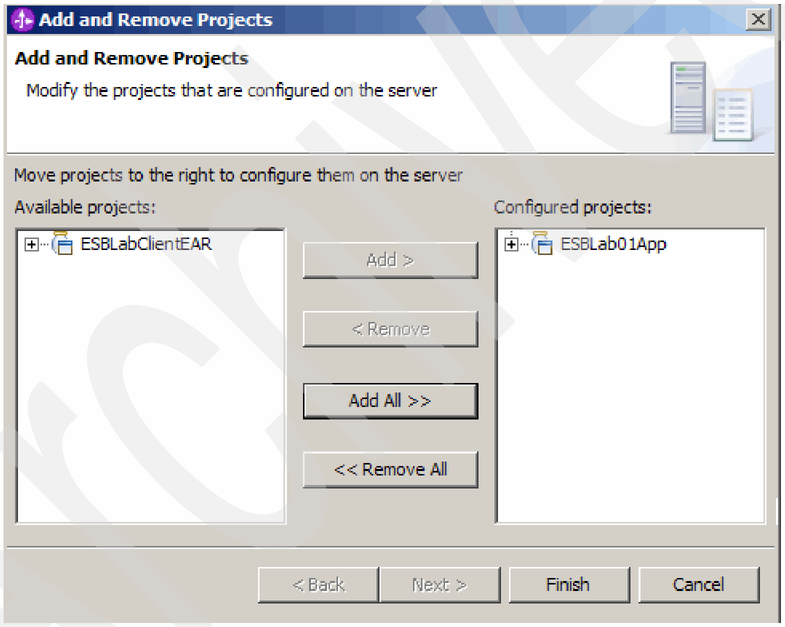
2. Click the **Servers** tab in the lower window pane, as shown in Figure 8-44.



Server	Host name	Status	State
WebSphere ESB Server v6.0	localhost	Started	Synchronized
WebSphere Process Server v6.0	localhost	Stopped	Synchronized

Figure 8-44 Servers tab

3. Right-click the **WebSphere ESB Server** and select **Add and Remove Projects**.
4. Move ESBLabXXApp to the Configured Projects side, as shown in Figure 8-45.


Add and Remove Projects

Add and Remove Projects

Modify the projects that are configured on the server

Move projects to the right to configure them on the server

Available projects:

ESBLabClientEAR

Configured projects:

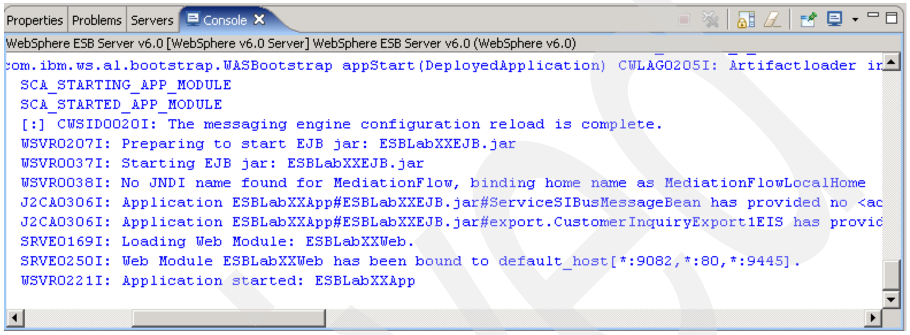
ESBLab01App

Figure 8-45 Add and Remove Projects window

5. Click **Finish**.
6. Wait until WebSphere Integration Developer finishes publishing the applications to the WebSphere ESB server; this may take a while.

Attention: If the WebSphere ESB server hangs during the publishing process, try this step from the beginning: unpublish the project and then publish it again.

The console should show that the applications have been started, as shown in Figure 8-46.



WebSphere ESB Server v6.0 [WebSphere v6.0 Server] WebSphere ESB Server v6.0 (WebSphere v6.0)

```
com.ibm.ws.al.bootstrap.WASBootstrap appStart(DeployedApplication) CWLAG0205I: Artifactloader in
  SCA_STARTING_APP_MODULE
  SCA_STARTED_APP_MODULE
  [:] CWSID0020I: The messaging engine configuration reload is complete.
  WSVR0207I: Preparing to start EJB jar: ESBLabXXEJB.jar
  WSVR0037I: Starting EJB jar: ESBLabXXEJB.jar
  WSVR0038I: No JNDI name found for MediationFlow, binding home name as MediationFlowLocalHome
  J2CA0306I: Application ESBLabXXApp#ESBLabXXEJB.jar#ServiceSIBusMessageBean has provided no <ac
  J2CA0306I: Application ESBLabXXApp#ESBLabXXEJB.jar#export.CustomerInquiryExport1EIS has provic
  SRVE0169I: Loading Web Module: ESBLabXXWeb.
  SRVE0250I: Web Module ESBLabXXWeb has been bound to default_host[*:9082,*:80,*:9445].
  WSVR0221I: Application started: ESBLabXXApp
```

