Enabling z/OS Applications for SOA

Learn how to use patterns to help you modernize your applications

Use our samples of a selection of patterns to get started

Understand integration and modernization

Alex Louwe Kooijmans
Takehiko Akimoto
Naveed Jedd
Wilbert Kho
Subhajit Maitra
Roy Panting
Ulrich Seelbach

ibm.com/redbooks
Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (June 2009)

This edition applies to a large variety of software versions and releases. Refer to the respective chapters for exact information regarding product versions and releases.
---

## Contents

**Notices** ................................................................. xi  
**Trademarks** ............................................................. xii

**Preface** ................................................................. xiii  
The team that wrote this book ........................................ xiii  
Become a published author ............................................ xvii  
Comments welcome ...................................................... xvii

**Chapter 1. Introduction** .............................................. 1  
1.1 Objectives ................................................................ 2  
1.2 Challenges to application modernization ...................... 2  
1.2.1 Malleability of existing applications .................... 2  
1.2.2 Extensibility and availability of skills .................. 3  
1.2.3 Multitude of platforms and middleware ................. 3  
1.3 Case for application modernization ............................ 4  
1.4 Styles of application modernization ............................ 4  
1.5 The case for application modernization on System z ...... 5  
1.6 SOA best practices .................................................. 6  
1.7 Critical success factors .......................................... 6  
1.8 How this book is organized ...................................... 7  
1.9 The audience for this book ...................................... 7

**Chapter 2. Analysis, design, and architecture** .................. 9  
2.1 IBM SOA strategy and the SOA lifecycle ..................... 11  
2.2 Choosing a development and design approach ................ 13  
2.3 Hosting the SOA infrastructure on System z ............... 14  
2.4 SOA infrastructure components .................................. 14  
2.4.1 Interaction Services ........................................... 15  
2.4.2 Process Services ............................................... 15  
2.4.3 Information Services ......................................... 15  
2.4.4 Partner Services .............................................. 16  
2.4.5 Business Application Services ............................. 16  
2.4.6 Access Services ............................................... 16  
2.4.7 Enterprise Service Bus (ESB) .............................. 16  
2.4.8 Business Innovation and Optimization Services ....... 18  
2.4.9 Development Services ........................................ 18  
2.4.10 IT Service Management ..................................... 18  
2.4.11 Infrastructure Services ...................................... 19  
2.5 Integrating existing mainframe applications in an SOA .. 19
2.5.1 The discovery phase ........................................ 19
2.5.2 Enterprise transformation strategy: Improve, adapt, and innovate . 20

Part 1. SOA application enablement patterns for z/OS ....................... 25

Chapter 3. Accessing data on z/OS as a service .......................... 27
3.1 Accessing VSAM data as a service ................................ 28
  3.1.1 Initial state ........................................... 28
  3.1.2 Current architecture .................................. 29
  3.1.3 Solution strategy ..................................... 29
  3.1.4 Solutions .............................................. 30
3.2 Accessing DB2 data as a service .................................. 39
  3.2.1 Initial state ........................................... 39
  3.2.2 Solution strategy ..................................... 40
  3.2.3 Current architecture .................................. 40
  3.2.4 Solutions .............................................. 41
3.3 Accessing IMS data as a service .................................. 46
  3.3.1 Initial state ........................................... 47
  3.3.2 Current architecture .................................. 47
  3.3.3 Solution strategy ..................................... 48
  3.3.4 Solutions .............................................. 49

Chapter 4. Accessing programs on z/OS as a service .................... 55
4.1 Accessing a CICS program on z/OS as a service .................... 56
  4.1.1 Initial state ........................................... 56
  4.1.2 Current architecture .................................. 56
  4.1.3 Solution strategy ..................................... 58
  4.1.4 Solutions .............................................. 59
4.2 Accessing a complex CICS application as a service ................ 76
  4.2.1 Initial state ........................................... 76
  4.2.2 Current architecture .................................. 77
  4.2.3 Solution using code refactoring. ............................ 77
4.3 Accessing IMS programs as a service ................................ 80
  4.3.1 Initial state ........................................... 80
  4.3.2 Current architecture .................................. 80
  4.3.3 Solution strategy ..................................... 81
  4.3.4 Solutions .............................................. 82
4.4 Accessing a CICS application as a composite service ............ 94
  4.4.1 Initial state ........................................... 94
  4.4.2 Current architecture .................................. 94
  4.4.3 Solution .............................................. 95
4.5 Accessing DB2 stored procedures on z/OS as a service ............ 97
  4.5.1 Initial state ........................................... 97
  4.5.2 Solution .............................................. 97
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6</td>
<td>Accessing presentation logic as a service</td>
<td>99</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Initial state</td>
<td>99</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Current architecture</td>
<td>99</td>
</tr>
<tr>
<td>4.6.3</td>
<td>Solution: Using HATS as a service enabler</td>
<td>100</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Integrating external services in existing z/OS applications</td>
<td>105</td>
</tr>
<tr>
<td>5.1</td>
<td>Integrating external Web Services into CICS applications</td>
<td>106</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Initial state</td>
<td>106</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Current architecture</td>
<td>106</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Solution strategy</td>
<td>107</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Solutions</td>
<td>107</td>
</tr>
<tr>
<td>5.2</td>
<td>Integrating external Web services with IMS applications</td>
<td>113</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Initial state</td>
<td>113</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Current architecture</td>
<td>113</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Solution strategy</td>
<td>114</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Solutions</td>
<td>114</td>
</tr>
<tr>
<td>5.3</td>
<td>Integrating external services into batch applications</td>
<td>117</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Initial state</td>
<td>117</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Solution strategy</td>
<td>117</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Solutions</td>
<td>118</td>
</tr>
<tr>
<td>Part 2</td>
<td>Samples</td>
<td>127</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Exposing VSAM data as Web Services</td>
<td>129</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>130</td>
</tr>
<tr>
<td>6.2</td>
<td>The JZOS toolkit and library</td>
<td>130</td>
</tr>
<tr>
<td>6.3</td>
<td>Creating the generic VSAM access bean</td>
<td>131</td>
</tr>
<tr>
<td>6.4</td>
<td>Creating the Web service implementation</td>
<td>134</td>
</tr>
<tr>
<td>6.5</td>
<td>Creating and deploying the Web service</td>
<td>136</td>
</tr>
<tr>
<td>6.6</td>
<td>Testing the service</td>
<td>140</td>
</tr>
<tr>
<td>6.7</td>
<td>Creating a Java client for the VsamAccess service</td>
<td>144</td>
</tr>
<tr>
<td>6.8</td>
<td>Creating a wrapper service</td>
<td>150</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>Exposing DB2 data as a service</td>
<td>159</td>
</tr>
<tr>
<td>7.1</td>
<td>Overview</td>
<td>160</td>
</tr>
<tr>
<td>7.2</td>
<td>Using Data Web Services (DWS)</td>
<td>160</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Data Web Services</td>
<td>161</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Introduction to the example</td>
<td>161</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Setting up the connection to DB2</td>
<td>162</td>
</tr>
<tr>
<td>7.2.4</td>
<td>Creating the SQL statement</td>
<td>163</td>
</tr>
<tr>
<td>7.2.5</td>
<td>Creating a Web service from the query</td>
<td>164</td>
</tr>
<tr>
<td>7.2.6</td>
<td>Installing the Web service in WebSphere Application Server</td>
<td>168</td>
</tr>
<tr>
<td>7.2.7</td>
<td>Testing the Web service</td>
<td>169</td>
</tr>
<tr>
<td>7.2.8</td>
<td>Additional activities</td>
<td>172</td>
</tr>
</tbody>
</table>
Chapter 11. Building composite CICS services using the Service Flow Feature
   11.1 Introduction .................................................. 308
       11.1.1 Introducing this sample ............................ 309
       11.1.2 Objective ............................................. 311
       11.1.3 Prerequisite ......................................... 311
   11.2 Creating a Service Flow project ................................. 311
       11.2.1 Understanding the Service Flow project structure ... 324
       11.3 Defining the GetCustomerInfo flow step .................. 325
   11.4 Defining the OrderItem flow step ................................ 329
       11.4.1 Importing BMS .................................... 330
       11.4.2 Defining a host connection ......................... 331
       11.4.3 Recording 3270 terminal screens .................... 333
   11.5 Mapping the Data Flow ...................................... 352
       11.5.1 variableAssignInput node .......................... 353
       11.5.2 variableAssignOutput Node ......................... 358
       11.5.3 Assign node ........................................ 362
   11.6 Generating the runtime artifacts ............................ 366
   11.7 Deploying generated artifacts ................................ 379
   11.8 Testing the Service Flow application ......................... 380
   11.9 Summary .................................................... 383
   11.10 Additional material ......................................... 383

Chapter 12. Integrating an external Web service with a CICS application...
   12.1 Introduction to the sample .................................. 387
   12.2 Prerequisites ................................................ 387
   12.3 Extracting the WSDL from the existing Web Service ............ 387
   12.4 Generating the CICS artifacts ................................ 395
   12.5 Updating the generated CICS service requester program ...... 404
   12.6 Creating a test CICS client program ........................ 408
   12.7 Deploying the application to z/OS ........................... 409
   12.8 Testing the CICS Service requester program .................. 413
   12.9 Handling complex CICS COMMAREA structures .................. 415
   12.10 Summary .................................................... 417

Chapter 13. Integrating an external Web service into an IMS application ..
   13.1 Prerequisites ................................................ 420
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.2 Outbound Web Services overview</td>
<td>420</td>
</tr>
<tr>
<td>13.3 Development and deployment steps</td>
<td>422</td>
</tr>
<tr>
<td>13.3.1 Step 1: Modify or create the IMS application for callout requests</td>
<td>423</td>
</tr>
<tr>
<td>13.3.2 Step 2: Define an OTMA destination routing descriptor for callout messages</td>
<td>425</td>
</tr>
<tr>
<td>13.3.3 Step 3: Generate artifacts with RDz</td>
<td>426</td>
</tr>
<tr>
<td>13.3.4 Step 4: Create a connection bundle for the callout function</td>
<td>445</td>
</tr>
<tr>
<td>13.3.5 Step 5: Create a correlator file for the callout function</td>
<td>448</td>
</tr>
<tr>
<td>13.3.6 Step 6: Deploy artifacts to IMS SOAP Gateway</td>
<td>450</td>
</tr>
<tr>
<td>13.3.7 Step 7: Start the callout thread</td>
<td>452</td>
</tr>
<tr>
<td>Chapter 14. Using DB2 as a Web service requester</td>
<td>455</td>
</tr>
<tr>
<td>14.1 Overview</td>
<td>456</td>
</tr>
<tr>
<td>14.2 Introduction to the example</td>
<td>458</td>
</tr>
<tr>
<td>14.3 Creating the SOAP request document</td>
<td>458</td>
</tr>
<tr>
<td>14.4 Creating the SQL to invoke the SOAP UDF</td>
<td>459</td>
</tr>
<tr>
<td>14.5 Parsing the SOAP response message</td>
<td>462</td>
</tr>
<tr>
<td>14.6 Creating a wrapper UDF</td>
<td>464</td>
</tr>
<tr>
<td>14.6.1 Invoking the wrapper UDF</td>
<td>465</td>
</tr>
<tr>
<td>14.6.2 Retrieving multiple columns from the SOAP response</td>
<td>465</td>
</tr>
<tr>
<td>Chapter 15. Using an external Web service in batch applications</td>
<td>469</td>
</tr>
<tr>
<td>15.1 Introduction</td>
<td>470</td>
</tr>
<tr>
<td>15.2 Using DB2 outbound Web services</td>
<td>470</td>
</tr>
<tr>
<td>15.3 Calling a Java client from Enterprise COBOL</td>
<td>471</td>
</tr>
<tr>
<td>Part 3. Appendixes</td>
<td>479</td>
</tr>
<tr>
<td>Appendix A. Defining data used in samples</td>
<td>481</td>
</tr>
<tr>
<td>JCL for creating and loading the catalog VSAM file</td>
<td>482</td>
</tr>
<tr>
<td>DDL to create the DEPT sample table</td>
<td>483</td>
</tr>
<tr>
<td>Appendix B. Additional material</td>
<td>485</td>
</tr>
<tr>
<td>Locating the Web material</td>
<td>485</td>
</tr>
<tr>
<td>System requirements for downloading the Web material</td>
<td>485</td>
</tr>
<tr>
<td>Using the Web material</td>
<td>486</td>
</tr>
<tr>
<td>Exposing VSAM data as Web Services</td>
<td>486</td>
</tr>
<tr>
<td>Exposing data as a service by using IBM WebSphere Message Broker</td>
<td>486</td>
</tr>
<tr>
<td>Exposing DB2 data as a service</td>
<td>486</td>
</tr>
<tr>
<td>Exposing IMS data as a service</td>
<td>487</td>
</tr>
<tr>
<td>Exposing an IMS application as a Web service</td>
<td>487</td>
</tr>
<tr>
<td>Integrating an external Web service with a CICS application</td>
<td>487</td>
</tr>
<tr>
<td>Integrating an external Web service into an IMS application</td>
<td>487</td>
</tr>
<tr>
<td>Using DB2 as a Web service requester</td>
<td>488</td>
</tr>
</tbody>
</table>
Using an external Web service in batch applications .................. 488

Related publications ................................................................. 489
IBM Redbooks publications ...................................................... 489
Other publications ................................................................. 489
How to get IBM Redbooks publications ................................. 490
Help from IBM ................................................................. 490

Index ................................................................. 491
Notices

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

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:

- alphaWorks®
- CICS®
- DB2®
- DRDA®
- IBM®
- Language Environment®
- MQSeries®

- OS/390®
- Parallel Sysplex®
- RAA®
- RACF®
- Rational®
- Redbooks®
- Redbooks (logo)®
- S/390®
- System z®
- WebSphere®
- z/OS®
- zSeries®

The following terms are trademarks of other companies:

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

Hibernate, Interchange, and the Shadowman logo are trademarks or registered trademarks of Red Hat, Inc. in the U.S. and other countries.

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

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

Expression, Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation 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

The objective of this IBM® Redbooks® publication is to demonstrate ways of enabling existing applications and data on z/OS® for integration into service-oriented architecture (SOA). The focus is on solutions using Web Services as the underlying technology for this integration.

This book describes a variety of patterns that can be applied to enabling z/OS applications for SOA. We explain many of these patterns by means of a sample. Each sample is included in the additional material of this book. For instructions to access the additional material, go to Appendix B, “Additional material” on page 485.

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, and specializes in SOA technology and solutions on System z®. He also specializes in application modernization and transformation on z/OS. Previously, he worked as a Client IT Architect in the Financial Services sector with IBM in The Netherlands, advising financial services companies about 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 clients starting up with Java™ and WebSphere® on System z. From 1997 to 2000, Alex completed a previous assignment with the ITSO, managing various IBM Redbooks publications projects and delivering workshops around the world in the areas of WebSphere, Java, and e-business technology on System z. Before 1997 Alex held a variety of positions in application design and development, product support, and project management, mostly in relation to the IBM mainframe.

Takehiko Akimoto is an IT Architect based in Sapporo Hokkaido, Japan. He has more than 20 years of experience in banking systems on mainframes. He has been in charge of a regional bank in Sapporo, Japan, as an IT Architect, IT Specialist, project leader, and application programmer. His areas of expertise include a banking system, especially application, infrastructure, integration, and information architecture on z/OS and IMS. He leads teams in making the
transition from existing applications on IMS to SOA, connecting the applications, and reusing them in an SOA-style architecture for clients. He holds a Bachelors of Engineering degree from Shibaura Institute of Technology, Japan.

**Naveed Jedd** is a Solution Architect working with Satyam Computer Services Limited, in Bangalore, India. He leads the SOA on Mainframe and Application Transformation Service offerings team at Satyam. He has over 12 years of experience in a variety of computer disciplines, ranging from MVS on S/390® mainframes, to the latest System z, focusing on the IBM SOA-based products. His experience includes architecting, designing, and developing solutions in mainframe technology areas that include major z/OS subsystems, such as CICS®, DB2®, and MQ. The majority of his experience has been in the banking domain. His current responsibilities are z/OS Systems programming and increasing his mainframe competency, in particular SOA on the mainframe, and he offers technology consultancy and solutions at Satyam. He holds a Engineering degree in Electronics and Communications from the University of Madras, Chennai, India.

**Wilbert Kho** is an IBM Consulting Certified Software IT Specialist based in the United States. He works on the Americas Rational® Software for System z team. His expertise includes Enterprise Modernization tools, such as Rational Asset Analyzer, Rational Developer for System z, and Rational Business Developer. He has over 25 years experience in the IT industry and has a Bachelor of Science in Electrical Engineering from the University of the Philippines, a Masters of Science Computer Science degree from Northern Illinois University, and an M.B.A. from the University of Phoenix.

**Subhajit Maitra** is a Senior IT Specialist working for IBM Advanced Technical Support based in Hartford, CT. His areas of expertise are WebSphere Message Broker and WebSphere MQ on System Z. He has over 16 years experience in information technology, as a developer, designer, and architect on various projects. He has previously worked with the ITSO in building workshops and delivering them worldwide. He holds a Masters degree in Computer Science from Jadavpur University, in Kolkata, India.

**Roy Panting** is a System z IT Architect based in the United States. He has more than 25 years of experience in mainframe and distributed computing. He has worked for multiple companies in the role of software engineer, systems programmer, project manager, and system support. He has provided presales and postsales support for multiple program products. He has written customization and installation guides for a variety of application products. He holds a Bachelor of Science degree from SUNY at Potsdam and a Masters of Science degree from the University of New Haven. His areas of expertise include z/OS, Linux on System z, and SOA.
Ulrich Seelbach is an IT Architect based in Frankfurt, Germany. He has been with IBM for twelve years. He started working on IBM mainframes in 1999 and never looked back. His main areas of expertise include everything related to Java on z/OS and its major subsystems, including WebSphere for z/OS, DB2, WebSphere MQ, and CICS Transaction Server. In the last eight years, he has supported various clients, mainly in the banking and insurance sectors, by designing and implementing solutions based on these technologies. By his birth name, Ulrich Gehlert, he has coauthored DB2 for z/OS and OS/390: Ready for Java, SG24-6435, and Java Application Development for CICS, SG24-5275. He holds a degree in Computer Science from University of Erlangen, Germany.
Thanks to the following people for their contributions to this project:

Maryela Weihrauch  
IBM Distinguished Engineer, DB2 z/OS Development - Solutions Architect, IBM Software Group

Michael Schenker  
Data Web Services and SOA Team Lead, IBM Software Group

Josef Klitsch  
Information Management, IBM Software Group

Richard M Conway  
International Technical Support Organization, Poughkeepsie Center

Gary Mazo, William Alexander, and Jonathan Gellin  
IBM Software Group, Rational
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 books in one of the following ways:

- Use the online Contact us review IBM Redbooks publication 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
Introduction

The benefits that can be derived from a service-oriented architecture (SOA) are accepted by both the Information Technology (IT) community and the business community that it serves. The high-level Qualities of Service (QoS) associated with running enterprise applications on the z/OS system are also acknowledged and proven in the technical community.

This book assumes that you have already embraced SOA and are looking to leverage your existing z/OS applications so that these applications form part of your SOA. A typical challenge you will face is knowing where to start. This decision can be specially daunting in light of the myriad of applications ranging from simple to complex that were developed many years ago and have been maintained by multiple software professionals over many years.

As you identify candidate services from your existing application portfolio, you will face the additional challenge of melding the existing application interfaces to the service interfaces. You will also determine that certain applications might need to be re-architected for reasons related to the proper separation of concerns, elimination of redundant business logic, reduction of application complexity, and so forth.

We term this set of application renovation activities application modernization.

In this chapter, we talk about the objectives, scope, and organization of this book and describe the target audience.
1.1 Objectives

In this book, we review the impetus behind SOA, a strategy to attain SOA, and in the context of z/OS applications, approaches that you can take to transform or modernize these applications to enable optimal business agility and flexibility. We also talk about common scenarios involved in application modernization and identify patterns for those scenarios. Lastly, we provide examples for several of the scenarios and patterns and show you the approach that we took toward integrating existing applications into an SOA environment.

1.2 Challenges to application modernization

There are challenges to application modernization, which we can broadly categorize into three areas:

- Malleability of existing applications
- Extensibility and availability of skills
- Multitude of platforms and middleware

1.2.1 Malleability of existing applications

Many z/OS applications, especially those applications that have been around for many years, were not written with reuse in mind and are classified as what the industry terms as monolithic applications. These applications are typically tightly coupled with strong cohesion and incur high costs of maintenance and enhancement, with the costs taking a large share of the limited IT budget. Industry experts have estimated that 70% to 80% of an organization's IT budget is typically spent on maintenance.

While these applications encode a lot of business knowledge, they are hard to repurpose to fulfill new business requirements. Often, this difficulty in repurposing applications leads to decisions about whether to rewrite the applications so that you can make them more flexible and easy to extend. But rewriting might not always be the best option, when you consider the amount of rewriting that needs to take place and the repercussions to interdependent applications during the rewrite. You also face the dilemma of having to consider the operating platform and the programming language so that you will not be in a similar “boxed-in” position when the newly rewritten applications become older applications.
1.2.2 Extensibility and availability of skills

We have three distinct challenges in the area of skills.

One challenge is that developers are usually specialized, and their skills are not portable across projects because they often only program on one platform (for example, the CICS developers do not create Web programs and the Web developers do not create CICS applications). They also use different tools, programming languages, and processes. Thus, there is a lot of fragmentation and suboptimal governance in an IT development organization.

The second challenge is that a lot of business savvy people do not know that new technologies are hard and costly and that it takes a long time to retrain the existing development staff to effectively adopt these new technologies.

The third challenge is that the “baby boomers,” the people who can handle the existing application platforms, are disappearing. They cannot be replaced by new generations who do not understand or have interest in the older technologies, which forces IT managers to consider outsourcing, stabilizing, end of life planning, rewrite, and so forth. All of these choices are difficult and costly.

Developing software is slow, repetitive, and error prone. Many existing applications are written in RPG, COBOL, PL/I, 4GL, and so forth. New applications require Java, Java Enterprise Edition, and other skills related to Web 2.0 technologies. The development paradigm usually differs between the procedure-oriented existing languages as compared to the object-oriented Java language. Many times, retraining a COBOL programmer to become a Java developer might not be an option due to the high costs associated with retraining and the possibility that the COBOL programmer might not be able to become a Java programmer. Furthermore, business pressures to deliver results quickly might force you to avoid retraining.

1.2.3 Multitude of platforms and middleware

With mergers and acquisitions, the need for business information to be used in new and consolidated ways presents a unique challenge in that you now have to deal with disparate, heterogeneous computing platforms and middleware. These consolidations might result in a fragmentation of skills needed to support these platforms and middleware and add to the complexity of interfaces among the disparate applications, which tends to slow down development.

These challenges often translate to high costs, hamper the ability to quickly respond to business requirements and deliver solutions, and often lead to compromising into building what the development team is capable of building rather than what the business asks for and needs.
1.3 Case for application modernization

Despite the challenges we have enumerated, there are many reasons to transform and modernize existing applications:

- They run the business and contain critical business logic that is unique, difficult, and costly to replicate.
- Modernizing existing applications allows you to reuse critical business assets and leverage an investment that you have already made.
- These systems are not necessarily broken, but they are restrictive and limit business agility and flexibility.
- It is absolutely feasible (doable) to modernize these systems in a timely, cost-effective manner.

1.4 Styles of application modernization

There are four styles for application modernization or transformation:

- **Transform the user experience**
  Change the current character-based user interface to an interface that gives the user a consolidated Web look and feel, which is fairly simple and cost-effective to undertake and does not change the underlying code.

- **Transform the application connectivity**
  Change the way that users or client applications can connect with the existing application, which can working with Java connectors and creating Java wrappers. It can also involve exposing Web service interfaces.

- **Transform the application architecture**
  Change the way that the application is constructed by making it more modular and making it easier to compose business services with the right level of granularity. These services represent mission-critical application components.

- **Transform the application code**
  This transformation involves the actual process of transforming the underlying application code to a new, modern language that supports deployment to any platform using whatever user interface (UI) is deemed necessary (Web, rich client platform (RCP), or text user interface (TUI) or traditional “green screen” for any type of processing (online or batch) or service creation. It is a strategic move forward.
1.5 The case for application modernization on System z

There are many reasons to modernize your applications on System z, which include:

► Many business rules are currently implemented on System z with a quality that cannot be matched by any other platform. It is the combination of the System z hardware and operating system (OS) attributes and the transaction and database managers that have matured over many decades that drive this extraordinary quality.

► Those business rules can be made more reusable with a relatively low investment while adhering to the principles of an SOA. Tools and technologies are in place for all major software products on z/OS.

► Modernized applications on z/OS will deliver the promises of SOA through better business and IT alignment, better time to market, and better manageability while keeping the original qualities of service (QoS) that are associated with System z.

Note on service granularity:

When designing service components, consider the granularity of the service. Typically, the discussion revolves around coarse-grained and fine-grained services. The differences in these services are:

► Coarse-grained service:
  – The goal is to reduce the number of invocations related to a service.
  – This service results in fewer services with one service implementing more functions.
  – The client in general will need to pass more information to the service and might need more intelligence.

► Fine-grained service:
  – This service produces better reusability, because each service maps to a single function.
  – The result is more services that are smaller and have less functionality.
  – The client in general passes less information to a service but will need more service invocations to accomplish its task.
1.6 SOA best practices

In general, you perform appropriate planning to identify the correct services. You might want to consider the use of methodologies, such as Component Business Modeling (CBM) or Service-Oriented Modeling and Architecture (SOMA). It is important to note that not every component, module, or program interface has to become a service. You might also want to consider technologies that facilitate the management and use of services, such as Enterprise Service Bus (ESB) or process choreography. Last but not least, SOA governance must be addressed.

In particular to System z, there is a tendency to think of the System z applications as service providers. In fact, System z applications can be service consumers. In SOA, every component can be both provider and consumer; restricting System z components as only providers pushes up unnecessary service choreography.

Another thing to note is that layering or the proper separation of concerns is good architecture applicable to System z components. When creating new or renovating existing CICS or IMS applications, consider a layering approach by separating programs and modules into presentation, business logic, and data access parts. Lastly, a good test bed and process are important for all SOA work, which is worth mentioning for the System z because of a dearth in this area for many System z organizations.

1.7 Critical success factors

Frequently, the critical success factors have more to do with business than with technology:

- Businesses must be motivated to embrace change through the adoption of more dynamic processes and a desire to optimize business through better metrics or Key Performance Indicators (KPIs).
- Businesses must establish an SOA governance process that focuses on the lifecycle of services and composite applications so that decision rights for the development, deployment, and management of new services are rightly established and that the proper monitoring and reporting of governance results exist.
- Businesses must take an incremental approach to projects that are prioritized by expected business benefits.
- Businesses must be willing to fund both infrastructure upgrades and new technology in support of SOA.
1.8 How this book is organized

This book is organized into two parts:

- In Part 1, “SOA application enablement patterns for z/OS” on page 25, we discuss a comprehensive range of patterns from which you can choose to accomplish an SOA modernization task on z/OS. The patterns are organized in the following groups:
  - Integrating data on z/OS in SOA, which is discussed in Chapter 3, “Accessing data on z/OS as a service” on page 27
  - Integrating existing z/OS applications into SOA, which is discussed in Chapter 4, “Accessing programs on z/OS as a service” on page 55
  - Integrating external services with existing z/OS applications, which is discussed in Chapter 5, “Integrating external services in existing z/OS applications” on page 105
- In Part 2, “Samples” on page 127, we describe samples of a selection of patterns. The following samples are included in this book:
  - Exposing VSAM data as Web Services
  - Exposing data as a service by using IBM WebSphere Message Broker
  - Exposing DB2 data as a service
  - Exposing IMS data as a service
  - Exposing an IMS application as a Web service
  - Building composite CICS services using the Service Flow Feature
  - Integrating an external Web service with a CICS application
  - Integrating an external Web service into an IMS application
  - Using DB2 as a Web service requester
  - Using an external Web service in batch applications

1.9 The audience for this book

This book is written for both the IT Architect or solution designer and the IT Specialist. Part 1, “SOA application enablement patterns for z/OS” on page 25, might be of particular interest to the IT architect, because it is written from an architectural point of view and is a starting point for SOA-enabling z/OS applications. Part 2, “Samples” on page 127, might be of particular interest to the IT implementation specialist, because it provides technical details of solution approaches and how-to implementation steps.
Chapter 2. Analysis, design, and architecture

Organizations that run the IBM System z platform have unique business requirements. Their commitment to this highly available server platform demonstrates their need for an environment that can support high-volume transaction processing with demanding batch windows and large, critical application portfolios.

More commercial transactions are processed on IBM mainframes than on any other platform. It is estimated that there are approximately 200 billion lines of COBOL code running on mainframes today with several billions of lines of code added annually. Rewriting or replacing this functionality involves considerable time, expense, and risk. An alternative to the rip-and-replace approach is to modernize IT systems using service-oriented architecture (SOA) as a strategic means to extend the value of existing mainframe assets.

Making the transition to an SOA offers significant benefits including reusing the investment in the application assets that support the business. This chapter presents an architectural overview of how an SOA implementation can be effectively achieved.
In this chapter:

- In 2.1, “IBM SOA strategy and the SOA lifecycle” on page 11, we present the SOA concepts and architectural principles.

- In 2.2, “Choosing a development and design approach” on page 13, we present the three general approaches to SOA analysis and design.

- In 2.3, “Hosting the SOA infrastructure on System z” on page 14, we present the benefits of deploying the SOA infrastructure on the mainframe.

- In 2.4, “SOA infrastructure components” on page 14, we present the building blocks necessary to support an SOA environment.

- Finally, in 2.5, “Integrating existing mainframe applications in an SOA” on page 19, we provide a strategy for determining what System z assets might be candidates for an SOA implementation and provide three transformation styles for both a tactical and strategic SOA integration.
2.1 IBM SOA strategy and the SOA lifecycle

The SOA lifecycle is the framework of the IBM SOA strategy. As Figure 2-1 shows, the SOA lifecycle consists of four stages:

- **Model**: Use modeling tools to define the business process at a business level and model the actual services that will be part of an assembled, composite application.

- **Assemble**: Assemble the individual services and write the code that is needed to implement the business rules for the application. Preexisting services can be reused, and new services can be developed, or both.

- **Deploy**: Deploy the services to runtime environments, such as transaction management engines, such as WebSphere Application Server, CICS, IMS, and so forth. Use integration components primarily and enterprise service bus (ESB) to link together the various services that are needed for the composite application.

- **Manage**: Implement the management infrastructure for monitoring and managing the services and the service infrastructure, which includes not only IT management tools, but also business management and monitoring tools to measure actual business activities.

Because this approach is a lifecycle, it implies a closed loop, meaning that at the end of the cycle, the Manage stage feeds information back to the Model stage. Results from the runtime management tool can be fed into the modeling tools to provide feedback for refinement of the business processes that are being instantiated into services and composite applications.

Underlying the four phases of the lifecycle is the function of governance. Governance of the SOA implementation is critical to the integrity of the
architecture. Governance can ensure the consistency of the service development strategy and adherence to the policies and procedures of the SOA implementation.

The SOA lifecycle is not an entirely new concept. Many application development models, including those models for developing mainframe applications, have evolved the same concepts. However, there are several aspects of SOA that make this particular lifecycle unique:

- SOA involves modeling business processes, not just the modeling of the applications themselves. With SOA, the business process is modeled first, and the results of the model are fed to additional development tools that produce the Business Process Execution Language (BPEL). The BPEL artifacts are then passed to other tooling that is designed to enable the construction of composite applications that consist of multiple, orchestrated services.

- SOA applications are assembled from multiple services that can be logically connected using orchestration and process management tools.

- In a full SOA implementation, the individual services are deployed to a runtime, just like a “traditional” transactional system. However, the composite applications are represented in and “executed” from a flow engine, and that runtime engine takes care of calling the services in the proper sequence, based upon what is specified in the BPEL (the input to the flow engine).

- While the logical linkage between the services is represented by the flows that execute under the control of a flow engine, the physical linkage between deployed services is typically implemented in an Enterprise Service Bus (ESB). The ESB is the abstraction layer that is designed to eliminate the point-to-point connectivity between specific services.

- In previous application architectures (mainframe and other), monitoring is traditionally only done at the IT level. Organizations are often concerned about the number of transactions per second being executed, what the user response time is, application server utilization, and so forth. In the SOA lifecycle, there is a keen interest in the business performance. For example, Bank A might be interested in how many new accounts were opened last week, or how many ATM withdrawals occurred, rather than what the average response time was for the ATM transactions. This kind of business information is fed into the business modeling tool to refine the business process.

- The sharp focus on governance by IBM is unique in this model. While architectural governance (including security) has always been an important facet of IT and architectural management, it is particularly critical in SOA due to the loosely coupled nature of services.
2.2 Choosing a development and design approach

One of the more interesting philosophical discussions in SOA is “Where do we start?” This discussion is particularly lively when mainframe assets are involved, because few companies are willing to simply discard working applications that contain valuable intellectual capital. The general approaches to SOA analysis and design tend to center on three philosophies:

- **Top-down**: This top-down approach is usually considered the “purist” approach to SOA. The top-down approach begins with analysis and understanding of the business environment and the services that make up the functions of the business. The business model is decomposed into the elementary components that represent business services. Those business services are eventually represented by IT services. This analysis is accompanied by the documentation of the processes that link together the various business services.

  You can follow the IBM *Component Business Modeling (CBM)* to discover the business architecture. When a model is developed, the strategically important parts of the business architecture can be identified and fed into the process of service modeling so that IT services can be created from them.

- **Bottom-up**: The bottom-up approach involves exposing existing applications as services, which are then used to create new, composite applications that are developed using techniques, such as business modeling and development with process orchestration tools.

  When a bottom-up approach is used, concerns often emerge regarding the granularity of the services. With existing mainframe systems, an interesting situation often arises: Older mainframe systems are frequently well suited for SOA, because the applications themselves already represent discrete business functions.

- **Meet-in-the-middle**: The meet-in-the-middle approach is a compromise that employs techniques from both top-down and bottom-up methodologies. It affirms that there is considerable value in existing assets that needs to be reused when appropriate, and it also utilizes top-down service identification and decomposition techniques.

This summary shows the need to address SOA analysis and design from both the top and the bottom. In a mainframe environment, where there is a pre-existing inventory of mature, high-performance business applications, it only makes sense to try to reuse as much as possible.
2.3 Hosting the SOA infrastructure on System z

As organizations adopt SOA as the guiding architectural framework for development of enterprise applications, the newly deployed services quickly become business-critical components of the application infrastructure. Therefore, services must be treated as business-critical components of the application infrastructure and must be deployed on a robust, scalable, secure, and high performance platform. Along with the services, the SOA infrastructure middleware and tools must also reside on such a platform. The mainframe, z/OS in particular, is the premier IBM computing environment for providing ultra-high qualities of service for enterprise applications.

The advantages of using System z and z/OS for SOA can be seen in three broad categories:

- **Quality of service (QoS):** A framework that incorporates SOA capabilities exploits well proven System z features, such as high scalability, availability, reliability, and security. System z clustering is provided through Parallel Sysplex® technology and workload management.

- **Core system transaction capabilities for SOA:** The source of most services that service requesters call upon are likely core systems, such as CICS and IMS transactions. The positioning of these systems within a System z environment means that performance is enhanced.

- **Total Cost of Ownership:** The centralized architecture of the mainframe helps to avoid many of the costs associated with distributed systems. The total cost of ownership for System z is also reduced with new virtualization technology and System z specialty engines.

2.4 SOA infrastructure components

The IBM SOA reference architecture, which is shown in Figure 2-2 on page 15, defines the necessary building blocks to support an SOA environment. This section includes a brief description of the following core components of the logical architecture.
2.4.1 Interaction Services

The Interaction Services component provides the user interface to the SOA application. A key principle of SOA is the abstraction of application layers. In this case, the application’s user interface (UI) is exposed in a separate layer from the business logic. Interaction services are commonly thought of as the portal layer, because the UI for many SOA applications is provided by a portal, although this component is not mandatory. The portal not only provides abstraction for the UI, but it also provides a standard set of services to give the user a customized, personalized user experience.

2.4.2 Process Services

A key attribute of an application in an SOA is that it is often a “composite application,” meaning an application that is constructed from several discrete services, all connected via a type of orchestration engine. The Process Services components provide the orchestration and workflow services that are required to meld multiple services into a single, composite business application. This application can be a single transaction, or it can be a series of transactions joined together into a business process.

2.4.3 Information Services

The Information Services component provides SOA-based access to the data repositories through techniques, such as accessing stored procedures as Web
services, providing standardized interfaces to non-relational data repositories, and other access mechanisms that return Information As A Service (IAAS). Because data or information is not executable, infrastructure is required to expose that data to applications, such as SOAP, that use SOA standards.

2.4.4 Partner Services

The Partner Services component is the interface to outside business partners. Exposing enterprise services to the outside world and invoking external services pose challenges to the integrity and security of the SOA. Services must be exposed in a manner that maintains the security and scalability of the service, which makes the usage patterns for the service much less predictable and controllable than if they were being accessed exclusively in-house.

2.4.5 Business Application Services

Newly developed services reside in the Business Application Services component. These new services are generally deployed on servers, such as IBM WebSphere Application Server, which is a transaction manager for Java 2 Enterprise Edition (J2EE™) applications. The WebSphere Application Server can be used for full-function applications (presentation, business logic, and data logic), or it can host business services that have the other functions abstracted to other portions of the reference architecture.

2.4.6 Access Services

The Access Services component is particularly interesting to most mainframe clients. It is the linkage to existing applications, both on and off the mainframe. The functionality here provides the connectivity to applications on IMS, CICS, SAP®, PeopleSoft®, and so forth, using various connector and adapter technologies.

2.4.7 Enterprise Service Bus (ESB)

This section includes a description of the supporting components of the SOA foundation logical architecture used in support of the core components. At the center of this reference architecture is the ESB (Figure 2-2 on page 15). Surrounding the ESB are the key services needed to support the SOA runtime environment: Interaction Services, Process Services, Information Services, Partner Services, Business Application Services, and Access Services. Along with those core services, the surrounding boxes describe the other functions that are necessary to support the services before, during, and after their deployment.
The ESB provides an infrastructure that removes the direct connection dependency between service consumers and providers. Consumers connect to the ESB and not to the provider that actually implements the service. The ESB provides the abstraction layer between the service requester and the service provider.

The IBM definition of an ESB is that of an architectural construct rather than a specific product. The ESB supports four primary functions. The ESB handles routing messages between services, removing the direct one-to-one relationship between endpoints. It also handles the conversion of transport protocols between the requester and the service, for example, SOAP to MQ, FTP to EXCI, and so forth. The ESB handles the transformation of message formats, for instance, the transformation of XML to binary. And, the ESB handles events from disparate sources. Events are received from the ESB endpoints and correlated to trigger new events based upon decisions in the ESB.

The connectivity between applications is one of the key benefits of implementing an SOA. The ESB establishes the connection between services without requiring a unique connection. Figure 2-3 shows the evolution of application connectivity.

**Figure 2-3  Evolution of the SOA connectivity challenge**
2.4.8 Business Innovation and Optimization Services

While the IT infrastructure is monitored and managed through the IT service management components, what about the business services? An organization can ensure proper levels of system and application performance, but what about business performance? The Business Innovation and Optimization Services component provides the functionality to effectively monitor and record what is happening in the enterprise from a purely business perspective. Business Innovation and Optimization Services exist in the architecture to help capture, encode, analyze, and iteratively refine the business design. The services also include tools to help simulate the business design. The results are used to predict the effect of the design, including the changes that the design has on the business.