Figure 8-46 Console tab with application status

8.4.2 Running the JMSTestClient

To test the application with the test client, proceed as follows:

1. In WebSphere Integration Developer, open the J2EE perspective. From the main menu bar, select **Window** → **Open Perspective** → **Other**. Select **J2EE** from the Select Perspectives dialog.

2. In the **Project Explorer**, expand **Application Client Projects** → **ESBLabClient** → **appClientModule** → **com.ibm.esb**, as shown in Figure 8-47.

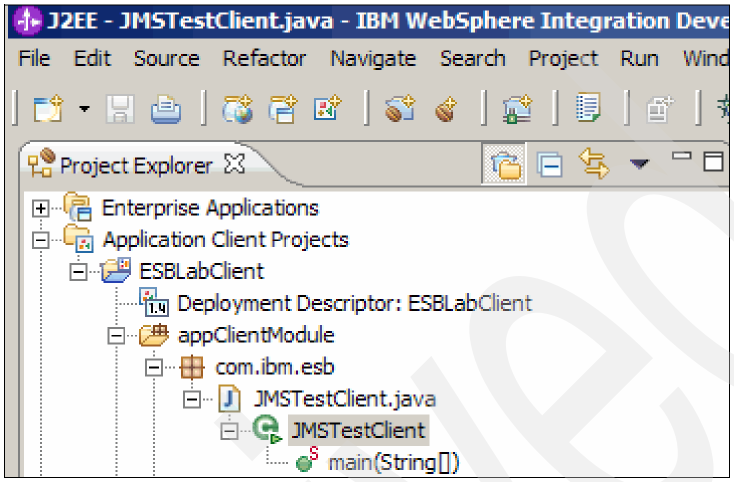
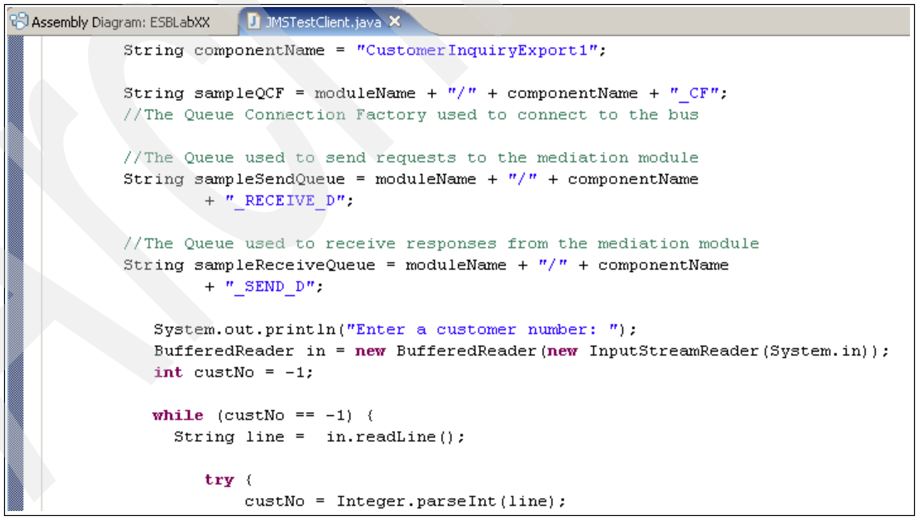

J2EE - JMSTestClient.java - IBM WebSphere Integration Deve
File Edit Source Refactor Navigate Search Project Run Wind
Project Explorer
Enterprise Applications
Application Client Projects
ESBLabClient
Deployment Descriptor: ESBLabClient
appClientModule
com.ibm.esb
JMSTestClient.java
JMSTestClient
main(String[])

Figure 8-47 Test client in Project Explorer

3. Double-click **JMSTestClient** to open it for editing. JMSTestClient.java is opened for editing, as shown in Figure 8-48.



```
        String componentName = "CustomerInquiryExport1";

        String sampleQCF = moduleName + "/" + componentName + "_CF";
        //The Queue Connection Factory used to connect to the bus

        //The Queue used to send requests to the mediation module
        String sampleSendQueue = moduleName + "/" + componentName
                + "_RECEIVE_D";

        //The Queue used to receive responses from the mediation module
        String sampleReceiveQueue = moduleName + "/" + componentName
                + "_SEND_D";

           System.out.println("Enter a customer number: ");
           BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
           int custNo = -1;

           while (custNo == -1) {
             String line =  in.readLine();

                try {
                    custNo = Integer.parseInt(line);
```

Figure 8-48 JMSTestClient open for editing

The Java code of the client is shown in Example 8-1.