2.4.9 Development Services

Development Services encompass the entire suite of architecture tools, development tools, visual composition tools, assembly tools, methodologies, debugging aids, instrumentation tools, asset repositories, discovery agents, and publishing mechanisms that are needed to construct an SOA-based application. Modeling consists of the modeling of the business process and the modeling of the actual services and the business logic within them. The tools might be higher level tools suitable for business analysts and architects or lower level tools, such as those tools used for object-oriented development. The output from these tools consists of artifacts, such as Uniform Modeling Language (UML) models, actual application source code in a variety of languages from Java to COBOL, and also markup languages, including XML and Business Process Execution Language (BPEL).

2.4.10 IT Service Management

The SOA infrastructure components consist primarily of middleware software. An SOA implementation requires adequate levels of monitoring and the management of the middleware and applications to ensure proper levels of system performance and security. The IT Service Management component in the SOA reference architecture includes those monitoring, management, and security tools. The IT Service Management component is particularly important in an SOA-based system, because the infrastructure that is required to support an SOA is significantly more complex than traditional mainframe transaction systems, such as CICS, which are relatively self-contained. The SOA has more components, making the monitoring and management tasks more complex.
2.4.11 Infrastructure Services

At the base of the SOA is the platform for deployment, which includes the deployment of hardware and software for the actual business services and service infrastructure. A production environment usually requires the highest degree of quality of service (QoS), and the Infrastructure Services component provides the QoS. Services at this layer can include operating system functions, security, and hardware functions.

2.5 Integrating existing mainframe applications in an SOA

Most mainframe users have a keen interest in the method that is used to integrate existing applications into an SOA, which generally means that they intend to expose existing transactions or data via a service interface. Exposing applications as services is the primary goal of the bottom-up SOA design model. But if there are drawbacks to the bottom-up style, why pursue that model rather than simply rewriting existing applications? Clients are often concerned that if their existing applications are written in COBOL or PL/I, they need to eliminate them. However, many studies have shown that significant savings can be realized by reusing existing code rather than rewriting. An often-quoted statistic states that it is up to five times more expensive to rewrite an application than to reuse existing code to achieve the same functionality. Reuse becomes a very attractive strategy to harvest existing code assets and save money during an SOA implementation.

The IBM enterprise transformation strategy is a path that you can follow to assist you in integrating mainframe applications into an SOA.

2.5.1 The discovery phase

Enterprise transformation often starts with a discovery phase. Many existing applications lack documentation, or the documentation is out-of-date, or it is in someone’s head and that person has left the company. In such situations, before modernization activities begin, it is necessary to discover the location of the application code, what it does, which applications and components it interacts with, and so forth.

*Discovery tools* are designed to assist in documenting relationships among applications and subroutines and creating visual representations that show asset relationships effectively and efficiently. These tools are intended to extract components and associated data items from existing code, simplifying the
difficult and time-consuming task of building components manually. Discovery
tools can also perform impact analysis to understand data movement and
application interaction during run time and record the results for use in future
application development and test. Both static (code-based) and dynamic
(runtime-based) analyses are supported. The discovery tools include:

- **Rational Asset Analyzer (RAA®)** and **WebSphere Asset Analyzer** are static
  analysis tools designed to assist in maintaining and extending existing assets
  through impact analysis, connector builder assistance, and graphical
  application understanding.

- **Relativity's Application Portfolio Manager (APM)** centralizes business and
technical intelligence for applications across mainframe, midrange, and
distributed environments. APM allows for rapid construction of operations
  from a top-down as well as a bottom-up approach. The generated models can
  be transferred to third-party business process management technologies.

After a client understands existing business assets and the impact of changes to
those assets, development tools can be used to more effectively develop new
business applications. Most tools support extending existing applications to
business without modifying existing code. This rapid development and
deployment capability enables a quick return on investment while more
substantial development projects are under way that might involve the creation of
a more complete SOA.

### 2.5.2 Enterprise transformation strategy: Improve, adapt, and innovate

SOA focuses on the concept of *reusability*, which means that application
components must be built in a way that facilitates reuse and avoids creating
components from scratch when existing assets can satisfy the requirements. The
combination of existing data and applications on the mainframe and the concept
of reuse through transformation leads to a fast startup into the SOA model, which
is the concept of *enterprise transformation*.

Enterprise transformation and application modernization are critical to an
enterprise’s application development strategy, because the high cost and risk of
rewriting existing applications is often an inhibitor. If a company can reuse
existing well tuned and proven code that has been implementing business
function satisfactorily for years, savings can be significant. Aside from the high
cost and risk of migration of these applications, performance, reliability, and
scalability requirements often make the current environment the best choice.
When considering reuse of existing assets, developers experienced in traditional
languages must be considered an important part of the overall business
development team. As the availability of these development skills continues to
tighten in the marketplace, it becomes more critical to create a development environment where existing assets can be maintained and extended using current development approaches and available skills.

SOA has particular importance in the enterprise transformation practice, because it has the power to unlock existing applications and data and expose them as services, therefore, providing extended value to the business. Considering the investments that companies have made in developing applications and the importance that those applications have to the core business, service enablement using standard-based interfaces helps to extend the lifecycle of those applications and leverages existing investments.

The move toward SOA is not going to happen overnight. It is not an appliance to be installed on top of an existing infrastructure, thus making it SOA ready. The implementation process is a set of granular changes applied to the existing architecture and the exploitation of new technologies that will gradually make the environment SOA ready.

The IBM enterprise transformation strategy supports this gradual, granular process that will help to enable existing assets as services. It defines three solution frameworks, sometimes referred to as *styles of transformation*, to help clients convert IT assets from siloed applications to shared resources, and then to interdependent software components and services.

The strategic goal is to innovate by enabling the creation of new software components that have strategic business value and support an on-demand environment. But, that takes time, effort, and money. So, more immediate and tactical solutions are required as well. For example, the ability to transform siloed applications to shared resources can enable better ways of interacting with customers, partners, and suppliers. Certain changes to applications will require immediate solutions that can drive business value, but they might not be as flexible in adapting to future changes in the business process. The three transformation styles, *Improve*, *Adapt*, and *Innovate*, address both tactical and strategic service enablement solutions.

While IBM identifies these three approaches for modernizing a client's enterprise, an IT organization will likely not use just one style. Businesses often choose multiple styles of modernization for various types of solutions, so the deployment scenario is determined case by case.

The three transformation styles of *Improve*, *Adapt*, and *Innovate* cover both tactical and strategic service-enablement solutions. This section presents a more detailed look at these styles.
**Improve**
The Improve style is characterized by the use of new technologies to Web-enable or service-enable applications at the user interface level without changing those applications and with minor changes or additions to the middleware infrastructure. Improve is often the first step toward SOA, where clients simply enable existing applications with SOAP or MQ protocols to facilitate integration. This style concentrates on transforming the user experience by providing a more sophisticated and productive user interface for applications.

**Adapt**
Adapt goes a bit farther in terms of application transformation and enhancement of the existing infrastructure. Adapt comes closer to full SOA architecture, but it requires more investment in business componentization and in IT services. The result is more flexibility for the business and the value that this flexibility represents.

Adapting existing connectivity enables broader application integration and provides the ability to incorporate core applications into more modern application flows. Adapting allows clients to leverage existing applications to develop better customer, partner, and supplier relationships. But the connection of many existing applications to new applications and new architectures poses a compatibility problem: Data formats, communication protocols, and existing programs frequently cannot communicate without an intermediary.

The requirement emerges for an intermediate broker to be used for data mediation, protocol transformation, message routing, and so forth. This functionality is where the requirement for an ESB usually emerges. The ESB is not a direct component of the Adapt transformation style, but it does serve as the hub to connect the applications, transactions, and services that are transformed using the adapt techniques. Other elements of the SOA reference architecture are also involved in this phase, including information and access services. The service management infrastructure is often planned and designed during this phase.

**Innovate**
The Innovate style of modernization is characterized by the creation of new applications that are fully compliant with the SOA model. Accordingly, with reference to SOA strategic approaches, this style is mostly top-down, where business processes are modeled using a modeling tool. Deployed applications invoke a service or services (applications) that make up the composite business applications or processes. This style might require that a totally new service or application be developed, or it might involve the transformation and reuse of an existing application to meet the business requirements.
Transforming the application structure and architecture requires the highest degree of investment, but it pays off with the greatest business value and process flexibility. In this Innovate style of transformation, core applications are restructured to provide the greatest amount of business benefit. This restructuring allows clients to more rapidly innovate and change their business processes using existing IT applications to create new and differentiated market solutions. In order to innovate, tools are required to design and deploy new applications, and a server is needed to orchestrate and direct the newly created business processes.

Another significant value that SOA brings to companies is an architecture that is programming language-neutral, thereby allowing companies to continue to develop new applications using traditional languages, which leverages existing skills, tools, and investments. Being able to use traditional languages is important to mainframe shops, because senior application developers can continue to use their preferred languages and tools, providing improvements in productivity and economics.
SOA application enablement patterns for z/OS

In this part of the book, we have grouped patterns in three scenarios. Each scenario represents a certain driver for service-oriented architecture (SOA) enablement. We discuss the following scenarios:

- Accessing data on z/OS as a service
- Accessing programs on z/OS as a service
- Integrating external services in existing z/OS applications

We describe each of the patterns within those scenarios using a standard structure. We discuss the following aspects for each pattern:

- **Initial state**
  The initial state covers the current maturity of the application in terms of complexity for SOA framework qualification criteria, for example, the extent to which service-level agreement (SLA) goals are currently being met and the nature of the application, such as simple, complex, hybrid, rigid, monolithic,
and so on. Quality of service (QoS) aspects also play a role. The initial state will tell us how much work needs to be done to become somewhat SOA compliant.

- **Current architecture**
  The current architecture depicts the current architecture diagram of the system that includes both the infrastructure and application level.

- **Solution strategy**
  The solution strategy covers a planned approach to a solution, taking into consideration the SOA features and QoS features of the mainframe, which require a well designed methodology. Also, there might be situations that need a quick development time or turnaround time in deploying a solution or solutions. In this case, there might be an approach to a solution that takes into consideration the short-term benefits only and has the guidelines (instead of a methodology) that need to be put in place for a smooth SOA transition in phases.

- **Solutions**
  For each solution, we discuss:
  - **Target architecture**
    Using the adopted methodology, we arrive at a target architecture diagram that includes both an infrastructure and an application perspective.
  - **Runtime software prerequisites**
    Runtime software prerequisites include the required software products to run the target architecture. We do not mention prerequisites at the level of the operating system, but we focus on the middleware level.
  - **Development tools**
    In this section, we mention the development tools that are required or recommended.
  - **Advantages**
    Under this heading, we mention the particular advantages of the solutions.
  - **Limitations**
    If there any important limitations, we mention those limitations.
  - **Guidelines**
    Under this heading, we mention any noteworthy guidelines for using the solution.

This information gives the reader an idea of what to expect in the forthcoming chapters. The patterns that we describe can be a good starting point for an architect to identify the available options and correlate the options with their own situation. There might be more than one solution for a pattern. The objective of this book is to provide almost all the possible solutions for a pattern and we expect the reader to make a decision that best suits their enterprise.
Accessing data on z/OS as a service

One of the first steps in a migration to service-oriented architecture (SOA) might be to offer an enterprise-wide, technology-neutral way of providing access to mission-critical business data. This concept is *information as a service*, and there are many ways to implement this idea. In this book, we look at a number of options from an application perspective. In this chapter, we discuss the following patterns involving data integration:

- Accessing VSAM data as a service
- Accessing DB2 data as a service
- Accessing IMS data as a service

**Note:** IBM provides rich set of Information Management solutions that can help you access data in an SOA and help you build a sophisticated Information Management architecture. In this book, however, we focus on solutions that exploit the capabilities of z/OS, VSAM, DB2, and IMS.
3.1 Accessing VSAM data as a service

The scenario presented in this chapter applies to anyone with a need to access data stored in VSAM files in an SOA. The service consumer with this need might reside on z/OS, but it can also reside elsewhere. The service consumer can be implemented in J2EE, CICS, .Net, and so on. With all of the scenarios in this book, a structural difference between technologies, such as having a .Net consumer and data stored in a VSAM file on z/OS, does not need to be any problem due to the richness of the SOA-enabling technology that is available these days. Be aware, though, that in the case of VSAM, the possible options are not that common and you might even initially think it is not possible to access VSAM through a (data access) service.

**Important:** There is not really a direct need to migrate data from VSAM to a relational database for the sake of improving integration and accessibility. In this chapter, we present three architectures to access VSAM data SOA-style. They are not equally pure and you can debate about each architecture’s SOA maturity, but they do the job of SOA integration.

One of the reasons, though, to migrate part of the data to a relational database, such as DB2, might be to benefit from the capability to parse and store XML inside DB2 tables. IMS supports storage of XML as well.

3.1.1 Initial state

A fictional company has been maintaining data over the last decades in VSAM key-sequenced data set (KSDS) files, and the company wants to enable real-time access to this data for a variety of business partner applications. Their business partners run their applications predominantly on distributed systems, and the master data from the VSAM files is currently replicated in an asynchronous manner. Multiple components are involved in this process. An issue with this replication process is that databases between the fictional company and their business partners are not always in sync at any given point of time except for the beginning of a business day, because the data sync process between the databases happens only during the overnight batch window. In order to avoid this complexity, the fictional company has made a decision to integrate the VSAM data with business partners’ applications in a real-time manner using open standards.
3.1.2 Current architecture

Figure 3-1 represents the current architecture diagram in a simplified form. The cloud represents a replication mechanism between the VSAM data and the Java and .Net client applications. At the same time, the data in the VSAM files is accessible through CICS transactions and batch programs running on z/OS.

![Diagram of current architecture for VSAM access on z/OS]

3.1.3 Solution strategy

The chosen solution depends upon requirements and factors, such as cost, time, and flexibility. If there is only a need for an easy, simple and inexpensive way of providing access to VSAM data and the business partners prefer to use Web Services (SOAP) over HTTP or MQ only, “Accessing VSAM data through an existing program” on page 301 is appropriate.

Assuming that there is a need for a more flexible, scalable, and platform-independent or software-independent infrastructure and the business partners use a wide range of transmission protocols, such as Java Message Service (JMS), MQ, HTTP, HTTPS, TCP/IP, SOAP, or other transmission protocols.
protocols, “Accessing VSAM data through WebSphere Message Broker” on page 34 or “Accessing VSAM data using JZOS and WebSphere Application Server” on page 37 will be more appropriate.

**Benefits**

The benefits of the solution strategy are:

- The issue of maintaining redundant data can be avoided.
- All the data sync issues can be eliminated, because this solution provides a single data store.
- Data is available in real time for both the fictional company and its business partners at the same time.
- Future (business partner) applications can reuse the same solution architecture.
- The cost, time, and effort to maintain the batch window for keeping the databases in sync have been eliminated.
- “Accessing VSAM data through WebSphere Message Broker” on page 34 has the following additional benefits:
  - Sources and destinations can be of any format, and they are independent of each other.
  - It provides better business and IT flexibility.
  - The integration is much easier to perform.
  - Message logging and exception handling facilities are available.
  - This solution is highly scalable.

**3.1.4 Solutions**

We discuss the following solutions:

- Accessing VSAM data through an existing program
- Accessing VSAM data through WebSphere Message Broker
- Accessing VSAM data using JZOS and WebSphere Application Server

**Accessing VSAM data through an existing program**

The easiest and quickest way to integrate VSAM data through Web Services is to expose the existing z/OS program that currently accesses the VSAM data as a Web service. If no such program currently exists, a new program might need to be built. The Web service can then be invoked by a Java application, a .Net application, or any other application that is capable of using Web Services as a protocol.
Assuming that a CICS program is used to access the VSAM data, then CICS Transaction Server Version 3.1 or 3.2 can be used to expose the existing CICS program as a Web service. The COMMAREA that contains the structure related to the VSAM record needs to be converted to a Web Services Description Language (WSDL) file using tools, such as *Rational Developer for System z (RDz)* or the *CICS Web Services Assistant (WSA)*. The WSDL file can then be published to the Java client or .Net client application to invoke the Web service.

CICS TS Version 3.1 and 3.2 provide new resource definitions, such as PIPELINE, URIMAP, and TCPIPSERVICE, that are used to define a CICS program as a Web service. Refer to Section 2.3.1 of *Application Development for CICS Web Services*, SG24-7126, to understand the concept of CICS as a service provider.

**Note:** The existing program can also be an IMS program. In that case, the IMS SOAP Gateway can be used instead of CICS Web Services. Refer to “IMS Web Services” on page 82 for more details.

**Target architecture**

Refer to Figure 3-2 on page 32, which depicts the process of exposing VSAM data to Web service consumers via CICS Web Services.
**Runtime software prerequisites**

The following software prerequisites exist for accessing VSAM through a Web service-enabled program on z/OS:

- CICS Transaction Server Version 3.1 or higher in the case of a CICS program
- IMS Version 10 or higher, in combination with IMS SOAP Gateway Version 10.1 or higher, and IMS Connect Version 10 or higher, in the case of an IMS program

**Development tools**

It is difficult, if not impossible, to develop this scenario without proper tooling. We recommend that you use Rational Developer for System z (RDz) for both the CICS and IMS variation. In the case of CICS, the CICS Web Services Assistant (WSA) can be used as an alternative to RDz.
**Advantages**
Several of the advantages of this solution are:

- Reuse of existing assets (CICS or IMS program) and, in many cases, no need to develop new logic
- Robust data access through CICS or IMS (transactional, recoverable, fast, and secure)
- No data migration needed, because the data can remain inside VSAM files

**Limitations**
Limitations of this solution are:

- No loose coupling between the program and data layer
- VSAM not providing support for XML natively, although adapters and connectors exist to access VSAM XML-style

**Guidelines**
The guidelines are:

- Prioritize the process of identifying either an existing CICS or an existing IMS program that accesses the VSAM data, or prioritize the creation of a new program.
- Be careful to avoid creating Web services with the wrong granularity. For example, a Web service per VSAM file might result in too many fine-grained services. A better design is to implement data integration inside the new service.
- Pay attention to the security aspects when external applications gain direct access to a CICS or IMS Web service. For CICS, for example, you can take these measures:
  - Having the user ID as part of the application data header and having the program defined in the CICS PIPELINE configuration file to use the user ID to check with RACF® or other security products, such as ACF/2
  - Using the security authentication validation features of the TCPIPSERVICE definition
  - Defining the TCPIPSERVICE to use Secure Sockets Layer (SSL) transport

Refer to *Securing Access to CICS Within an SOA*, SG24-5756, for an extensive discussion of CICS Web Services security.
Accessing VSAM data through WebSphere Message Broker

This solution is based on using an enterprise service bus (ESB) concept to access VSAM data. We have chosen to discuss the WebSphere Message Broker for z/OS product, because WebSphere Message Broker for z/OS provides specific VSAM nodes. *WebSphere Message Broker* is an advanced version of an ESB. An ESB offers many advantages in a service-oriented architecture framework. An ESB offers the advantage of stateless data transformations between the source and target message formats and, as a result, the avoidance of changes to source and target application programs.

With regard to VSAM data integration, WebSphere Message Broker provides a number of nodes to access VSAM data sets as part of a *message flow*:

- **VSAMInput** to read records from a VSAM data set
- **VSAMRead** to read a record from a VSAM dataset
- **VSAMWrite** to write a record to a VSAM dataset
- **VSAMUpdate** to update a record to a VSAM dataset
- **VSAMDelete** to delete a record from the dataset

These nodes can be used to construct a *message flow*. Figure 3-3 on page 35 shows an example of a message flow.
Figure 3-3  Example of WebSphere Message Broker message flow with VSAM access

**Target architecture**

Figure 3-4 on page 36 depicts the target architecture when using WebSphere Message Broker to access VSAM data.
Runtime software prerequisites
The following runtime software prerequisites exist for this solution on z/OS:

- WebSphere Message Broker Version 5 or higher
- VSAM node SupportPac (IA13) for WebSphere Message Broker

Development tools
Message flows in WebSphere Message Broker are developed using WebSphere Message Broker Toolkit. We recommend that the level of the Message Broker Toolkit corresponds with the level of the WebSphere Message Broker run time. However, in certain cases combinations of different levels are supported.

Advantages
The advantages of this solution are:

- Usage of an ESB providing SOA benefits, such as loose coupling
- Data access that is fully transparent for client applications
Limitation
The limitation of this solution is that ESB might introduce overhead because of the transformation.

Guidelines
In this scenario, SOA characteristics, such as business-IT complexity, loose coupling, and flexibility, can really only be achieved with this solution.

Example
We describe a sample of this scenario in Chapter 9, “Exposing data as a service by using IBM WebSphere Message Broker” on page 217.

Accessing VSAM data using JZOS and WebSphere Application Server
This solution uses the JZOS toolkit, which is a set of utility classes for access to z/OS services from Java applications, including the ability to use “traditional” MVS data sets, such as VSAM files. So, the approach here is to create a Java application, which then is exposed as a Web service in WebSphere Application Server.

Target architecture
The target architecture is depicted in Figure 3-4 on page 36. The Web service in WebSphere Application Server is accessible by any Web Services client program. The WebSphere Application Server Web service wraps the JZOS access to the VSAM data. This scenario is a direct and standard-based solution that uses Java and Web Services.
The following runtime software prerequisites exist for using JZOS on z/OS:

- **JZOS toolkit.**
  We have tested this solution with JZOS included in the Java Software Developer Kit (SDK) on z/OS.

  **Note:** Starting with SDK™ Version 1.4 for z/OS, the JZOS toolkit is part of the z/OS SDK. It is available to all Java applications running on a z/OS JVM™, because it is a standard extension.

- **WebSphere Application Server for z/OS.**
  We have tested this solution with WebSphere Application Server Version 6.1.

**Development tools**
Web services running in WebSphere Application Server can be developed using IBM Rational Application Developer (RAD) or any other tool that supports the appropriate levels of J2EE and Web Services.
**Advantages**
The advantages of this solution are:

- Web service hosted in J2EE server, which provides a highly standardized environment for hosting and accessing a Web service
- Data access to VSAM local on z/OS, providing fast and secure access

**Limitation**
JZOS and WebSphere Application Server are not a mainstream technology combination, such as, for example, Java Database Connectivity (JDBC™).

**Guidelines**
Just as with any data-centric solution for service enablement, be careful that you ensure a reasonable service granularity.

**Example**
We provide a sample for this scenario in Chapter 6, “Exposing VSAM data as Web Services” on page 129.

### 3.2 Accessing DB2 data as a service

This scenario is similar to the scenario that we discussed in 3.1, “Accessing VSAM data as a service” on page 28. The data is on z/OS, but this time, the data is in an hierarchical database management system (DBMS). DB2 by nature has been more open and more easily accessed from distributed applications than VSAM data sets. Java DataBase Connectivity Type 4 drivers and the DB2 Distributed Remote Data Access (DRDA®) protocol are commonly used protocols for remote access to DB2 on z/OS. However, with SOA techniques, we can still advance further.

#### 3.2.1 Initial state

The initial state of this scenario is a DB2 database on z/OS and various traditional programs (CICS and IMS) on z/OS accessing this data. The DB2 database is also accessed by batch programs. Certain distributed applications access the database using JDBC.
3.2.2 Solution strategy

The data needs to be exposed as a service, which accelerates the reuse of the data and helps to access the data using common queries. In addition to data access, stored procedures also need to be callable using a Web Services call.

Benefits

The benefits of the new solution strategy are:

- All data remains on z/OS.
- You can run common queries, which means that heterogeneous client applications do not need to know any specifics about how to run SQL for a DB2 database on z/OS.
- Client applications are not confronted with DB2 on z/OS specific data types.

3.2.3 Current architecture

Figure 3-6 depicts the architecture of the current state.
3.2.4 Solutions

We discuss the following solutions:

- Accessing DB2 on z/OS using Data Web Services (DWS)
- Accessing DB2 on z/OS using custom data access
- Accessing DB2 on z/OS using WebSphere Message Broker

**Accessing DB2 on z/OS using Data Web Services (DWS)**

Data Web Services (DWS) can be used to expose existing SQL statements as Web services. These statements can be (and typically *will* be) parameterized, that is, they contain host variables, which become parameters of the Web service to be created.

For more details and an example, refer to 7.2, “Using Data Web Services (DWS)” on page 160.

**Target architecture**

Figure 3-7 on page 42 depicts the target architecture for this solution. The Web service in this solution is deployed to a J2EE server, such as WebSphere Application Server. In our example, WebSphere Application Server runs on z/OS. The deployed Web service accesses DB2 using Java Database Connectivity (JDBC), either locally on the same logical partition (LPAR) using a JDBC Type 2 driver or on another LPAR using a JDBC Type 4 driver. This solution works for both SQL queries and calling stored procedures.
Runtime software prerequisite
The runtime software prerequisite for using DWS on z/OS is WebSphere Application Server. We have tested this scenario with WebSphere Application Server Version 6.1.

Development tool
The development tool that was used for this solution is IBM Data Studio Developer. The latest version is Version 2.1, but we used Version 1.2.

Advantages
The advantages of this solution are:
- Extremely easy to use
- Support for static SQL using pureQuery
- Support for SOAP over both HTTP(S) and JMS
- Existing queries can easily be reused
**Limitations**
The limitations of this solution are:

- Separate tool required
- Additional tier required in the form of a J2EE server to host the Web service
- Only basic create, read, update, and delete functionality is exposed
- Purely data-centric, not necessarily business-centric
- No enforcement of business rules (except by the database itself, in the form of constraints)

**Guidelines**
The guidelines for this solution are:

- Read-only access (queries) work as long as they expose a business view of the data. Be careful about modifying services, and especially, be sure to consider service granularity and business rules enforcement.
- Database constraints are an excellent vehicle for enforcement of business rules.

**Example**
Refer to 7.2, “Using Data Web Services (DWS)” on page 160.

**Accessing DB2 on z/OS using custom data access**
The essence of this solution is the usage of hand-written (or tool-generated) SQL statements to access DB2 from a J2EE application. This J2EE application is then exposed as a Web service, and a J2EE application server is then used to host this Web service.

Various technology and tooling options are available for this scenario:

- Hand-coded JDBC or SQLJ access
- Use of pureQuery
- Container-Managed Persistence Entity JavaBeans™ (CMP EJBs)
- An Object-Relational Mapping (ORM) tool, such as Hibernate™

**Target architecture**
Figure 3-8 on page 44 shows the target architecture of this solution. In the diagram, a Web service is implemented in WebSphere Application Server. This Web service can use any of the techniques previously mentioned to access DB2. The protocol to access DB2 is Java DataBase Connectivity (JDBC) or SQLJ. This scenario can be used for SQL queries and stored procedures.
Enabling z/OS Applications for SOA

Figure 3-8 Target architecture for custom data access hosted in WebSphere Application Server

Runtime software prerequisites
WebSphere Application Server or an equivalent J2EE application server is the runtime software prerequisite that exists for custom data access on z/OS.

Development tools
IBM Rational Application Developer (RAD), Eclipse, or any development tool that supports developing J2EE applications and Web services can be used for this solution.

Advantages
The advantages of this solution are:

- In contrast to “Accessing DB2 on z/OS using Data Web Services (DWS)” on page 41, you have the option to add business functionality and to enforce business rules.
- ORM tools allow a high degree of flexibility when defining and evolving the object model.
**Limitations**
The limitations of this solution are:

- Hand-coding JDBC or SQLJ data access is error prone. Consider tool support and a utility framework, such as Spring.
- ORM tools add another level of complexity, and they have a considerable initial learning curve.

**Accessing DB2 on z/OS using WebSphere Message Broker**
By using WebSphere Message Broker, DB2 on z/OS can be accessed using an ESB approach. WebSphere Message Broker provides a number of nodes specifically for database access, but database access can also be manually programmed inside a so-called *Compute node*. WebSphere Message Broker provides a generic Compute node and a Java Compute node. The generic Compute node uses the Open Database Connectivity (ODBC) protocol to access DB2, and the Java Compute node uses JDBC to access DB2.

**Target architecture**
Figure 3-9 on page 46 depicts the target architecture for using WebSphere Message Broker to access DB2 on z/OS. A typical message flow in WebSphere Message Broker consists of an HTTP or JMS input node to receive the request from the client application. This input can also be in the form of a SOAP request. Eventual transformation nodes are executed to prepare the input for the database calls. One or several database nodes are used to perform create, read, update, and delete actions on the data. In the case of extremely complex database access, Compute nodes or Java Compute nodes are added. After the database access has been completed, transformation is typically required again before building a reply back to the client application.
**Runtime software prerequisite**

WebSphere Message Broker is the prerequisite for accessing DB2 on z/OS using WebSphere Message Broker.

**Development tool**

WebSphere Message Broker Toolkit is the development tool that is used for this solution.

**Advantages**

The advantages of using this solution are:

- Using an ESB provides SOA benefits, such as loose coupling.
- Data access is fully transparent for client applications.

**Limitation**

ESB might introduce overhead because of transformation.

### 3.3 Accessing IMS data as a service

This scenario is similar to the scenario that is discussed in 3.1, “Accessing VSAM data as a service” on page 28 and 3.2, “Accessing DB2 data as a service” on
The data is on z/OS, but this time, the data is in an IMS database. IMS databases are highly efficient, but have a specific organization. In the past, you were only able to access IMS databases on MVS and OS/390® using native protocols. Over the past few years, this restriction has changed, and now, you can access IMS databases directly from outside. Also, IMS databases can be accessed in a relational style using SQL.

3.3.1 Initial state

In this example, the initial state is similar to the scenario that is described in 3.1, “Accessing VSAM data as a service” on page 28, but the master data is now kept in IMS databases.

3.3.2 Current architecture

Figure 3-10 on page 48 represents the current architecture diagram in an extremely simplified form. The cloud represents a replication mechanism between the IMS data and the Java and .Net client applications. At the same time, the data in the IMS databases is accessible through IMS transactions and batch programs running on z/OS.
3.3.3 Solution strategy

The chosen solution depends on requirements and factors, such as cost, time, and flexibility. If you only need an easy, simple, and inexpensive way to provide access to the IMS data and the client applications and prefer to use Web Services (SOAP) over HTTP, “Accessing IMS databases through an existing program” on page 491 is appropriate.

If you need a more flexible, scalable, platform-independent, and software-independent infrastructure and the client applications use a wide range of transmission protocols, such as JMS, MQ, HTTP, HTTPS, TCP/IP, SOAP, or other transmission protocols, “Accessing IMS databases using JDBC” on page 51 is more appropriate.

Benefits

The benefits of the solution strategy are:

- The issue of maintaining redundant data can be avoided.
All the data sync issues can be eliminated, because this solution provides a single data store.

Data is available in real time for both the fictional company and its business partners at the same time.

Future (business partner) applications can reuse the same solution architecture.

The cost, time, and effort to maintain the batch window for keeping the databases in sync has been eliminated.

### 3.3.4 Solutions

We discuss the following solutions:

- Accessing IMS databases through an existing program
- Accessing IMS databases using JDBC

**Accessing IMS databases through an existing program**

This solution focuses on using an existing IMS (or CICS) program to access the IMS database. This existing program will then be exposed as a Web service. If the program is an IMS program, follow the approach that is explained in “IMS Web Services” on page 82. If the program is a CICS program, follow the approach that is explained in “CICS Web Services” on page 59.

**Target architecture**

Figure 3-11 on page 50 shows a diagram of the architecture of this solution. In this case, we assume that the existing program is an IMS program. The vehicle that is used to access the IMS program as a Web service is the IMS SOAP Gateway. At the same time, you can continue to use the existing program from a 3270 terminal.
Figure 3-11  Target architecture for accessing IMS databases through an existing program

**Runtime software prerequisites**

The following runtime software prerequisites exist for this solution:

- IMS Version 10 or higher
- IMS Connect Version 10 or higher
- IMS SOAP Gateway Version 10.1 or higher

**Development tool**

We used Rational Developer for z (RDz) Version 7.1.1 (or higher) for this solution.

**Advantages**

The advantages of this solution are:

- This solution is a simple and easy way to service-enable IMS transactions accessing IMS databases.
- You can retain your existing transactions, which have been proven, secure, reliable, and scalable.
**Limitations**
The limitations of this solution are:

- Only non-conversational transactions are supported.
- Only commit mode 1, sync level NONE processing is supported.
- Two-phase commit is not supported.

**Accessing IMS databases using JDBC**
This solution is quite sophisticated. Again, we assume that we need a Web service that is reusable by the enterprise. In this solution we implement the Web service in a J2EE server, WebSphere Application Server. Then, we use a resource adapter to connect to the IMS database.

**Target architecture**
In this architecture, we do not use IMS Transaction Manager and IMS Connect. We access the IMS database directly from WebSphere Application Server on z/OS. You can write Java applications that run on WebSphere Application Server for z/OS and access IMS databases when WebSphere Application Server for z/OS and IMS are on the same logical partition (LPAR). JDBC or IMS hierarchical database interface for Java calls are passed to the IMS DB resource adapter, which converts the calls to DL/I calls. The IMS DB resource adapter passes these calls to Open Database Access (ODBA), which uses the database resource adapter (DRA) to access the DL/I region in IMS.

In order for a Java application to access an IMS database, it needs information about that database. The DLIModel utility allows you to transform your IMS database information (program specification blocks, database descriptions, and COBOL copybooks) into application-independent metadata.

Figure 3-12 on page 52 shows the target architecture for this solution.
Fig. 3-12 Target architecture for accessing an IMS database using ODBA

**Runtime software prerequisite**
IMS Version 10 or higher is the prerequisite to access an IMS database using ODBA.

**Development tools**
We used the following development tools for this solution:

- Rational Developer for System z (RDz)
- DLIModel utility

**Advantages**
The advantages of this solution are:

- WebSphere Application Server is used as the container for the data access programs. WebSphere Application Server provides a common environment that is easily accessible from outside applications.

- The development process is relatively easy, considering that we are accessing an IMS database.
Even though the database is hierarchical by nature, we can use JDBC statements to access the database.

When WebSphere Application Server and the IMS DB are on the same LPAR, a high quality of service (QoS) is achieved.

The IMS implementation of JDBC supports a selected subset of the full facilities of the JDBC 2.0 application programming interface (API). This subset allows you to perform all the tasks that traditional IMS applications using DL/I calls can perform. You can also use the IMS Java hierarchical database interface for lower level access to IMS databases.

**Limitations**
The limitations of this solution are:

- You cannot use the DLIModel utility to modify a data structure in IMS on the mainframe.

- This solution does not support:
  - Main storage database (MSDB), hierarchical sequential access method (HSAM), and simple hierarchical sequential access method (SHSAM) databases
  - Shared secondary indexes
  - The processing option to make an application sensitive to only the segment key of a segment (PROCOPT=K option) in a PSB SENSEG statement

**Guidelines**
Refer to the following sources for guidelines:

- Chapter 8, “Configuring WebSphere Application Server for z/OS,” in *IMS Communications and Connection Guide Version 10*, SC18-9703


- Chapter 12, “Java class libraries for IMS,” in *IMS Application Programming API Reference Version 10*, SC18-9699

- The DLIModel utility Web site at:
  

**Example**
Refer to the example with a step-by-step technical explanation in Chapter 8, “Exposing IMS data as a service” on page 179.
Accessing programs on z/OS as a service

This chapter explains most of the technology solutions that are available today to integrate existing z/OS programs into a service-oriented architecture (SOA) infrastructure.

We discuss the following patterns:

- Accessing a CICS program on z/OS as a service
- Accessing a complex CICS application as a service
- Accessing IMS programs as a service
- Accessing a CICS application as a composite service
- Accessing DB2 stored procedures on z/OS as a service
- Accessing presentation logic as a service
4.1 Accessing a CICS program on z/OS as a service

An example of this pattern is a company whose application portfolio comprises well designed CICS applications. The programs generally have their presentation and business logic layers clearly demarcated. The applications contain several programs with business logic in high demand for reuse throughout the company. The company is looking for solutions to provide an alternate user interface to the 3270 displays, as well as a solution for providing broad access to certain CICS business logic programs. This access ideally includes business partners. These vendors possess a wide variety of distributed systems and several vendors also use System z.

4.1.1 Initial state

The company has a large number of CICS-based applications that use traditional character-based 3270 interfaces using CICS Basic Mapping Support (BMS) maps as an interface to send and receive the data that is entered. The process of sending and receiving the COMMAREA data is contained separately within programs, and the data is passed to other CICS programs containing the core business logic.

The application portfolio also contains programs that are a mix of stand-alone business logic and terminal-based CICS programs that take input from the users.

4.1.2 Current architecture

Figure 4-1 on page 57 shows the current application architecture at a high level.
Figure 4-1  High-level view of the current application architecture

Figure 4-2 shows a more detailed view of the current architecture.

Figure 4-2  More detailed view of the current application architecture
4.1.3 Solution strategy

The solution strategy is to make the business logic programs accessible for a broad variety of potential client programs in a standardized way. The quality of service (QoS) plays an important role, too. At this point, you can make one of three architectural decisions:

- Web service-enable selected CICS business logic programs and provide the Web Services Description Language (WSDL) to those individuals who want to access those service (directly).

- Do not Web service-enable the CICS business logic program, but implement a Web service interface in WebSphere Application Server and access the CICS program from WebSphere Application Server through any of the supported protocols and connectors, for example, J2EE Connector architecture (J2C) or WebSphere MQ.

- Do not Web service-enable the CICS business logic programs, but provide an enterprise service bus (ESB) layer with access to those CICS programs using native/traditional access methods, such as external CICS interface (EXCI) and WebSphere MQ. The access point for those CICS programs for clients will then be in the ESB.

All solution directions can be considered standardized and SOA-compliant. However, the ESB direction is a better architecture, providing advantages in the area of governance, flexibility, quality of service (QoS), and more. After all, just providing a Web Services interface without using an ESB remains a “point to point” solution, but just with another interface protocol.

In 4.1.4, “Solutions” on page 59, we discuss solutions of all categories.

Benefits

The benefits of providing broader and more standardized access to CICS business logic programs are:

- Leverage robust and proven business logic with high performing native data access

- Avoid building redundant programs throughout the company that perform the same functions as the CICS programs, but usually with lower quality of service
4.1.4 Solutions

We discuss the following solutions:

- CICS Web Services
- Web service interface in WebSphere Application Server and J2C to CICS
- Web service interface in WebSphere Enterprise Service Bus and J2C to CICS
- Web service interface in WebSphere Enterprise Service Bus and WebSphere MQ to CICS
- Web service interface in WebSphere Enterprise Service Bus and SOAP/JMS to CICS
- Web service interface in WebSphere Message Broker and EXCI to CICS
- Web service interface in WebSphere Message Broker and WebSphere MQ to CICS
- Web service interface in WebSphere Message Broker and SOAP/JMS to CICS

**CICS Web Services**

One of the first solutions to consider is to use CICS Web Services support, which is part of CICS TS Version 3.1 and Version 3.2, to expose the CICS business logic programs as a Web service. A lot of detailed information with regard to CICS Web Services is already available. For example, refer Section 2.3.1 in *Application Development for CICS Web Services*, SG24-7126.

**Target architecture**

Figure 4-3 on page 60 shows the target architecture when using CICS Web Services. After a CICS Web service is implemented using CICS Web Services support, it is accessible from a Web service consumer using Web Services protocols. The consumer can exchange SOAP messages with CICS over HTTP or Java Message Service (JMS).
Runtime software prerequisite
CICS Transaction Server Version 3.1 or Version 3.2 is the runtime software prerequisite exist for this solution.

Development tools
Use either Rational Developer for System z (RDz) or CICS Web Services Assistant to Web service-enable CICS programs.

Advantages
This solution has the following advantages:
- Simple and easy way to service-enable CICS programs
- Existing programs, which have been proven, secure, reliable, and scalable, retained
- Excellent tooling support
- Standard protocols
**Limitation**

If the CICS Web Services Assistant DFHLS2WS utility is used to generate the Web service artifacts, such as the wsdl file, only basic support is available. Copybook fields, such as COMP and arrays, are not supported.