Example 8-1 JMSTestClient.java

```
package com.ibm.esb;
import java.io.*;

import javax.naming.Context;
import javax.naming.InitialContext;

public class JMSTestClient {
    public static void main(String[] args) throws Exception {
        String studentNumber = "XX";
        String moduleName = "ESBLab" + studentNumber;
        String componentName = "CustomerInquiryExport1";

        String sampleQCF = moduleName + "/" + componentName + "_CF";
        //The Queue Connection Factory used to connect to the bus

        //The Queue used to send requests to the mediation module
        String sampleSendQueue = moduleName + "/" + componentName + "_RECEIVE_D";

        //The Queue used to receive responses from the mediation module
        String sampleReceiveQueue = moduleName + "/" + componentName + "_SEND_D";

           System.out.println("Enter a customer number: ");
           BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
           int custNo = -1;

           while (custNo == -1) {
             String line =  in.readLine();

                try {
                    custNo = Integer.parseInt(line);
                } catch (NumberFormatException e) {
                    System.out.println("Invalid Number: " + e.getMessage());
                    System.out.println("Enter a customer number: ");
                }
           }


        //The XML representation of an Order Business Object required by the
        //placeOrder operation on the OrderService Interface
        String message = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>";
        message += "\n<ns2:JMSCustomerInquiry xmlns:ns2=\"http://ESBLab" + studentNumber + "\">";
        message += "\n<CustNo>" + custNo + "</CustNo>";
        message += "\n</ns2:JMSCustomerInquiry>";


        Context ctx = new InitialContext();

        //Lookup the ConnectionFactory
        javax.jms.ConnectionFactory factory = (javax.jms.ConnectionFactory) ctx
                .lookup(sampleQCF);
        //Create a Connection
        javax.jms.Connection connection = factory.createConnection();

        //Start the Connection
        connection.start();
        //Create a Session
        javax.jms.Session jmsSession = connection.createSession(false,
                javax.jms.Session.AUTO_ACKNOWLEDGE);
        //Lookup the send Destination
        javax.jms.Destination sendQueue = (javax.jms.Destination) ctx
                .lookup(sampleSendQueue);
        //Create a MessageProducer
```

```
            javax.jms.MessageProducer producer = jmsSession
                    .createProducer(sendQueue);
            //Create the TextMessage that will hold our Order as text
            javax.jms.TextMessage sendMessage = jmsSession.createTextMessage();
            //Set the content of the message to be the XML defined Order
            sendMessage.setText(message);
            //Set the operation to call on the OrderService interface to be
            // placeOrder
            sendMessage.setStringProperty("TargetFunctionName", "getCustomerInfo");

            //Send the message
            producer.send(sendMessage);
            String msgId = sendMessage.getJMSMessageID();
            System.out.println("Messsage Sent:");
            System.out.println(message);

            //Lookup the receive Destination
            javax.jms.Destination receiveQueue = (javax.jms.Destination) ctx
                    .lookup(sampleReceiveQueue);
            //Create a MessageConsumer
            String filter = "JMSCorrelationID = '" + msgId + "'";
            javax.jms.MessageConsumer consumer = jmsSession
                    .createConsumer(receiveQueue,filter);


            //QueueReceiver receiver = qSession.createReceiver(replyQueue, filter);

            //Wait 30 seconds to receive the response
            javax.jms.TextMessage receiveMessage = (javax.jms.TextMessage) consumer
                    .receive(30000);
            //If we receive a response print the contents of the message to the
            // screen
            String resp = "No Message Reply Received";
            if (receiveMessage != null) {
                //Print the contents of the message.
                resp = "Message Received:\n" + receiveMessage.getText();
            }
            System.out.println(resp);

            //Close the Connection
            connection.close();

        }
    }
```

4. Press the ☒ on the editor tab to close the JMSTestClient.java file.
5. In the Project Explorer, select **JMSTestClient.java** as shown in Figure 8-47 on page 383.