**Guidelines**

Guidelines for this solution include:

- You can use Rational Developer for z (RDz) to generate the Web service artifacts, covering all possible data definitions.
- You need to carefully consider the security aspects when external applications gain direct access to a CICS Web service. Examples of security measures that can be taken are:
  - Having user ID as part of the application data header and having the program defined in the CICS PIPELINE configuration file to use the user ID to check with RACF or other security products, such as ACF/2
  - Using the security authentication validation features of the TCPIPSERVICE definition
  - Defining the TCPIPSERVICE to use Secure Sockets Layer (SSL) transport

Refer to *Securing Access to CICS Within an SOA*, SG24-5756, for an extensive discussion of CICS Web Services security.

**Web service interface in WebSphere Application Server and J2C to CICS**

In this approach, a Web service is developed and implemented in WebSphere Application Server. This Web service then uses a J2C connector, in this case, the resource adapter for the CICS Transaction Gateway. You can use any J2EE server that supports this resource adapter. Alternatively, you can use WebSphere MQ as well between WebSphere Application Server and CICS.

The Web service can be developed with a tool, such as Rational Application Developer (RAD). RAD contains wizards to easily create a J2C-based Web service that connects to a CICS program. In this approach, CICS Web Services support is not used, and no additional definitions need to be created in CICS.

You can use the RAD’s J2C wizard to create a J2C bean. Mapping COMMAREA data with Java data types is part of this process. When the J2C bean is ready, a Web service can be generated that will use that J2C bean. The Web Service is then deployed to the J2EE application server, such as the WebSphere Application Server on z/OS.
Refer to Chapter 6 in *WebSphere for z/OS V6 Connectivity Handbook*, SG24-7064, for technical hands-on instructions.

**Target architecture**

Refer to Figure 4-4 for a diagram of the target architecture. After the Web service is implemented in WebSphere Application Server, it can be accessed directly from a Web service consumer using the Web Service protocols. In principle, this approach means exchanging SOAP messages over HTTP or JMS. A browser can also access the Web service in WebSphere Application Server using REST.

![Target architecture using WebSphere Application Server and J2C](image)

**Runtime software prerequisites**

This solution requires the following software:

- CICS Transaction Server
- CICS Transaction Gateway (CICS TG) on z/OS
- WebSphere Application Server or a J2EE server supporting the CICS TG resource adapter
Both CICS TG and WebSphere Application Server can run on z/OS or the distributed platform. In our pattern, we assume they run on z/OS. If WebSphere Application Server and CICS TG run on the same z/OS logical partition (LPAR), the communication is cross-memory.

**Development tool**
This solution requires Rational Application Developer (RAD) for developing WebSphere Application Server Web services and J2C beans.

**Advantages**
The advantages of this solution are:

- Applications can be created quickly using the wizards that are provided for Web services and J2C beans in RAD.
- CICS TG is a mature connector supporting two-phase commit, advanced security, and systems management.
- No changes to the existing CICS programs are required, and no additional definitions are required in CICS, apart from the setup for using the EXCI protocol.
- In terms of performance, this solution rates extremely high, especially when WebSphere Application Server, CICS TG, and CICS all run on z/OS.
- WebSphere Application Server provides extensive support for Web Services, including Universal Description, Discovery, and Integration (UDDI), IBM WebSphere Service Registry and Repository (WSRR), and many of the Web Services standards (security, reliability, and so forth).

**Limitation**
The connection between WebSphere Application Server and CICS is not entirely loosely coupled; this solution is more oriented toward a peer-to-peer connection. Transformation, although generated using the tools at development time, is integrated into the new Web service application in WebSphere Application Server.

**Guidelines**
For guidelines about how to develop this solution, refer to Chapter 6 of *WebSphere for z/OS V6 Connectivity Handbook*, SG24-7064, for technical hands-on samples.
Web service interface in WebSphere Enterprise Service Bus and J2C to CICS

In this solution, you use the combination of WebSphere Enterprise Service Bus and a J2C adapter to integrate existing CICS programs into the SOA. ESB functions of WebSphere Enterprise Bus, such as data format and transport protocol transformation, service virtualization, and quality of service (QoS) features are leveraged to make this solution more comprehensive. In this solution, WebSphere Integration Developer is used to create a mediation with a J2C connection to CICS. WebSphere Integration Developer is used to create mediation modules that are deployed to the run time of WebSphere Enterprise Service Bus. The J2C connector used between WebSphere Enterprise Service Bus and CICS is the CICS Transaction Gateway.

Target architecture
Refer to Figure 4-5.

Figure 4-5  Target architecture: WebSphere Enterprise Service Bus, J2C, and CICS TG
Runtime software prerequisites
This solution requires the following runtime software:

- CICS Transaction Server
- CICS Transaction Gateway (CICS TG) on z/OS
- WebSphere Enterprise Service Bus on z/OS

Both CICS TG and WebSphere Enterprise Service Bus can run on z/OS or the distributed platform. In our pattern, we assume that they run on z/OS. If WebSphere Enterprise Service Bus and CICS TG run on the same z/OS LPAR, the communication is cross-memory.

Development tool
This solution requires WebSphere Integration Developer for developing mediation modules and mediation flows.

Advantages
The advantages of this solution are:

- Rich data format transformation functionality is available in WebSphere Enterprise Service Bus, allowing the service requestor to use a standard data format, such as XML or HTML, which is independent of the COMMAREA that is delivered to CICS.
- Transport protocol transformation is available in WebSphere Enterprise Service Bus. The service requestor can use a variety of standard protocols, such as JMS, HTTP, SOAP over HTTP, and SOAP over JMS.
- CICS TG is a mature connector supporting two-phase commit, advanced security, and systems management.
- There are no changes required to the existing CICS programs, and there are no additional definitions required in CICS other than the setup for using the EXCI protocol.
- In terms of performance, this solution rates extremely high, especially when WebSphere Application Server, CICS TG, and CICS all run on z/OS.

Limitations
No real limitations exist for this solution.
**Guidelines**

- There are many books already available with more details about how to use WebSphere Enterprise Service Bus. Refer to:
  - *Implementing an ESB using IBM WebSphere Message Broker V6 and WebSphere ESB V6 on z/OS*, SG24-7335
  - *Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB*, SG24-7369

**Web service interface in WebSphere Enterprise Service Bus and WebSphere MQ to CICS**

In this solution, WebSphere Enterprise Service Bus is used for the ESB, and WebSphere MQ is used as a transport layer between WebSphere Enterprise Service Bus and CICS. No Web Services protocol is used between WebSphere Enterprise Service Bus and CICS, but a WebSphere MQ message gets created using JMS in WebSphere Enterprise Service Bus. A CICS program that is already used to work with WebSphere MQ messages does not need to be changed.

**Target architecture**

Refer to Figure 4-6 on page 67 for the target architecture. WebSphere Enterprise Service Bus receives the request from the Web service consumer, performs the necessary mediation, creates a JMS message, and forwards it to a WebSphere MQ queue. CICS receives the MQ messages through the CICS-MQ Bridge and triggers the CICS program.
Runtime software prerequisites

The following runtime software prerequisites apply:

- CICS Transaction Server
- WebSphere Enterprise Service Bus
- WebSphere MQ

WebSphere Enterprise Service Bus can run on z/OS, Linux on System z, or in a distributed environment. A local WebSphere MQ must exist, however, on z/OS to deliver the message to CICS.

Development tool

We used WebSphere Integration Developer to develop mediation modules and mediation flows for this solution.
**Advantages**

The advantages of this solution are:

- Rich data format transformation functionality is available in WebSphere Enterprise Service Bus, allowing the service requestor to use a standard data format, such as XML or HTML, which is independent of the MQ message that is delivered to CICS.

- Transport protocol transformation is available in WebSphere Enterprise Service Bus. The service requestor can use a variety of standard protocols, such as JMS, HTTP, SOAP over HTTP, and SOAP over JMS.

- The usage of WebSphere MQ between WebSphere Enterprise Service Bus and CICS provides an industry standard and a proven method of transport.

- If the CICS program is already MQ-enabled, no changes are required on the CICS side.

**Limitations**

No real limitations exist for this solution.

**Guidelines**

We recommend:

- For more technical hands-on samples, refer to *Implementing an ESB using IBM WebSphere Message Broker V6 and WebSphere ESB V6 on z/OS*, SG24-7335

- You can also refer to the following developerWorks article, “Building an Enterprise Service Bus using WebSphere ESB, Part 4” by Rachel Reinitz and Andre Tost, 28 February 2006:
  

**Web service interface in WebSphere Enterprise Service Bus and SOAP/JMS to CICS**

The solution is similar to the solution that we discussed in “Web service interface in WebSphere Enterprise Service Bus and WebSphere MQ to CICS” on page 66. However, instead of using a JMS message that gets delivered to CICS as an MQ message using the CICS-MQ Bridge, we now use a SOAP message packaged inside a JMS message. Because we now use a SOAP message, the SOAP message can be delivered to CICS using the CICS Web Services support. The CICS program that is accessed needs to be Web service-enabled.

**Target architecture**

Refer to Figure 4-7 on page 69 for an architectural diagram of this solution.
Runtime software prerequisites
The following runtime software is needed for this solution:

- CICS Transaction Server Version 3.1 or Version 3.2 with CICS Web Services configured
- WebSphere Enterprise Service Bus
- WebSphere MQ

WebSphere Enterprise Service Bus can run on z/OS, Linux on System z, or in a distributed environment. A local WebSphere MQ must exist, however, on z/OS.

Development tools
We used the following tools to develop this solution:

- WebSphere Integration Developer for developing mediation modules and mediation flows
- Rational Developer for System z (RDz) or CICS Web Services Assistant to Web service-enable CICS programs
**Advantages**
The advantages of this solution are:

- Rich data format transformation functionality is available in WebSphere Enterprise Service Bus, allowing the service requestor to use a standard data format, such as XML or HTML, which is independent of the SOAP/JMS message delivered to CICS.

- Transport protocol transformation is available in WebSphere Enterprise Service Bus. The service requestor can use a variety of standard protocols, such as JMS, HTTP, SOAP over HTTP, and SOAP over JMS.

- The usage of WebSphere MQ between WebSphere Enterprise Service Bus and CICS provides an industry standard and proven method of transport.

- After the CICS program is Web Service-enabled, it can also be accessed from sources other than WebSphere Enterprise Service Bus (refer to “CICS Web Services” on page 59).

**Limitations**
No real limitations exist for this solution.

**Guidelines**
We recommend:

- For more technical hands-on samples, refer to *Implementing an ESB using IBM WebSphere Message Broker V6 and WebSphere ESB V6 on z/OS*, SG24-7335

- You can also refer to the following developerWorks article, “Building an Enterprise Service Bus using WebSphere ESB, Part 4” by Rachel Reinitz and Andre Tost, 28 February 2006:

  

**Web service interface in WebSphere Message Broker and EXCI to CICS**

In this solution, access to CICS programs takes places through an Enterprise Service Bus (ESB), such as in “Web service interface in WebSphere Enterprise Service Bus and J2C to CICS” on page 64. However, in this solution WebSphere Message Broker is used for the ESB. WebSphere Message Broker provides a special node to access CICS synchronously using the CICS EXCI protocol. WebSphere Message Broker provides broad functionality for routing and data format and transport protocol conversion. The architecture, along with its advantages and limitations, is generally the same as the WebSphere Enterprise Service Bus architecture.
The Web service consumer sends the request to WebSphere Message Broker, and WebSphere Message Broker forwards the request to CICS using the EXCI node. Eventually, transformation takes place, and logic can be executed before actually accessing CICS.

**Target architecture**

Figure 4-8 shows the architecture of this solution. Web service clients access WebSphere Message Broker using Web service requests. Other clients can access WebSphere Message Broker too using non-Web Services protocols, such as FTP, JMS, and HTTP.

![Figure 4-8 Target architecture using WebSphere Message Broker and EXCI](image)

**Runtime software prerequisites**

The following runtime software prerequisites apply to this solution:

- CICS Transaction Server with EXCI configured
- WebSphere Message Broker on z/OS

Note that the EXCI node is not supported in WebSphere Message Broker on platforms other than z/OS. To use this solution, the Broker needs to run on z/OS, on the same LPAR, or in another LPAR in the same sysplex with CICS.
**Development tool**
We used WebSphere Message Broker Toolkit for developing message flows to run in WebSphere Message Broker.

**Advantages**
The advantages of this solution are:
- Data format transformation functionality is available, allowing the service requestor to use a standard data format, such as XML or HTML, which is independent of the MQ message that is sent to CICS.
- Transport protocol transformation is available. The service requestor can use a variety of standard protocols, such as JMS, HTTP, SOAP over HTTP, and SOAP over JMS.
- EXCI is a CICS native protocol supporting two-phase commit and cross-memory.
- No changes to the existing CICS programs are required, and no additional definitions are required in CICS, apart from the setup for using the EXCI protocol.
- In terms of performance, this solution rates extremely high.

**Limitations**
There are no real limitations with this solution.

**Guidelines**
For additional information about using WebSphere Message Broker, refer to:
- *WebSphere Message Broker Basics*, SG24-7137
- *Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB*, SG24-7369

**Web service interface in WebSphere Message Broker and WebSphere MQ to CICS**
This solution is similar to the solution that we discussed in “Web service interface in WebSphere Message Broker and EXCI to CICS” on page 70. However, this scenario uses another communication technique between the Broker and CICS. Instead of the EXCI node, WebSphere MQ is used to integrate with CICS.

**Target architecture**
Figure 4-9 on page 73 shows the target architecture of this solution. The original request can be sent to WebSphere Message Broker using a variety of protocols, including SOAP over HTTP and SOAP over JMS. Also, you can use non-Web Services protocols, such as HTTP, JMS, and FTP. In our solution, we are mainly
interested in the usage of standard Web Services protocols. After the necessary transformation and routing logic in WebSphere Message Broker, the request is put on an MQ queue. This queue is then used to trigger further processing in CICS.

**Figure 4-9  Target architecture using WebSphere Message Broker, WebSphere MQ, and CICS-MQ Bridge**

**Runtime software prerequisites**

The following software prerequisites apply to this solution:

- WebSphere Message Broker
- WebSphere MQ
- CICS Transaction Server with the CICS-MQ Bridge configured

In this solution, WebSphere Message Broker can run on z/OS, Linux on System z, or in a distributed environment. Messages can be placed on a queue remotely.

**Development tool**

This solution uses WebSphere Message Broker Toolkit for developing message flows to run in WebSphere Message Broker.
**Advantages**
This solution has the following advantages:

- All of the benefits from using WebSphere Message Broker as an ESB
- Proven communication mechanism between WebSphere Message Broker and CICS
- Loosely coupled

**Limitations**
There are no real limitations with this solution.

**Guidelines**
For more technical hands-on samples, refer to *Implementing an ESB using IBM WebSphere Message Broker V6 and WebSphere ESB V6 on z/OS*, SG24-7335.

You can refer to *WebSphere Business Integration Message Broker Basics*, SG24-7090, for the simple basics of Message Broker.

**Web service interface in WebSphere Message Broker and SOAP/JMS to CICS**
This solution is a mix of the solution that was discussed in “CICS Web Services” on page 59 and the solution that was discussed in “Web service interface in WebSphere Message Broker and WebSphere MQ to CICS” on page 72. A SOAP message is sent over WebSphere MQ from the Broker to CICS, and CICS Web Services support will receive, parse, and convert the SOAP message. CICS Web Services must be configured, and Web Services definitions have to be made in CICS for the CICS program that is being accessed.

**Target architecture**
The target architecture is shown in Figure 4-10 on page 75. Requests can be sent to WebSphere Message Broker using any of the protocols supported. After the necessary routing and transformation logic in the WebSphere Message Broker, a SOAP message gets built that is being transported to CICS through WebSphere MQ. CICS will receive this message as a JMS message, strips of the JMS, and a SOAP header and delivers the message in a COMMAREA format to the program.
Figure 4-10  Target architecture using WebSphere Message Broker, WebSphere MQ, JMS, and CICS Web Services

Runtime software prerequisites
The following runtime software prerequisites exist for this solution:

- CICS Transaction Server V3.1 or V3.2 with CICS Web Services configured
- WebSphere MQ
- WebSphere Message Broker

Tools
This solution is developed using the following tools:

- WebSphere Message Broker Toolkit for developing the message flows for the Broker
- Rational Developer for System z or the CICS Web Services Assistant for defining the Web service in CICS
**Advantages**
This solution has the following advantages:

- All of the benefits from using WebSphere Message Broker as an ESB
- Proven communication mechanism between WebSphere Message Broker and CICS, that is, WebSphere MQ
- Loosely coupled

**Limitations**
For limitations, refer to “Limitation” on page 61 regarding CICS Web Services and “Limitations” on page 74 regarding WebSphere MQ.

**Guidelines**
For more information about using WebSphere Message Broker:

- For more technical hands-on samples, refer to *Implementing an ESB using IBM WebSphere Message Broker V6 and WebSphere ESB V6 on z/OS*, SG24-7335.
- You can refer to *WebSphere Business Integration Message Broker Basics*, SG24-7090, for the simple basics of WebSphere Message Broker.

### 4.2 Accessing a complex CICS application as a service

An example of this pattern is a company with a large base of CICS applications, which were designed and built without much focus on the demarcation of types of logic and without focus on future service granularity.

### 4.2.1 Initial state

The existing applications are complex, rigid, and volatile with presentation, business, and data access layers all scattered around in hundreds of programs and copybooks. Due to the nature of these applications, maintenance and enhancement of the business functionality have become difficult and expensive. The business functionality is mixed with one or multiple complex CICS panels, that is, basic mapping support (BMS maps). In particular, the new users of the system find it hard to navigate through the panels and have to remember complex panel flows. The impact of any new change to a panel or a database causes issues with regard to the development, testing, and integration of the changes.
4.2.2 Current architecture

Figure 4-11 shows a simplified diagram of the current state.

![Diagram of current application architecture]

Figure 4-11  Current application architecture

4.2.3 Solution using code refactoring

This solution targets a better demarcation of types of logic and eventually a better organization of business logic. When the business logic is better organized, maintenance improves significantly and the integration of business logic into an SOA becomes much more effective. The solution for this pattern can be executed in phases and ideally involves a good refactoring tool. A tool, such as IBM Rational Asset Analyzer (RAA) and CICS Interdependency Analyzer (IA), can help you to obtain an organized view of the existing code and get an understanding of how components relate to each other.

After the code has been reorganized, Web services can be created from the business layer programs that can be consumed by other distributed and mainframe applications.

You collect an inventory of all the elements and analyze the source code. The data analyzed is then used to identify those programs that contain the mix of presentation, business, and data layers. You then extract the code and create
simple COMMAREA-based programs. Those programs can then be easily exposed as a CICS service provider using Rational Developer for System z (RDz) and the CICS Web Services functionality in CICS Transaction Server Version 3.1 or 3.2 (refer to Figure 4-12 on page 78).

You can adopt any one of the approaches that was explained in 4.1.4, “Solutions” on page 59 to “Web service interface in WebSphere Enterprise Service Bus and SOAP/JMS to CICS” on page 68 for enabling the refactored CICS code as a Web service provider.

![Figure 4-12 Rational Developer for z: Generate and deploy CICS Web Services](image)

**Target architecture**

After the code refactoring process, separate Web Services exist with the desired granularity. Certain business logic programs might still have their BMS map. Refer to Figure 4-13.
Chapter 4. Accessing programs on z/OS as a service

Runtime software prerequisites
The following runtime software prerequisites exist for this solution:

- CICS Transaction Server V3.1 or V3.2 with CICS Web Services configured
- WebSphere Application Server 6.1 (optional; only for RAA)
- DB2 V 7.0 or higher (optional; only for RAA)

Development tools
This solution is developed using the following tools:

- Rational Developer for System z (RDz) Version 7.1
  Use Rational Developer for System z for Web service-enabling CICS programs.
- Optionally, Rational Asset Analyzer (RAA) Version 5.5

Advantages
This solution has the following advantages:

- Better strategic reuse of existing assets with a better demarcation and better application architecture
- Effective tooling to help analyze and reproduce quality code
Continuing leverage of proven and robust programs

**Limitations**
We cannot really call this a limitation, but there is no tool that just converts an existing "spaghetti" application into a beautiful SOA-capable set of reusable services. In this solution, you have to factor in manual coding and lots of testing.