6. Right-click **JMSTestClient.java** and select **Run** → **Run** from the pop-up menu, as shown in Figure 8-49.

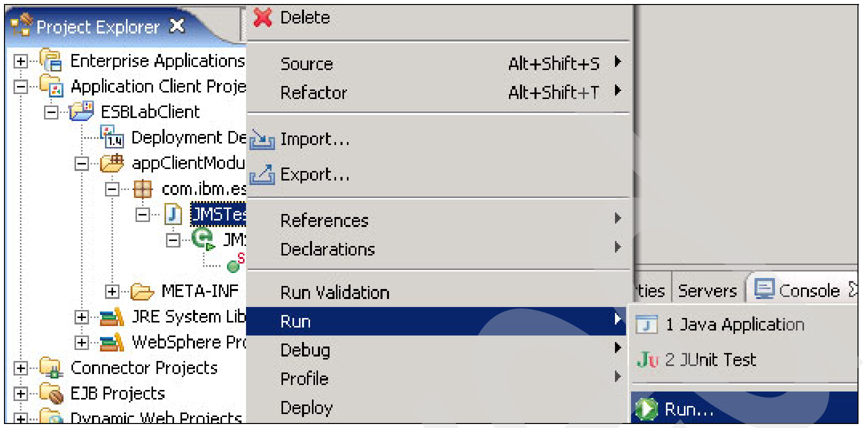
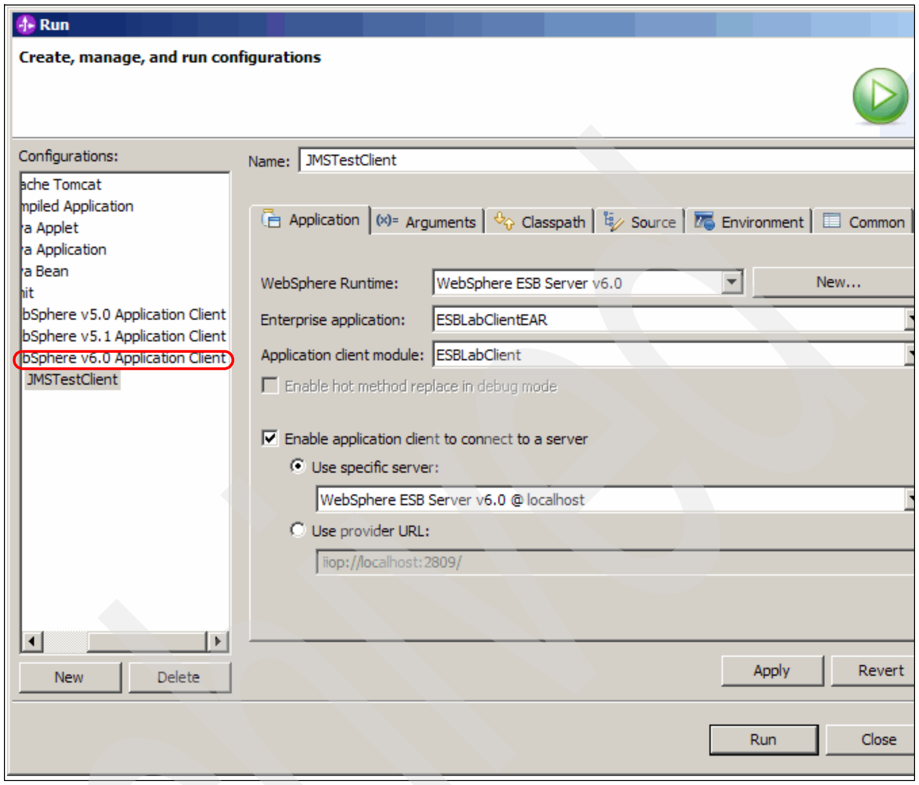


Figure 8-49 Running the test client program

7. The run dialog is displayed as shown in Figure 8-50.


Run

Create, manage, and run configurations

Configurations:
- ache Tomcat
- mpiled Application
- a Applet
- a Application
- a Bean
- it
- bSphere v5.0 Application Client
- bSphere v5.1 Application Client
- bSphere v6.0 Application Client
- JMSTestClient

Name: JMSTestClient

WebSphere Runtime: WebSphere ESB Server v6.0

Enterprise application: ESBLabClientEAR

Application client module: ESBLabClient

[ ] Enable hot method replace in debug mode

[✓] Enable application client to connect to a server

(●) Use specific server: WebSphere ESB Server v6.0 @ localhost

( ) Use provider URL: iiop://localhost:2809/

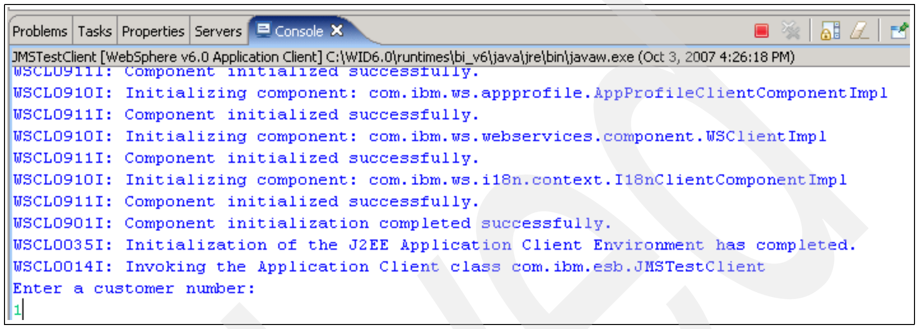
Figure 8-50 Run dialog

8. In the Configurations list, select **WebSphere 6.0 Application Client** highlighted in red, and click **New**.

9. For Enterprise application, enter `ESBLabClientEAR`. For Application client module, enter `ESBLabClient`. Check **Enable application client to connect to a server.** Select **Use specific server**. Select the server where the ESBLabXX mediation module is deployed. It is the WebSphere ESB Server.

10. Click **Apply** → **Run**.

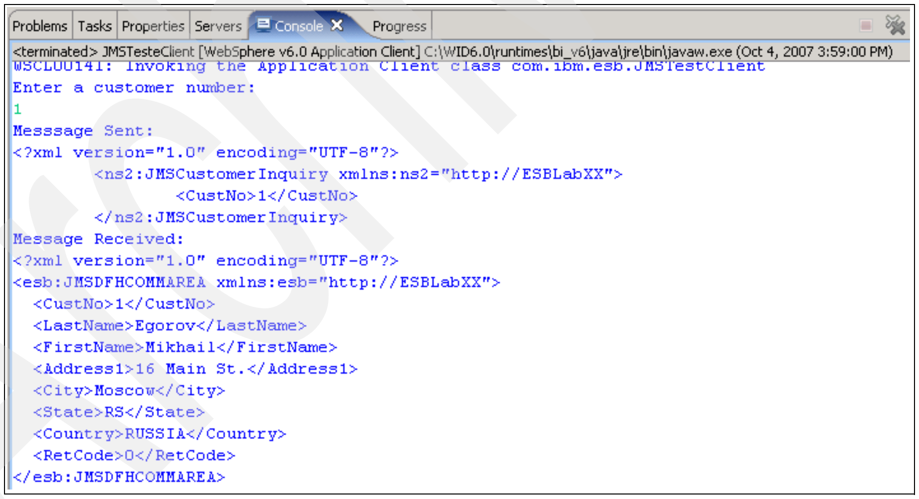
The program executes in the Console in the lower pane of WebSphere Integration Developer. A series of informational messages are written to the console. The client application then pauses for you to enter a customer number, as shown in Figure 8-51. Type a customer number between 1 and 10 when you see the prompt `Enter a customer number`.



```
JMSTestClient [WebSphere v6.0 Application Client] C:\WID6.0\runtimes\bi_v6\java\jre\bin\javaw.exe (Oct 3, 2007 4:26:18 PM)
WSCL0911I: Component initialized successfully.
WSCL0910I: Initializing component: com.ibm.ws.appprofile.AppProfileClientComponentImpl
WSCL0911I: Component initialized successfully.
WSCL0910I: Initializing component: com.ibm.ws.webservices.component.WSClientImpl
WSCL0911I: Component initialized successfully.
WSCL0910I: Initializing component: com.ibm.ws.i18n.context.I18nClientComponentImpl
WSCL0911I: Component initialized successfully.
WSCL0901I: Component initialization completed successfully.
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
WSCL0014I: Invoking the Application Client class com.ibm.esb.JMSTestClient
Enter a customer number:
1
```