### 4.3 Accessing IMS programs as a service

One example of the pattern that we present here is a manufacturing company whose application portfolio comprises well designed IMS applications containing programs that have their presentation and business logic layers clearly demarcated and also contain stand-alone business modules. The company is looking for solutions that help its users avoid using the mainframe terminal displays and that also provide access to its existing IMS business logic programs from other sources, such as vendors who provide the necessary raw materials required for the manufacturing process. The vendors possess a wide variety of distributed systems as well as several mainframe systems.

#### 4.3.1 Initial state

The company contains a good number of IMS-based applications that use traditional character-based displays or “green screens” using Message Format Service (MFS) as an interface for sending and receiving the data. There is a need for the reuse of business logic and interfacing between IMS business logic programs and vendor programs for Enterprise Resource Planning (ERP) purposes.

#### 4.3.2 Current architecture

Figure 4-14 shows a simplified view of the initial architecture. There are 3270 terminal-based IMS applications accessing IMS databases, and there are numerous other applications in the company, such as Java and .Net.
4.3.3 Solution strategy

The solution strategy is to make the business logic programs accessible for a broad variety of potential client programs in a standardized way. Quality of service (QoS) plays an important role, too. At this point, you can make one of roughly three architectural decisions:

- Web service-enable selected IMS business logic programs and provide the Web Services Description Language (WSDL) to those individuals who want to access those services (directly).
- Do not Web service-enable the IMS business logic program, but implement a Web service interface in WebSphere Application Server and access the IMS program from WebSphere Application Server through any of the supported protocols and connectors, for example, J2C or WebSphere MQ.
- Do not Web service-enable the IMS business logic programs, but provide an ESB layer with access to those IMS programs using native or traditional access methods, such as Open Transaction Manager Access (OTMA) and WebSphere MQ. The access point for those IMS programs for clients will then be in the ESB.
All solution directions can be considered standardized and SOA-compliant. However, the ESB direction is a better architecture, providing advantages in the area of governance, flexibility, quality of service, and more. After all, just providing a Web Services interface without using an ESB remains a “point to point” solution, but just with another interface protocol.

In 4.3.4, “Solutions” on page 82, we discuss solutions of all categories.

**Benefits**
The benefits of providing broader and more standardized access to IMS business logic programs are:

- Leverage robust and proven business logic with high performing native data access
- Avoid building redundant programs throughout the company that perform the same functions as the IMS programs, but usually with a lower quality of service

4.3.4 Solutions

We discuss the following solutions:

- IMS Web Services
- WebSphere Application Server and the IMS TM Resource Adapter
- WebSphere Enterprise Service Bus and IMS TM Resource Adapter
- WebSphere Message Broker and IMS TM Resource Adapter
- WebSphere Message Broker and WebSphere MQ-IMS Bridge
- WebSphere Enterprise Service Bus and WebSphere MQ-IMS Bridge

**IMS Web Services**

An easy solution is to use the IMS SOAP Gateway to expose the IMS business logic programs as Web services. IMS SOAP Gateway is a solution that enables IMS applications to interoperate outside of the IMS environment through the SOAP protocol to provide and request services that are independent of platform, environment, application language, or programming model.

In the case where IMS acts as a service provider, various types of client applications, such as Microsoft® .NET, Java, and third-party applications, can submit SOAP requests to IMS SOAP Gateway to drive business logic. In conjunction with the IMS Connect XML adapter, you can enable your IMS application to become a Web service without the need of changing the back-end IMS application. A step-by-step technical hands-on implementation for IMS as a service provider is available in Chapter 10, “Exposing an IMS application as a Web service” on page 289.
**Target architecture**

Figure 4-15 shows the target architecture for the solution using IMS SOAP Gateway.

![Target architecture using IMS SOAP Gateway on z/OS](image)

**Runtime software prerequisites**

The following runtime software requirements exist for this solution:

- IMS Version 10 or higher
- IMS Connect Version 10 or higher
- IMS SOAP Gateway Version 10.1 or higher

**Development tool**

We used Rational Developer for z (RDz) Version 7.1.1 (or higher) to develop this solution.

Refer to the following Web site for details regarding the required level of development tooling:

http://www-01.ibm.com/software/data/ims/soap/

**Advantages**

The advantages of this solution are:
This solution is a simple and easy way to service-enable IMS transactions.

Existing transactions, which have been proven, secure, reliable, and scalable for many years, can still be retained.

**Limitations**

IMS SOAP Gateway has the following limitations:

- Only non-conversational transactions are supported.
- Only commit mode 1, sync level NONE processing is supported.
- MFS-based transactions are not supported.
- Two-phase commit is not supported.

**Guidelines**

You can use tools, such as Rational Developer for z (RDz), to generate the Web Service artifacts that cover all possible data definitions.

**Example**

Refer to Chapter 10, “Exposing an IMS application as a Web service” on page 289 for a hands-on example of this solution.

**WebSphere Application Server and the IMS TM Resource Adapter**

The *IBM IMS TM Resource Adapter* is used by Java applications, Java 2 Platform, Enterprise Edition (J2EE) applications, or Web services to access IMS transactions that are running on host IMS systems. In addition, the IMS TM Resource Adapter implements the J2C *Common Client Interface (CCI)*, a programming interface that allows your application to communicate with IMS Transaction Manager. The IMS TM Resource Adapter can be used with a J2EE server, such as IBM WebSphere Application Server. The IMS TM Resource Adapter also enables an IMS application to act as a client to invoke applications in a J2EE server.

**Target architecture**

The architecture of this solution is depicted in Figure 4-16 on page 85.
Figure 4-16 Target architecture: WebSphere Application Server and IMS TM Resource Adapter

**Runtime software prerequisites**
The runtime software prerequisites for this solution are:

- IMS Version 9 or higher
- IMS Connect Version 9 or higher
- WebSphere Application Server Version 6 or higher on z/OS or in a distributed environment

Refer to the following Web site for the exact details regarding software levels:
http://www-01.ibm.com/software/data/ims/ims/components/tm-resource-adapter.html#sysreqs
**Development tool**
This solution can be developed using Rational Application Developer (RAD) Version 6.

Refer to the following Web site for the exact details regarding the supported combinations of development tools and runtime software:

http://www-01.ibm.com/software/data/ims/ims/components/tm-resource-adapter.html#sysreqs

**Advantages**
Advantages of this solution are:

- A quick way of creating J2C applications
- Two-phase commit support offered by WebSphere Application Server, the IMS TM Resource Adapter, and IMS Connect
- No change to existing IMS transactions

**Limitation**
A limitation of this solution is that it is still tightly coupled; this solution is more oriented to a peer-to-peer communication environment.

**Guidelines**

**WebSphere Enterprise Service Bus and IMS TM Resource Adapter**
This solution is similar to the solution that we discussed in “WebSphere Application Server and the IMS TM Resource Adapter” on page 84. In this solution, however, the Web service interface is implemented in WebSphere Enterprise Service Bus instead of WebSphere Application Server. The development process differs slightly though, because WebSphere Enterprise Service Bus requires WebSphere Integration Developer as a development tool. For WebSphere Application Server, a Web service interface is developed as a J2EE application, but a mediation module is developed for WebSphere Enterprise Service Bus.

**Target architecture**
Figure 4-17 on page 87 depicts the architecture of this solution.
Figure 4-17  Target architecture: WebSphere Enterprise Service Bus and IMS TM Resource Adapter

**Runtime software prerequisites**
The following runtime software prerequisites exist for this solution:

- IMS Version 9 or higher
- IMS Version 9 or higher
- WebSphere Enterprise Service Bus Version 6 or higher on z/OS or in a distributed environment

Refer to the following Web site for the exact details regarding software levels:

http://www-01.ibm.com/software/data/ims/ims/components/tm-resource-adapter.html#sysreqs

**Development tool**
This solution is developed using WebSphere Integration Developer Version 6 or higher.
**Advantages**
The advantages of this solution are:

- Data format transformation provided by WebSphere Enterprise Service Bus
- Transport protocol transformation provided by WebSphere Enterprise Service Bus
- Two-phase commit support offered by WebSphere Application Server, the IMS TM Resource Adapter, and IMS Connect
- No change to existing IMS transactions

**Limitation**
This solution does not have any real limitations.

**Guidelines**
There are many books that are already available containing more details about WebSphere Enterprise Service Bus:

- *Implementing an ESB using IBM WebSphere Message Broker V6 and WebSphere ESB V6 on z/OS*, SG24-7335
- *Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB*, SG24-7369

**WebSphere Message Broker and IMS TM Resource Adapter**
This solution is similar to the solution that we discussed in “WebSphere Enterprise Service Bus and IMS TM Resource Adapter” on page 86. In this solution, however, the Web service interface is implemented in WebSphere Message Broker instead of WebSphere Enterprise Service Bus. The development process differs slightly though, because WebSphere Message Broker requires WebSphere Message Broker Toolkit as a development tool. For WebSphere Enterprise Service Bus, a Web service interface is developed as a mediation module, but a message flow is developed for WebSphere Message Broker.

**Target architecture**
Figure 4-18 on page 89 depicts the architecture of this solution.
Runtime software prerequisites
The following runtime software prerequisites exist for this solution:

- IMS Version 10 or higher
- IMS Version 10 or higher
- WebSphere Message Broker Version 6.1.0.3 or later

Refer to the following Web site for the exact details regarding software levels:
http://www-01.ibm.com/software/data/ims/ims/components/tm-resource-adapter.html#sysreqs

Development tool
This solution is developed using WebSphere Message Broker Toolkit Version 6.1.0.3 or later.

Advantages
The advantages of this solution are:

- Data format transformation provided by WebSphere Message Broker
Transport protocol transformation provided by WebSphere Message Broker
Two-phase commit support offered by WebSphere Message Broker, the IMS TM Resource Adapter, and IMS Connect
No change to existing IMS transactions

Limitations
This solution does not have any real limitations.

Guidelines
There are many books already available that contain more details about WebSphere Enterprise Service Bus:

- Implementing an ESB using IBM WebSphere Message Broker V6 and WebSphere ESB V6 on z/OS, SG24-7335
- Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB, SG24-7369

WebSphere Message Broker and WebSphere MQ-IMS Bridge
This solution integrates existing IMS MQ-based applications and exposes them as a Web service. IBM MQSeries®, which WebSphere MQ was previously called, has brought the first stage of loosely coupled applications to IMS. Combined with the WebSphere MQ-IMS Bridge, IMS applications can now integrate with external applications asynchronously. This principle is taken to the next stage with WebSphere Message Broker by adding broad functionality in the area of routing and conversion.

In those environments where the WebSphere MQ-IMS Bridge is already in place, WebSphere Message Broker can be added without changing anything in the IMS environment. WebSphere Message Broker will just insert messages into the (existing) queue, and IMS MQ applications consume these messages as usual. WebSphere Message Broker takes over the role of the MQ client application.

Figure 4-19 on page 91 shows the new target architecture.
Target architecture

![Target architecture diagram](image)

**Runtime software prerequisites**

The following software prerequisites exist for this solution:

- WebSphere MQ
- WebSphere Message Broker on z/OS or in a distributed environment
- WebSphere MQ-IMS Bridge, which includes the triggering mechanism of IMS transactions with local queues defined on z/OS

**Development tool**

This solution is developed using WebSphere Message Broker Toolkit.

**Advantages**

This solution offers the following advantages:

- Flexible communication
- Simplified communication
- Loosely coupled
- Solution not impacted by the complexity of the infrastructure
**Limitations**

There are limitations when you import the COBOL files in the wizard, such as the DEPENDING-ON clause, and hexadecimal binary values are not supported. Refer to the WebSphere Message Broker toolkit documentation.

**Guidelines**

We recommend:

- For more technical hands-on samples, refer to *Implementing an ESB using IBM WebSphere Message Broker V6 and WebSphere ESB V6 on z/OS*, SG24-7335.

- You can refer to *WebSphere Business Integration Message Broker Basics*, SG24-7090, for an introduction to WebSphere Message Broker.

**WebSphere Enterprise Service Bus and WebSphere MQ-IMS Bridge**

This solution is similar to the solution that was discussed in “WebSphere Message Broker and WebSphere MQ-IMS Bridge” on page 90. The difference is that another ESB product is used to implement the service interface. The process for developing the solution is similar too, but another tool is used and various types of components are developed, that is, mediation modules for WebSphere ESB as opposed to message flows for WebSphere Message Broker.

Figure 4-20 on page 93 shows the architecture of this solution.
**Target architecture**

![Target architecture: WebSphere ESB and WebSphere MQ-IMS Bridge](image)

**Runtime software prerequisites**
The following software prerequisites exist for this solution:

- WebSphere MQ
- WebSphere Enterprise Service Bus on z/OS or in a distributed environment
- WebSphere MQ-IMS Bridge, which includes the triggering mechanism of IMS transactions with local queues defined on z/OS

**Development tool**
This solution is developed using WebSphere Integration Developer.

**Advantages**
This solution offers similar advantages to the solution that was discussed in “Web service interface in WebSphere Message Broker and WebSphere MQ to CICS” on page 72.
Limitations
There are no real limitations for this solution.

Guidelines
- You can also refer to this journal article “Building an Enterprise Service Bus using WebSphere ESB, Part 4” by Rachel Reinitz and Andre Tost, 28 February 2006:
  

4.4 Accessing a CICS application as a composite service

As companies move to more closely align their IT investments with their business priorities and strategies, many companies find that a service-oriented architecture (SOA) provides the desired resiliency and flexibility that enables their IT organizations to more rapidly respond to a changing competitive and regulatory landscape in a more robust and scalable manner. These companies have also embarked on endeavors to map their business processes to their IT applications. Often, they find that their existing applications can become building blocks or nodes, which can be built into a new application to support those processes. As part of this effort, they also find that external applications that provide common industry functionalities can be utilized to optimize software development cycles. Being able to create a composite application that coordinates the invocation of internal applications and external services is the ideal solution.

4.4.1 Initial state

A fictional company contains multiple CICS applications that are 3270 terminal-based and COMMAREA-based. In addition, the fictional company has a requirement to call an external service provider to obtain information that is needed to support a prescribed business process.

4.4.2 Current architecture

A client application is written to support a business process. This application leverages existing CICS applications and also makes a call to an external Web service provider. The CICS applications are both 3270 terminal-based and COMMAREA-based. The client application has to make multiple invocations to
the CICS system and one Web service call. It then aggregates the data that is returned from these sources into a consolidated result. While this process achieves the desired result, it is expensive and involves longer instruction path lengths because of the multiple invocations. The client application developer also needs to be immersed in the complexity of writing controlling logic that coordinates the various invocations and aggregates the varied results.

**Figure 4-21  Client application with multiple invocations**

### 4.4.3 Solution

Next, we describe the solution for this scenario.

**Target architecture**

The CICS Service Flow feature consists of both development tooling and run time. The development tooling is provided as part of Rational Developer for System z (RDz) and provides the environment where processes can be modeled graphically with nodes representing CICS applications and outbound Web service calls. After these processes are modeled, the Service Flow modeling component of RDz can be used to generate a Service Flow application and attendant artifacts that are then deployed to the CICS Service Flow run time running on CICS Transaction Server on z/OS. The deploy Service Flow application can be accessed just as any CICS COMMAREA-based application, or a service interface can be described and implemented that enables various types of client applications to be written to perform a single service invocation to obtain the results that were aggregated on the CICS system.
Runtime software prerequisites
This solution requires IBM CICS Transaction Server for z/OS V3.1 or V3.2 with the Service Flow Feature configured.

Development tool
Service Flow applications are developed using IBM Rational Developer for System z V7.1 or higher.

Advantages
This solution has the following advantages:
► Process integration and coordination within CICS
► Reuse of existing CICS applications without the need to change them
► Simplification of the integration of CICS applications into an SOA by aggregating service invocations

Guidelines
Refer to the following publications for guidelines:
► Developing CICS Web Services, SG24-7126
► Implementing CICS Web Services, SG24-7206

Example
We describe a detailed example in Chapter 11, “Building composite CICS services using the Service Flow Feature” on page 307.
4.5 Accessing DB2 stored procedures on z/OS as a service

Next, we describe accessing DB2 stored procedures on z/OS as a service.

4.5.1 Initial state

DB2 stored procedures are an excellent way to implement business logic in a single, centralized place, namely, in the database rather than in application programs.

You have existing DB2 stored procedures that encapsulate business logic. You want to make this business logic available within the enterprise, but you want to avoid coupling the users to a specific technology, or even making them aware of the underlying technology.

4.5.2 Solution

As of now, DB2 by itself cannot act as a Web service provider. Therefore, the service has to be hosted in an application server, such as CICS or WebSphere.

An easy way to expose a DB2 stored procedure is to use Data Web Services tooling, which produces a J2EE-compliant Web archive (WAR) file that can be deployed to any supported application server.

Target architecture
Figure 4-23 on page 98 shows the target architecture for this solution.
Runtime software prerequisites
The following runtime software prerequisites exist for this solution:

- Data Web Services run time
- J2EE application server supported by Data Web Services (might not be on z/OS)

Development tool
This solution is developed using IBM Data Studio Developer Version 1.2 or higher.

Example
We show an example for this solution in 7.3, “Exposing a DB2 stored procedure as a service” on page 172.
4.6 Accessing presentation logic as a service

An example of the pattern that we present here is a company that has many 3270 terminal-based applications that have traditionally been accessed only by internal users from the internal network of the company. Customer service representatives have to access multiple applications in order to effectively respond to customer service requests. With the ubiquity of Web browsers and the relative ease of access to information on the Internet, providing customers access to their information through self-service applications has become a standard business practice for many companies and a standard customer expectation. This capability requires transforming the 3270 terminal-based user interface of these applications into Web-based user interface. In addition, to facilitate potential interactions between other partners, providing a Web service interface is also necessary for optimal interoperability.

4.6.1 Initial state

This company has a set of 3270 terminal-based applications that run on CICS and IMS. The user interface is primarily through CICS BMS-based and IMS MFS-based displays that are accessed through 3270 terminals or emulated sessions. A user might have to access multiple applications to obtain all the needed information. Navigation is thus not limited to multiple panels within one application but likely through multiple applications that are accessed through multiple terminals or sessions.

4.6.2 Current architecture

Figure 4-24 on page 100 depicts the current architecture.
4.6.3 Solution: Using HATS as a service enabler

For competitive and tactical reasons, the solution for this scenario needs to be delivered in a short time frame. In addition, the existing 3270-based terminal interface needs to be maintained in order to minimize operational disruptions. This requirement precludes the wholesale replacement of existing systems through either a rewrite or the adoption of a package solution. A quick, tactical solution with a shorter time to application delivery is to use a user interface transformation technology, such as IBM Rational Host Access Transformation Services (HATS).

HATS enables the transformation of terminal applications quickly and easily. With HATS, 3270-based and 5250-based applications can be extended to the Web, to a portlet, to a rich client, or to browsers on mobile devices with low risk. With HATS, you can also:

- Improve the workflow and navigation of your host 3270 and 5250 applications, without any access or modification to the application source code.
- Create service-oriented architecture (SOA) assets from logic that is contained in your terminal applications.
- Support a variety of mobile devices, such as cell phones, data collection terminals, and personal digital assistants (PDAs).

- Create portlets that conform to the Java Portlet Specification (JSR) 168.

The enterprise applications that are built with HATS are deployed to and run on either IBM WebSphere Application Server or IBM WebSphere Portal.

To facilitate the development of HATS applications, the IBM Rational HATS Toolkit is a set of plug-ins for the Eclipse-based IBM Rational Software Delivery Platform. The HATS Toolkit lets you develop a new application that generates a GUI for your 3270 and 5250 applications. These applications can be developed one step at a time. Over time, the HATS application can be streamlined, making it easier to use than the host applications whose data it presents.