Figure 8-51 Executing a Java program

11. As a result, we can see the application response as shown in Figure 8-52.



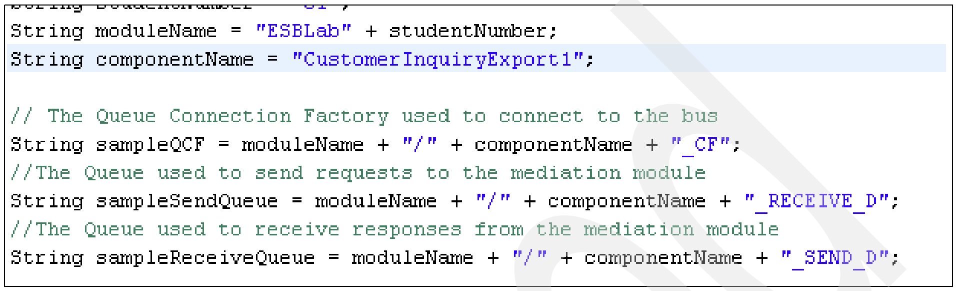
```
<terminated> JMSTesteClient [WebSphere v6.0 Application Client] C:\WID6.0\runtimes\bi_v6\java\jre\bin\javaw.exe (Oct 4, 2007 3:59:00 PM)
WSCL0014I: Invoking the Application Client class com.ibm.esb.JMSTestClient
Enter a customer number:
1
Messsage Sent:
<?xml version="1.0" encoding="UTF-8"?>
        <ns2:JMSCustomerInquiry xmlns:ns2="http://ESBLabXX">
                <CustNo>1</CustNo>
        </ns2:JMSCustomerInquiry>
Message Received:
<?xml version="1.0" encoding="UTF-8"?>
<esb:JMSDFHCOMMAREA xmlns:esb="http://ESBLabXX">
  <CustNo>1</CustNo>
  <LastName>Egorov</LastName>
  <FirstName>Mikhail</FirstName>
  <Address1>16 Main St.</Address1>
  <City>Moscow</City>
  <State>RS</State>
  <Country>RUSSIA</Country>
  <RetCode>0</RetCode>
</esb:JMSDFHCOMMAREA>
```

Figure 8-52 Executing the client - result

You can repeat with other Customer Numbers. Use the range from 1 to 10. Note that during this test the application has made a round trip to a CICS Web Service, based on the CUSTINQ program discussed earlier in this book.

8.4.3 Exploring creation of JMS queue names

The code snippet in Figure 8-53 shows the JMS queues that the client writes to and reads from. The default queue names are specified from the ESB's perspective.



```
String moduleName = "ESBLab" + studentNumber;
String componentName = "CustomerInquiryExport1";

// The Queue Connection Factory used to connect to the bus
String sampleQCF = moduleName + "/" + componentName + "_CF";
//The Queue used to send requests to the mediation module
String sampleSendQueue = moduleName + "/" + componentName + "_RECEIVE_D";
//The Queue used to receive responses from the mediation module
String sampleReceiveQueue = moduleName + "/" + componentName + "_SEND_D";
```

Figure 8-53 Code snippet from the ESB client queue names

Appendix A

The objective of this appendix is to highlight the key installation considerations and options and identify the components installed regarding the usage of IBM Data Server Developer Workbench.

This appendix contains the following:

- Supported Web servers, SOAP engines, and databases, discussed in “Supported Web servers, SOAP engines, and databases” on page 392
- Supported message protocols, discussed in “Supported message protocols” on page 393
- Mapping of the SQL and JDBC data types to XML data types, discussed in “Mapping of SQL and JDBC data types to XML data types” on page 393

Supported Web servers, SOAP engines, and databases

You can deploy Web services to several types of Web servers, use a number of different SOAP engines, and access DB2.

Supported Web servers

You can use the following Web servers:

- WebSphere Application Server, Version 6.1
- WebSphere Application Server, Community Edition, Version 1.1.*x*
- Apache Tomcat, Version 5.5

Supported SOAP engines

If you want client applications to send and receive SOAP-wrapped XML messages when communicating with Web services, you can use the SOAP engines, as shown in Table A-1.

Table A-1 Supported SOAP engines

SOAP engines	**Apache Axis 1.4**	**Apache Axis 2.0**	**Apache SOAP 2.3**	**WebSphere Application Server JSR 109**
Web servers				
Apache Tomcat	X	X	X	
WebSphere Application Server	X	X	X	X
WebSphere Application Server Community Edition	X	X	X	

You can download the Apache SOAP engines from the following locations:

- Apache Axis 1.4: http://ws.apache.org/axis/
- Apache Axis 2.0: http://ws.apache.org/axis2/
- Apache SOAP 2.3: http://ws.apache.org/soap/

Supported databases

You can use the following JCC-compliant databases:

- DB2 for Linux, UNIX, and Windows, Versions 8, 9, and 9.5
- DB2 for z/OS, Versions 8 and 9
- DB2 on iSeries®, Version 5 Release 2 and above
- Informix® Dynamic Server, Versions 10 and 11

The supported version of JCC is 3.50 (JDBC 3) and the supported Java platforms are IBM JDK 1.4 and IBM JDK 5.