The development process for building HATS applications that are accessed from a browser, portal, rich client, and mobile device is similar. The GUI that you create can be customized to suit your business needs and standards. You can hide unnecessary information, organize data into tables, or display only required input fields. You can provide a drop-down list of valid values for an input field, change the size and location of certain text, and provide navigation buttons.

In addition to the basic customization capabilities, you can create HATS macros to provide streamlined navigation through multiple host panels or combine data from several host panels. From these HATS macros, you can then generate HATS integration objects, which are Java beans that encapsulate interactions with a host application. These integration objects can then be used to create Web pages and Web services.
Target architecture
The target architecture is shown in Figure 4-26.

![Diagram showing the target architecture using HATS](image)

Runtime software prerequisites
This solution has the following runtime software prerequisites:

- IBM Rational Host Access Transformation Services for Multiplatforms V7.1 or higher
- IBM WebSphere Application Server V6.0 or higher

Development tools
The following tools are used for development:

- IBM Rational HATS Toolkit V7.1 or higher
- One of the following software delivery platforms:
  - IBM Rational Application Developer for WebSphere Software V7.1 or higher
  - IBM Rational Software Architect for WebSphere Software V7.1 or higher
– IBM Rational Developer for System z for SOA Construction with EGL V7.1 or higher
– IBM Rational Developer for System z for SOA Construction with Java V7.1 or higher
– IBM Rational Business Developer V7.1 or higher

Advantages
The advantages of this solution are:

► Shortened application delivery time
► Minimal operational disruption
► Improved productivity from reducing the number of panel interactions and application navigations
► Reduced training costs

Limitations
Limitations of this solution are:

► Requires IBM WebSphere Application Server, introducing an additional middleware layer in certain cases
► Integrates only panel-based applications

Best practices
The best practices or guidelines for this solution are:

► Design the user interface from the requirements perspective, which applies to most development projects that involve a human interface. Determine how information is to be presented to the user before you start creating the interface.
► Identify any established Web page templates or conventions that are used within your company.
► Document conventions for:
  – Layout of Web pages
  – Expressing names, addresses, phone number, currency, and so forth
  – Format headers
  – Breaking up logically grouped form data
  – Where and when to use radio buttons, check boxes, or drop-down lists for various kinds of input data
  – Code formatting and other settings
- Create a story board for panel flows, which is especially important when you are controlling process flows.
- Identify panels that can be modified to make them easier to transform.
- Identify panels that are the same but with differences in authorized inputs based on login credentials.
- Identify all static data that can be string-replaced in the tool as compared to codes or inputs that defy hard-coded transformations.
- Identify panels that will be combined as early as possible.
- Create the HTML layouts of each window before putting in the host components or widgets, because moving HATS components or widgets around on the window is in many cases easier than rearranging the base layout.
- Use a source control management system, which is especially critical when you have a team of developers.
- Divide a project into component ownership areas for easier development.
Integrating external services in existing z/OS applications

In this chapter, we describe patterns with technology solutions to reuse existing or new external services by integrating them into existing z/OS applications.

We discuss the following patterns:

- In Integrating external Web Services into CICS applications, we discuss patterns to integrate external services into CICS applications:
  - Using CICS outbound Web Services support
  - Using an enterprise service bus
  - Using DB2 outbound Web Services support

- In Integrating external Web services with IMS applications, we discuss a pattern to integrate external services into IMS applications:
  - Using IMS asynchronous callout support

- In Integrating external services into batch applications, we discuss patterns to integrate external services into z/OS batch applications:
  - Using messaging
  - Using messaging over an ESB
  - Using Web services over JMS transport
  - Using a Java Web services client
  - Using DB2 outbound Web Services support
5.1 Integrating external Web Services into CICS applications

An example of the pattern explained here is a company whose application portfolio includes both z/OS CICS applications and non-mainframe applications to support key supply chain processes. The z/OS CICS applications strongly benefit from having access to external services inside and outside the company, which is especially true for those CICS applications that use input from vendor applications and provide output to client applications. Traditionally, input has been entered into CICS panels manually based on vendor information and output has been provided through a batch interface process.

5.1.1 Initial state

In this pattern, the CICS applications are used for internal use only, but they are frequently maintained to include new functionality. It is recognized that, in certain cases, the new functionality that is required in these CICS applications exists elsewhere in the company, typically in other environments or on other platforms. New service-oriented architecture (SOA) technology and features in CICS make it easy now to reuse functionality outside CICS.

As mentioned earlier, certain input from vendors is still entered manually through 3270 terminals.

5.1.2 Current architecture

Figure 5-1 on page 107 depicts the current architecture.
5.1.3 Solution strategy

The solution strategy lies in providing access to external services from existing CICS applications. We will look at the primary outbound connectors in CICS, including Web Services support and WebSphere MQ.

Special attention needs to be paid to the transactional behavior and the quality of service (QoS) of the CICS transactions. An outbound service call must not jeopardize the existing service level of the current CICS transactions.

5.1.4 Solutions

We discuss the following solutions:

- Using CICS outbound Web Services support
- Using an enterprise service bus
- Using DB2 outbound Web Services support

Using CICS outbound Web Services support

In this solution, the Web Services Description Language (WSDL) file of the existing Web service is used to convert the XML interface structure of the distributed service into a COBOL copybook structure using Rational Developer for System z (RDz) or the CICS Web Services Assistant (WSA). RDz and WSA create both the request and response structures in the form of a COBOL copybook. RDz can assist you to create a CICS Web Services requestor.
A program that can be updated with the simple basics of a CICS program, such as populating request and response structures and validating error messages. RDz-generated artifacts, such as the service requestor program and WSBind file, are then deployed to z/OS. The external Web service can be hosted in many places. Refer to Figure 5-2.

**Target architecture**

Figure 5-2 depicts the target architecture of this solution.

---

**Runtime software prerequisites**

The following software is required:

- CICS Transaction Server Version 3.1 or 3.2
- WebSphere Application Server or other Web service container

**Development tool**

You can develop this solution with either Rational Developer for System z (RDz) or CICS Web Service Assistant (WSA)
Advantages
This solution has the following advantages:

- This solution is a simple and easy way to integrate CICS applications with external Web services.
- There is no impact to the existing CICS applications.
- The CICS service requestor solution can be extended to use an enterprise service bus (ESB) to get more benefits.

Limitations
There are no additional restrictions in this solution other than the limitation that is explained in 12.9, “Handling complex CICS COMMAREA structures” on page 415, which is not a solution limitation but is due to a COBOL language limitation.

Guidelines
We recommend:

- Use Rational Developer for System z (RDz) instead of CICS-supplied JCL utilities to generate CICS Web Service artifacts.
- In cases where you have a complex-type structure defined in the XML data structure of the external Web Service WSDL file, refer to the instructions in “Handling complex CICS COMMAREA structures” on page 415.
- In situations where you need only part of the existing CICS program integrated into the external Web service, you need to write a wrapper program. Refer to “Using an enterprise service bus” on page 109.

Example
Refer to Chapter 12, “Integrating an external Web service with a CICS application” on page 385 for a hands-on example.

Note: To understand more of the concepts of CICS as a service requestor, refer to Chapter 6 of Application Development for CICS Web Services, SG24-7126.

Using an enterprise service bus
Another approach to integrate external services into existing CICS applications is to use an enterprise service bus (ESB). You can use mediation components to perform transformation and routing between the CICS requestor and the Web service. An ESB supports multiple data and transport protocols, such as SOAP, Java Message Service (JMS), MQ, HTTP, and HTTPS. Refer to Figure 5-3 on page 110.
**Target architecture**

Figure 5-3 shows the target architecture. Note that the protocol that is used between the service requestor in CICS and the ESB can be any of the protocols that are supported by both CICS and the ESB. In most cases, those protocols are SOAP over HTTP or JMS, JMS, or WebSphere MQ.

![Diagram of target architecture](image)

Figure 5-3  Integration of CICS application and external Web Services using an ESB

For a more detailed understanding of WebSphere Message Broker and WebSphere ESB, refer to *Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB*, SG24-7369.

In order to provide more flexibility in these solutions, the Web services can be published using a registry, such as WebSphere Service Registry and Repository, which eliminates any dependency on the endpoints. For example, a service requestor does need not know the service provider location. For a detailed understanding of WebSphere Service Registry and Repository, refer to *WebSphere Service Registry and Repository Handbook*, SG24-7386.

**Runtime software prerequisites**

This solution requires the following software products:

- CICS Transaction Server Version 3.1 or 3.2 with Web Services support configured
- Enterprise Service Bus (ESB) product on z/OS or in a distributed environment
Development tools
This solution is developed using the following tools:

- Rational Developer for System z (RDz) or CICS Web Service Assistant (WSA)
- Development tool for the ESB to be used

Advantages
This solution has the following advantages:

- Flexibility from the usage of the ESB
- No major changes to existing CICS applications

Limitations
No real limitations exist for this solution.

Guidelines
Refer to Implementing an ESB using IBM WebSphere Message Broker V6 and WebSphere ESB V6 on z/OS, SG24-7335.

Using DB2 outbound Web Services support
As an alternative to CICS as a service requestor, if you have CICS DB2-based applications, you can use DB2 outbound Web Services support to invoke external Web services. In this case, DB2 even takes care of XML parsing (refer to Figure 5-4 on page 112).

Target architecture
Figure 5-4 on page 112 shows the target architecture for this solution.
Figure 5-4  Integration of CICS application and external Web services using DB2 outbound service

**Runtime software prerequisites**
Refer to Chapter 14, “Using DB2 as a Web service requester” on page 455 for details about the runtime software prerequisites.

**Development tools**
Refer to Chapter 14, “Using DB2 as a Web service requester” on page 455 for details about the development tools.

**Advantages**
An advantage of this solution is that you do not need to implement any additional outbound communication channels from a CICS perspective.

**Limitations**
Refer to “Using DB2 outbound Web Services support” on page 123 for limitations in the DB2 outbound Web service support.

**Guidelines**
We explain all of the guidelines in the technical hands-on example in Chapter 12, “Integrating an external Web service with a CICS application” on page 385.
Example
Refer to Chapter 14, “Using DB2 as a Web service requester” on page 455 for an example.

5.2 Integrating external Web services with IMS applications

An example of the pattern that we explain here is a company whose application portfolio includes both z/OS IMS applications and non-mainframe applications to support key supply chain processes. The z/OS IMS applications will strongly benefit from having access to external services inside and outside the company, which is especially true for those IMS applications that use input from vendor applications and provide output to client applications. Traditionally, input has been entered into IMS panels manually based on vendor information and output has been provided through a batch interface process.

5.2.1 Initial state

In this pattern, the IMS applications are used for internal use only, but they are frequently maintained to include new functionality. It is recognized that, in certain cases, the new functionality that is required in these IMS applications exists elsewhere in the company, typically in other environments or on other platforms. New SOA technology and features in IMS make it easy now to reuse functionality outside IMS.

As mentioned earlier, certain input from vendors is still entered manually through 3270 terminals.

5.2.2 Current architecture

Figure 5-5 on page 114 shows the current architecture of this scenario.
5.2.3 Solution strategy

The solution strategy lies in providing access to external services from existing IMS applications. We will look at the primary outbound connectors in IMS, including IMS Callout support and WebSphere MQ.

Special attention needs to be paid to the transactional behavior and the quality of service (QoS) of the IMS transactions. An outbound service call must not jeopardize the service level of these existing CICS transactions.

5.2.4 Solutions

The solutions for IMS are similar to those solutions for CICS. There are three principal solutions:

- Using IMS asynchronous callout support
- Use an enterprise service bus (ESB), similar to “Using an enterprise service bus” on page 109 and not discussed again
- Use DB2 Outbound Web Services support, similar to “Using DB2 outbound Web Services support” on page 111 and not discussed again
Using IMS asynchronous callout support

IMS SOAP Gateway is a lightweight Web Services solution that enables IMS applications to interoperate in a service-oriented architecture (SOA). IMS asynchronous callout support can be used to access a Web service through the IMS SOAP Gateway. With this function, users can develop new applications or extend their existing IMS applications to access new business logic or data outside IMS using Web Services protocols.

The IMS SOAP Gateway enables IMS applications to make asynchronous callout requests to external Web services and optionally receive responses back.

Rational Developer for System z (RDz) provides wizards that generate the XML converter and the correlator files needed to enable the IMS application to run as a Web service consumer.

**Target architecture**

Figure 5-6 shows the target architecture for an IMS application using asynchronous callout to access a Web service.

![Figure 5-6 Integration of IMS application and external Web services](image-url)
**Runtime software prerequisites**
The following software is required on z/OS to implement an asynchronous callout solution:

- IMS Version 10 or higher
- IMS Connect Version 10 or higher
- IMS SOAP Gateway Version 10 or higher

**Development tool**
Rational Developer for System z (RDz) Version 7.1.1 or higher is used for development.

**Advantages**
This solution has the following advantages:

- This solution is a simple and a easy way to integrate IMS applications with external Web Services.
- The time to develop these solutions is quicker than other approaches.
- The IMS Service requestor solution can be extended to use ESB or a Message Broker to get more benefits.

**Limitations**
IMS SOAP Gateway Version 10 has the following limitations:

- Only non-conversational transactions are supported.
- Only commit mode 1, sync level NONE processing is supported.
- Message Format Service (MFS)-based transactions are not supported.
- Two-phase commit is not supported.

**Guidelines**
We provide more detailed guidance in Chapter 13, “Integrating an external Web service into an IMS application” on page 419, but important considerations are:

- You must modify or create your IMS application for sending callout requests and receiving response messages from an external Web service if necessary.
- Optionally, an IMS database is needed to store the data necessary to correlate the callout response to the initiating IMS client, for example, the LTERM name of the initiating IMS client.
- The Open Transaction Manager Access (OTMA) destination routing descriptor enables an IMS application to route callout requests to services that are accessible to the IMS SOAP Gateway without the need to code routing exits.
Example
We provide a technical hands-on example in Chapter 13, “Integrating an external Web service into an IMS application” on page 419.

5.3 Integrating external services into batch applications

The setting for this pattern is a typical z/OS environment with daily, weekly, and monthly batch jobs and cycles. Many of these jobs perform functions in which they can benefit from accessing certain services directly. Traditionally, batch applications access data directly and write output to files, printers, or databases again. Batch applications can also take input from WebSphere MQ queues.

5.3.1 Initial state

You have an existing batch application, which is written in a traditional language, such as COBOL or PL/I, that needs to be enhanced by integrating an external service. For example, you might need to integrate a credit card validation service that is hosted on a distributed platform.

5.3.2 Solution strategy

The solution strategy involves providing access to external services from existing batch applications. Because batch applications are carefully designed to fit a certain processing window, outbound service calls need to be carefully evaluated and designed. Performing an outbound service call and waiting for a response message back is usually many times slower than a direct database call. Also, when batch applications involve high volumes, outbound service calls are not an obvious choice. We only recommend looking into this pattern if:

- The data accessed in the external Web service is not available locally in a database.
- The amount of outbound service calls is reasonable in the context of the batch program processing time and its successor programs’ processing times.
5.3.3 Solutions

The solutions that we discuss for performing outbound Web service calls from batch applications are:

- Using messaging
- Using messaging over an ESB
- Using Web services over JMS transport
- Using a Java Web services client
- Using DB2 outbound Web Services support

Using messaging
A traditional way to solve this problem is to establish a messaging connection between the two applications, that is, connecting the applications via WebSphere MQ. For example, the application providing the service can be exposed as a Message-Driven Bean (MDB) hosted in WebSphere Application Server.

Target architecture
The solution in this case is a batch program that inserts messages into a WebSphere MQ queue and a target service that retrieves those messages from the queue. If needed, the service sends a response message back either with full data in certain cases and in other cases perhaps with just a return code. WebSphere MQ does not provide protocol conversion or data format conversion, so the message needs to be sent in the format in which the target service expects it. If the service is a Web service, the expected format is likely to be SOAP over JMS. The batch program needs to insert SOAP messages into the WebSphere MQ queue, which most batch programs that are written in traditional languages cannot easily do. The least intrusive method for an existing batch program is to insert plain MQ messages into a queue and let another layer (Java 2 Platform, Enterprise Edition (J2EE) layer or an ESB) sort out eventual conversion issues.

Runtime software prerequisite
This architecture requires WebSphere MQ.

Development tools
The development tool to be used for this pattern depends on the programming language in use for the existing batch application. If plain MQ messages are used, you can continue to use your existing development tooling for your batch environment.
**Advantages**
Advantages of this solution are:

- Easy and proven way to connect two applications
- In most cases, no additional software required

**Limitations**
Limitations of this solution are:

- Most traditional batch applications that are written in languages, such as PL/1 and COBOL, cannot easily send SOAP/JMS messages.
- Batch applications might slow down significantly if many messages are being exchanged with an outside service and if the program has to wait for a response message each time.
- There might be loose runtime coupling between applications but high development-time coupling, because the applications have to agree on a common message format.
- A homegrown message format might prevent reuse, especially when using a binary format, such as COBOL data structures (as opposed to an XML-based format, for example).
- There is a risk of a high amount of coupling between the two applications, creating “MQ spaghetti.”

**Using messaging over an ESB**
In this solution, the interface to the Web service is exposed as a “traditional” COBOL or PL/I language structure. The application uses WebSphere MQ messaging to put the request structure to a queue. An ESB retrieves the request messages from the queue and converts them to the format that is expected by the target service (for example, a SOAP request message) using its transformation capabilities. It then invokes the target service and converts the response back to an appropriate language structure.

This solution is similar to the previous simple messaging solution. The major advantage is that it leverages the “messaging switchboard” capabilities of the ESB, avoiding the tight coupling between service consumer and service provider.

**Target architecture**
In this solution, the batch program inserts request messages into a WebSphere MQ queue, ideally, structures that fit the existing programming language being used for the batch program. The messages are delivered to an MQ or JMS input node of an ESB solution, such as WebSphere Message Broker or WebSphere ESB. The ESB performs protocol conversion, data format conversion, service lookups, and eventual routing functions. A traditional MQ message can thus be
converted and delivered in SOAP/XML to a target Web service in any environment and on any platform. The response messages will follow the reverse path back to the batch program.

**Runtime software prerequisites**
This architecture requires an ESB product, such WebSphere ESB or WebSphere Message Broker.

**Development tools**
The development tool to use for this pattern depends on the programming language in use for the existing batch application and on the ESB product that is chosen.

**Advantages**
The advantages of this solution are:
- Loose coupling between applications, both at run time and at development time
- Minimum impact for the calling batch applications, because they do not necessarily need to use the message format required by the service provider
- Central point of administration and control

**Using Web services over JMS transport**
At the time of writing this book, there is no predefined, ready to use support for Web Services calls from pure COBOL (or PL/I) over HTTP or HTTPS. However, SOAP over JMS transport is a feasible approach. Enterprise COBOL has built-in support for XML generation and parsing, and sending and receiving messages via WebSphere MQ is daily business for most COBOL programmers.

**Target architecture**
The target architecture for this solution is shown in Figure 5-7 on page 121. The batch service consumer sends SOAP messages to a WebSphere MQ queue, and the service provider processes these messages. The SOAP reply message is sent back from the service provider to the batch program.
**Runtime software prerequisite**
Enterprise COBOL or Enterprise PL/1 is required on z/OS for XML parsing.

**Development tool**
You can use your existing COBOL (or PL/1) development workbench to develop these types of applications.

**Advantages**
The advantage of this solution is that it is a pure language solution and does not require any additional middleware.

**Limitations**
The limitations include:
- XML parsing in Enterprise COBOL is event-driven (Simple API for XML (SAX) style), which is more difficult to handle than tree-style parsing.
- Namespace support for XML generation and parsing is only available with Enterprise COBOL V4.1 and later.
As with all asynchronous loosely coupled integration solutions, waiting time is consumed between sending a message and receiving the reply. In most cases, this wait time is not a problem in an online transaction processing (OLTP) one message at a time environment, but in a batch job, this solution is only possible in limited cases. Refer to “Solution strategy” on page 117 for considerations.