Supported message protocols

Client applications can access Web Services by sending messages that use the SOAP/HTTP message protocol or three REST-like message protocols.

Client applications can perform database operations by sending requests to Web services that offer those operations. The requests are messages that are in one of the following four protocols:

- SOAP/HTTP
- REST-like HTTP GET (url-encoded)
- REST-like HTTP POST (url-encoded)
- REST-like HTTP POST (text/xml)

Mapping of SQL and JDBC data types to XML data types

The data type of a parameter or column of a result set in a message is specified using XML data types.

Table A-2 on page 394 shows the default mappings from SQL and JDBC data types to XML data types.

Table A-2 Data type mapping to XML

DB2 SQL type	Informix SQL type	JDBC type	XML type
BIGINT	INT8	java.sql.Types.BIGINT	xsd:long
CHAR FOR BIT DATA, BINARY	BYTE	java.sql.Types.BINARY	xsd:base64Binary
		java.sql.Types.BIT	xsd:short
BLOB		java.sql.Types.BLOB	xsd:base64Binary
	BOOLEAN	java.sql.Types.BOOLEAN	xsd:boolean
CHAR, GRAPHIC	CHAR	java.sql.Types.CHAR	xsd:string
CLOB, DBCLOB		java.sql.Types.CLOB	xsd:string
DATALINK		java.sql.Types.DATALINK	xsd:anyURI
DATE	DATE	java.sql.Types.DATE	xsd:date
DECIMAL, DECFLOAT	DECIMAL	java.sql.Types.DECIMAL	xsd:decimal
DOUBLE		java.sql.Types.DOUBLE	xsd:double
REAL	FLOAT	java.sql.Types.FLOAT	xsd:float
INTEGER	INTEGER	java.sql.Types.INTEGER	xsd:int
	BLOB	java.sql.Types.LONGVARBINARY	xsd:base64Binary
	CLOB, TEXT	java.sql.Types.LONGVARCHAR	xsd:string
	MONEY	java.sql.Types.NUMERIC	xsd:decimal
	SMALLFLOAT	java.sql.Types.REAL	xsd:float
SMALLINT	SMALLINT	java.sql.Types.SMALLINT	xsd:short
TIME	DATETIME HOUR TO SECOND	java.sql.Types.TIMESTAMP	xsd:time
TIMESTAMP	DATETIME YEAR TO FRACTION	java.sql.Types.TIMESTAMP	xsd:dateTime
		java.sql.Types.TINYINT	xsd:short
VARCHAR FOR BIT DATA, VARBINARY		java.sql.Types.VARBINARY	xsd:base64Binary

VARCHAR, VARGRAPHIC	VARCHAR	java.sql.Types.VARCHAR	xsd:string
		java.sql.Types.ARRAY	xsd:string
		java.sql.Types.DISTINCT	xsd:string
		java.sql.Types.JAVA_OBJECT	xsd:string
		java.sql.Types.NULL	xsd:string
		java.sql.Types.OTHER	xsd:string
		java.sql.Types.REF	xsd:string
		java.sql.Types.STRUCT	xsd:string
ROWID		100	xsd:base64Binary
XML		2009	xsd:anyType
		Any other type that is not listed	xsd:string

B

Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG247548`

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 Redbooks form number, SG247548.

Using the Web material

The additional Web material that accompanies this book includes the following files:

File name	*Description*

CICSWSIN.zip	Project Interchange file for the scenario in “Accessing CICS from WebSphere using Web Services over HTTP” on page 3.
DWS.zip	Zip file with all contents of the scenario discussed in “Accessing DB2 from WebSphere using Web Services” on page 85. Details on the contents of this zip file can be found in “Additional material” on page 138.
IMS.zip	Zip file with all the contents of the scenario discussed in Chapter 4, “Accessing IMS transactions as Web Services” on page 139. Details on the contents of this zip file can be found in “Additional material” on page 203.
CICSWSOUT.zip	Zip file with all the contents of the scenario discussed in Chapter 5, “Accessing a Web Service in WebSphere Application Server from CICS” on page 205. Details on the contents of this zip file can be found in “Additional material” on page 237.
WMB.zip	Zip file with all the contents of the scenario discussed in Chapter 7, “Using WebSphere Message Broker for connectivity” on page 293. Details on the contents of this zip file can be found in “Additional material” on page 347.
WESB.zip	Zip file with the client application for the WESB sample application explained in Chapter 8, “Using WebSphere Enterprise Service Bus for connectivity” on page 349.

System requirements for downloading the Web material

To use the samples included in the additional material, you need to use tools described in each chapter. It is recommended to have at least 1.5 GB RAM to use any of the tools discussed.

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder. Make sure you import the artefacts discussed into the tools from this directory.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 401. Note that some of the documents referenced here may be available in softcopy only.

- *WebSphere for z/OS V6 Connectivity Handbook*, SG24-7064
- *Implementing CICS Web Services,* SG24-7206
- *SOA Transition Scenarios for the IBM z/OS Platform*, SG24-7331
- *Developing Web Services Using CICS, WMQ, and WMB*, SG24-7425

Other publications

These publications are also relevant as further information sources:

- *IBM WebSphere Developer for System z Prerequisites,* SC31-6352
- *IBM WebSphere Developer for System z Host Planning Guide,* SC31-6599
- *IBM WebSphere Developer for System z Installation Guide,* SC31-6316
- *IBM WebSphere Developer for System z Host Configuration Guide,* SC31-6930

Online resources

These Web sites are also relevant as further information sources:

- WebSphere - infocenter

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp`
- WebSphere Application Server V6.1 - detailed system requirements

  `http://www-1.ibm.com/support/docview.wss?uid=swg27007678`

- WebSphere Application Server V6.1 - product support page

  `http://www-306.ibm.com/software/webservers/appserv/zos_os390/support`
- CICS Transaction Server V3.1 - Infocenter

  `http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp?topic=/com.ibm.cics.ts31.doc/prod/home.html`
- CICS Transaction Server V3.1- detailed system requirements

  `http://www-1.ibm.com/support/docview.wss?uid=swg27006364`
- CICS Transaction Server V3.1 - product support page

  `http://www-306.ibm.com/software/htp/cics/tserver/support/`
- WebSphere MQ - support page

  `http://www-306.ibm.com/software/integration/wmq/support/`
- WebSphere MQ v6 for z/OS - detailed system requirements

  `http://www-1.ibm.com/support/docview.wss?uid=swg27006267`
- Websphere Developer for System z - product documentation

  `http://www-306.ibm.com/software/awdtools/devzseries/library/`
- Websphere Developer for System z - detailed system requirements

  `http://www-306.ibm.com/software/awdtools/devzseries/sysreqs/`
- Websphere Developer for System z - product support page

  `http://www-306.ibm.com/software/awdtools/rdz/support/index.html`
- WebSphere Integration Developer - detailed system requirements

  `http://www.ibm.com/support/docview.wss?uid=swg27006409`
- WebSphere Integration Developer - product support page

  `http://www-306.ibm.com/software/integration/wid/support/`
- WebSphere Integration Developer and WebSphere Enterprise Service Bus - infocenter

  `http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp`
- WebSphere Enterprise Service Bus - detailed system requirements

  `http://www.ibm.com/support/docview.wss?uid=swg27006912`
- WebSphere Enterprise Service Bus - product support page

  `http://www-306.ibm.com/software/integration/wsesb/support/`

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

Index

IBM

Redbooks

Integrating Back-end Systems with WebSphere Application Server on z/OS through Web

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Integrating Back-end Systems with WebSphere Application Server on z/OS through Web Services

Service-oriented scenarios

ESB scenarios

Integration with CICS, IMS, and DB2

Service Oriented Architecture (SOA) requires a new way of looking at integration between subsystems. The primary subsystems of interest on z/OS are CICS, IMS, and DB2. There are many ways to integrate WebSphere Application Server on z/OS with those subsystems.

In this IBM Redbooks publication, we focus on Web Services integration scenarios and scenarios using an Enterprise Service Bus (ESB) solution. In the IBM Redbooks publication *WebSphere for z/OS V6 Connectivity Handbook*, SG24-7064, we elaborate on both Web Services and J2C scenarios.

The scenarios in this book are described from a development perspective. In the Additional material, a workspace is included for most scenarios.

SG24-7548-00 ISBN 0738431060