**Using a Java Web services client**

This solution is similar to the solution discussed in “Using Web services over JMS transport” on page 120, but in this case, you use the COBOL/Java interoperability features of Enterprise COBOL. You create a Web services client stub in Java and call that stub from COBOL, as depicted in Figure 5-8. The Java part of the application has the responsibility of communicating with the service provider using either SOAP over HTTP or SOAP over JMS. So, one of the advantages of this solution over the solution discussed in “Using Web services over JMS transport” on page 120 is that you can now also use HTTP as the transport protocol. In addition, Java provides much more functionality and flexibility in terms of using Web Services standards.

**Target architecture**

Figure 5-8 shows the target architecture of this solution.
Runtime software prerequisite
Enterprise COBOL is required on z/OS for XML parsing.

Development tools
You can use your existing COBOL development workbench to adapt the COBOL. In addition, you need a simple Java development tool to develop the Java classes for the Web Services calls.

Advantages
The advantages of this solution are:

► Almost seamless integration
► COBOL does not have to consider the SOAP message format
► Common code base if you also have Java-based clients

Limitations
Limitations are:

► JVM creation incurs a significant startup cost. Therefore, this solution might be unsuitable for “small” job steps.
► As in all mixed-language solutions, the build and deployment procedures might be more difficult.

Example
Refer to 15.3, “Calling a Java client from Enterprise COBOL” on page 471 for an example.

Using DB2 outbound Web Services support
DB2 outbound Web Services support is an extremely attractive solution, because it isolates your application code from the intricacies of the HTTP protocol. Taking it one step further, you can wrap the call to the SOAP-enablement user-defined functions (UDFs) into a UDF (or stored procedure) of your own, thus hiding the details of SOAP request building and SOAP response parsing from the application. Using this technique, calling a Web service from a DB2 application program can be made to look like a call to a local stored procedure or UDF.

Target architecture
Figure 5-9 on page 124 shows the target architecture.
**Runtime software prerequisite**
The only prerequisite of this solution is to have DB2 on z/OS with UDFs installed.

**Development tool**
There is no need for a specific development tool.

**Advantages**
The advantages of this solution are:

- Web Services calls are completely transparent to the application program.
- The response can be presented to the application program in familiar tabular format.
- Maintenance is centralized when using wrapper UDFs.

**Limitations**
Limitations of this solution are:

- As of DB2 Version 9 for z/OS, only HTTP and HTTPS are supported as transport protocols; there is no support for JMS transport.
You need a good understanding of DB2 XML support, XQuery syntax, and so forth.

Problem determination can be more difficult, because communication and XML parsing are handled by DB2 and are outside the control of the application program.

**Example**

We provide a sample for this solution in Chapter 14, “Using DB2 as a Web service requester” on page 455.
Samples

In this part of the book, we show you a number of samples of scenarios that were introduced in Part 1, “SOA application enablement patterns for z/OS” on page 25.

We discuss the following samples in detail:

- Exposing VSAM data as Web Services
- Exposing DB2 data as a service
- Exposing IMS data as a service
- Exposing data as a service by using IBM WebSphere Message Broker
- Exposing an IMS application as a Web service
- Building composite CICS services using the Service Flow Feature
- Using DB2 as a Web service requester
- Using an external Web service in batch applications
Exposing VSAM data as Web Services

In this chapter, we first develop a generic Web service that is written in Java to access existing VSAM data. We then deploy the Web service to WebSphere Application Server for z/OS and explore several ways to test and invoke it.

We then show building a less generic but more business-oriented Web service that exposes a structured view of the data but not the raw data. We use the JZOS Record Generator tool to create Java classes for conversion from and to COBOL record layouts.
6.1 Introduction

In this section, we first develop a simple Java bean to access a VSAM key-sequenced data set (KSDS) file. We use the JZOS library to perform the file operations (inserting, updating, deleting, and listing records). We then create a Web service that wraps those operations and then deploys the service to WebSphere on z/OS.

6.2 The JZOS toolkit and library

One of the weak points in the adoption of Java technology on the mainframe in the beginning was that access to z/OS-specific system services was not possible, or at least not without considerable effort. Most importantly, Java did not have access to the “traditional” MVS file system (specifically, sequential and partitioned access methods and VSAM).

With the advent of JZOS, things have changed. It originally was offered as a free download from Dovetailed technologies, which was later acquired by IBM. JZOS is now part of the IBM SDK for z/OS and is constantly being improved and functionally enhanced. The latest versions are published on the IBM developerWorks Web site before they are integrated into one of the next SDK maintenance levels.

JZOS actually consists of two parts:

- **JZOS launcher** is a utility program to invoke Java applications in MVS batch (that is, from JCL). Because the launcher is not relevant to this discussion, we do not go into more detail here. Refer to *JZOS Batch Launcher and Toolkit Installation and User’s Guide*, SA23-2245, for more information.

- **JZOS library** is a set of utility classes that provide access to traditional z/OS data and z/OS system services directly from Java applications.

In this section of the book, we use the **ZFile class**, which provides access to traditional MVS data, such as sequential and partitioned data sets, as well as VSAM files, extensively. In fact, the ZFile class is just a thin wrapper around the C/C++ standard library functions for file I/O, which can be used to access both traditional and z/OS UNIX® data.

First, we create a generic VSAM service that can be used to access any VSAM file regardless of its structure.
6.3 Creating the generic VSAM access bean

In this step, we create a simple Java bean that wraps part of the ZFile class, exposing only the ZFile methods that are relevant for VSAM-related operations:

1. In *Rational Developer for System z (RDz)* or IBM Rational Application Developer (RAD), create a new Project (select File → New → Project → Java Project). Enter VsamAccess as the project name.

2. First, we need to bring the JZOS library into the workspace and add it to the project's classpath. In the project, create a folder named lib (select File → New → Folder). Now, you need to copy ibmjzos.jar from the SDK installation directory on z/OS into that folder. You can use either FTP or the Uniform Symbol Specification file access tooling in RDz. Finally, add it to the project's classpath by right-clicking the project and selecting Properties → Java Build Path → Libraries → Add JARs. Navigate to lib/ibmjzos.jar and click OK.

**Tip:** The ibmjzos.jar file can be found in the lib/ext subdirectory of the SDK installation, for example, /usr/lpp/java/J5.0/lib/ext/ibmjzos.jar.

3. Next, create a new Java class as shown in Example 6-1. Again, this class just serves as a wrapper to hide the ZFile operations that are not relevant to (or not even usable for) VSAM file access.

*Example 6-1   Wrapper class for VSAM access*

```java
package com.ibm.itso.sg247669.vsam;

import java.io.IOException;
import com.ibm.jzos.ZFile;
import com.ibm.jzos.ZFileException;

public class VsamFile {

    private ZFile vsam;

    private byte[] buf;

    public VsamFile(String name) throws IOException {
        String mvsName = ZFile.getSlashSlashQuotedDSN(name, true);
        vsam = new ZFile(mvsName, "rb+,type=record");
        if (vsam.getVsamType() != ZFile.VSAM_TYPE_KSDS
            && vsam.getVsamType() != ZFile.VSAM_TYPE_KSDS_PATH) {
```
vsam.close();
   throw new IOException("Not a KSDS dataset: " + name);

   buf = new byte[getRecordLength()];
}

public int getRecordLength() throws IOException {
   try {
      return vsam.getLrecl();
   } catch (ZFileException e) {
      throw new IOException("Error obtaining record length: " + e.getMessage());
   }
}

public int getKeyLength() throws ZFileException {
   return vsam.getVsamKeyLength();
}

public int read(byte[] buf) throws ZFileException {
   return vsam.read(buf);
}

public byte[] read() throws ZFileException {
   if (read(this.buf) > 0) {
      return this.buf;
   } else {
      return null;
   }
}

public void close() throws ZFileException {
   vsam.close();
   vsam = null;
}

public void write(byte[] record) throws ZFileException {
   vsam.write(record);
}

public boolean locate(byte[] record, int keyOffset, int keyLength) 
   throws ZFileException {
   return vsam.locate(record, keyOffset, keyLength, ZFile.LOCATE_KEY_GE);
}
public boolean locate(byte[] key) throws ZFileException {
    return locate(key, 0, key.length);
}

public int update(byte[] record) throws ZFileException {
    return vsam.update(record);
}

public void deleteRecord() throws ZFileException {
    vsam.delrec();
}

dpublic boolean delete(byte[] record, int keyOffset, int keyLength)
    throws ZFileException {
    if (locate(record, keyOffset, keyLength)) {
        read();
        deleteRecord();
        return true;
    } else {
        return false;
    }
}

dpublic boolean delete(byte[] key) throws ZFileException {
    return delete(key, 0, key.length);
}
}

Notes on Example 6-1 on page 131 (these numbers correspond to the numbers on the right):

1. The underlying JZOS ZFile object used to access the data set is shown.
2. Convert the data set name to the naming conventions of the C/C++ standard I/O library. The Boolean argument instructs the ZFile class to not automatically prepend the TSO prefix if the name is not enclosed in single quotation marks, that is, the constructor expects a fully qualified dataset name.
4. Verify that the file is a VSAM KSDS.
5. Allocate a buffer for records.
6. Read the current record into a user-supplied buffer.
Read the current record into the pre-allocated buffer. Note that client code will have to make a copy of the buffer contents, because it will change when the next record is read.

Search for a record whose key is greater than or equal to the key that is passed as a parameter.

### 6.4 Creating the Web service implementation

In this step, we use the wrapper class that we developed in 6.3, “Creating the generic VSAM access bean” on page 131 to create a Web service. The service interface will offer the methods to create, read, update, and delete records.

Unlike the wrapper class just created, there will be no externally exposed methods to open or close a VSAM file, however. The reason is simple: Web services are stateless by nature. There is no way to pass a “handle” to an open file to the service consumer and let the service consumer then invoke additional operations that can pass that handle as a parameter.

Therefore, all operations follow a similar sequence of steps:

1. Open the VSAM file.
2. Perform the operation (such as insert, update, list, or delete).
3. Close the VSAM file.
4. Return the result (if applicable).

**Example 6-2 The VsamAccess service implementation**

```java
package com.ibm.itso.sg247669.vsam;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class VsamAccess {
  
  public void insert(String fileName, byte[] record) throws IOException {
    VsamFile file = new VsamFile(fileName);
    try {
      file.write(record);
    } finally {
      file.close();
    }
  }
}
```


public void update(String fileName, byte[] record) throws IOException {
    VsamFile file = new VsamFile(fileName);
    try {
        file.locate(record);
        file.read();
        file.update(record);
    } finally {
        file.close();
    }
}

public boolean delete(String fileName, byte[] key) throws IOException {
    VsamFile file = new VsamFile(fileName);
    try {
        return file.delete(key);
    } finally {
        file.close();
    }
}

public byte[][] list(String fileName, byte[] key, int maxRecords) throws IOException {
    VsamFile file = new VsamFile(fileName);
    try {
        List<byte[]> result = new ArrayList<byte[]>();
        if (file.locate(key)) {
            for (int count = 0; count < maxRecords || maxRecords == 0; count++) {  
                byte[] record = file.read();
                if (record == null) {
                    break;
                }
                result.add(record.clone());
            }
        }
    return result.toArray(new byte[result.size()][]);
    } finally {
        file.close();
    }
}
Notes on Example 6-2 on page 134 (These numbers correspond to the numbers on the right):

1. We have to make sure that the file is closed when the method ends, no matter what; otherwise, the file remains open to the operating system, and the next attempt to open it will fail. Therefore, the call to file.close() is in an end block to ensure that it gets called even when file.write() caused an exception.

2. Loop until the maximum number of records has been read, unless maxRecords is set to zero in which case all records are returned.

3. Exit the loop when the end-of-file has been reached.

4. We have to clone the byte array that was returned from file.read(), because it will be reused in the next read() call. Otherwise, we end up with all identical records in the list (namely, the last record read).

6.5 Creating and deploying the Web service

In this step, we turn the service implementation class into a Web service and deploy it to WebSphere Application Server on z/OS.

As a prerequisite for deployment, we have to make the WebSphere Application Server instance on z/OS known to RDz so that we can deploy to it:


2. In the Server’s host name field, type your WebSphere Application Server server’s host name. Select IBM → WebSphere v6.1 Server as the server type, and click Next (Figure 6-1 on page 137).
3. On the next window, enter either your server's Object Request Broker (ORB) bootstrap port or SOAP connector port. If security is enabled on the server, you must fill in the respective fields as well. Click Finish.

The server is now listed in the Servers view (Figure 6-2 on page 138).
4. Now, we are ready to create our VsamAccess Web service and deploy it to the application server. Select the VsamAccess class, right-click and click Web Services → Create Web service. The Web service wizard opens (Figure 6-3).

**Figure 6-3  The Web Service wizard**
5. We are going to create a **Bottom-up Java bean Web service** from our service implementation class, com.ibm.itso.sg247669.vsam.VsamAccess, so leave the first two entry fields as is. Move the top slider to **Start Service**.

6. Next, we have to adjust the four settings under the Configuration heading, by clicking the respective links. First, change the target server type by clicking **Server: WebSphere v6.0 server**. From the dialog box that opens, select **Choose server** first, **Existing servers**, and select the WebSphere on z/OS server that we made known to the workbench in the previous step.

7. Leave the Web Service Runtime configuration item unchanged.

8. The Web Service wizard is going to create a Web project and an Enterprise project for us. We want more descriptive names than the default names (WebServiceProject and WebServiceProjectEAR). So, click either the Service Project or the Service EAR project link, and enter **VsamAccessWebservice** as the service project name. Because we are going to reuse the EAR project later, we chose a more generic name for the EAR project, **SG24-7669** (Figure 6-4). Click **OK** to return to the Web service wizard.

![Figure 6-4 Service project settings](image)

9. Move the Client generation slider to **No client** and click **Next**. The Web service wizard now creates a Web project with all the necessary artifacts, such as the Web module deployment descriptor.

10. On the next window, Service Endpoint Interface Selection, leave the defaults as is and click **Next**. The next window, Web Service Java Bean Identity, allows you to change the generated Web Services Description Language (WSDL) port and file names and to select the methods that you want to expose. Again, leave the settings as is and click **Next**.
11. The wizard now generates the WSDL file to match the Java method signature. It also creates an EAR project and installs the EAR on the application server, which might take a while.

12. In the next and final panel, leave the defaults and click **Finish**.

If everything went well, the Servers view now indicates that the application has been installed and is running on the application server (Figure 6-5).

![Figure 6-5  VSAM Web service successfully created and deployed](image)

### 6.6 Testing the service

Now, we are ready to test the Web service. RDz and RAD come with a great tool for that purpose: The Web Services Explorer. To test the service:

1. Select **Run** → **Launch the Web Services Explorer** (or click the icon). From the icon bar in the top row of the view, click the **WSDL Page** icon (Figure 6-6 on page 141).
2. Next, we need to find the WSDL document for the VSAM Web service. Click the WSDL Main node in the Navigator pane. Next, click the **Browse** link in the Actions bar. The WSDL Browser dialog opens. Select **WebSphere Web Services WSDL documents** as the WSDL source category and **VsamAccessWebservice** as Workspace project (Figure 6-7). Alternatively, you can choose **Workspace WSDL documents**, because the WSDL file actually sits in our workspace.
3. Click **Go** to close the WSDL browser, and then, click **Go** again in the Actions pane. The Web Services Explorer now retrieves the WSDL document from the server, parses it, and displays a tree in the Navigator pane showing the services declared in that WSDL, their respective bindings, and the operations (Figure 6-8).

![Navigator pane showing the VsamAccess Web service](image)

**Figure 6-8  Navigator pane showing the VsamAccess Web service**

4. The list operation is probably the obvious one to test first, because it does not change any data. So, select **list** from the operations as shown in Figure 6-8. The Actions pane changes to reflect the currently selected operation, so you see three text entry fields corresponding to the three parameters on the list operation.

5. The first parameter is easy: Enter the name of an existing VSAM KSDS data set on your z/OS system. Be sure to enter a fully qualified data set name; the double slashes can be omitted (the `ZFile.getSlashSlashQuotedDSN` method takes care of that; refer to the comments about Example 6-1 on page 131).

**Note:** The VSAM file that we used in this example is the one that comes with the CICS Catalog sample application. Refer to “JCL for creating and loading the catalog VSAM file” on page 482 for instructions about how to create a copy of that sample file.

6. The second parameter, however, is a bit tricky. As you can see, the Java byte array type has been mapped to a Base64-encoded string (“base64Binary”) in the WSDL. So, to search for the key 0180 we have to find the Base64 representation of the bytes making up the string “0180”. No, wait a minute, we have to find the Base64 representation of those bytes in **EBCDIC encoding**.

An easy way is to use class `sun.misc.BASE64Encoder` (which is not an official part of the SDK, but for our purpose, that does not matter). Open a new Java Scrapbook page (**File** → **New** → **Other** → **Java Scrapbook Page**) and type
the following expression (we are using EBCDIC code page 1047, but any other EBCDIC variant will produce the same result):

```java
new sun.misc.BASE64Encoder().encode("0180".getBytes("Cp1047"))
```

Then, select the entire expression, right-click, and select Display. You get the following output in the scrapbook window:

```
(java.lang.String) 8PH48A==
```

Copy the expression result (8PH48A==), go back to the Web Service Explorer and paste it into the key entry field. Finally, enter any positive value or 0 for no limit as the third parameter, maxRecords.

7. Click Go.

8. If the output looks like Figure 6-9, you did it.

Just like the key, the return values are Base64-encoded strings in EBCDIC encoding. To convert these values into human-readable format, select and copy one of them and go back to the Java Scrapbook page.

![Figure 6-9  Output from invoking VsamAccessService.list](image)
9. In the scrapbook page, enter the following expression:

```java
new String(new sun.misc.BASE64Decoder().decodeBuffer(""), "Cp1047")
```

Paste the output string into the argument of `decodeBuffer`, highlight the entire expression again and display the result as you did before. If you selected the first row of the output, you get:

```
0180Highlighters Yellow 5pk                 010003.490088010
```

At this point, feel free to experiment a bit with the Web Services Explorer. Maybe, try the insert and delete operations. You have to use the Scrapbook page again to get the encoded values to paste into the record argument.

Now, admittedly, all of that Base64 encoding and decoding is rather clumsy, so in the next step, we create a simple Java client.

### 6.7 Creating a Java client for the VsamAccess service

To create a stand-alone Java client that can invoke our VsamAccess Web service, we use the Web service wizard again:

1. The first step is to create a new Java project for the Web service client. Select **Project → New → Java Project**. Enter VsamAccessThinClient as the project name, and click **Finish**.

2. Now, in the VsamAccessWebservice project, expand the **WebContent/WEB-INF/wsdl folder** and select the WSDL file, **VsamAccess.wsdl**. Right-click and select **Web Services → Generate Client**. The Web service wizard opens (Figure 6-10 on page 145). Move the slider to **Deploy client**.
3. Next, we have to tell the wizard that we want to generate the client code into the Java project that we just created. So, click the **Client project** link. In the dialog that opens, select the project name (**VsamAccessThinClient**) from the **Client project** list box (Figure 6-11). Because this project is a plain Java project (as opposed to a Dynamic Web project, for example), the Client project type and Client EAR project entry fields are disabled automatically.

**Figure 6-11 Specify client project settings for the Web service client project**
4. Click **OK** to get back to the Web services client, and click **Finish**. The wizard now generates Java client code and adds a library containing the Web services client run time to the project’s classpath (Figure 6-12).

![Package Explorer](image1.png)

**Figure 6-12  Generated client code**

5. Now, we are ready to create a Java program that looks up the service and invokes it. Select the project (**VsamAccessThinClient**) and create a new Java class by selecting **File → New → Class**. Enter the package name (**com.ibm.itso.sg247669.vsam**) and the class name (**ClientTestMain**). Paste the client program as shown in Example 6-3 on page 147.
Example 6-3  Simple Java client program to invoke the VSAM Web service

```java
package com.ibm.itso.sg247669.vsam;

public class ClientTestMain {

    public static void main(String[] args) {
        try {
            VsamAccess vsam = new VsamAccessServiceLocator().getVsamAccess();
            byte[][] result = vsam.list("ulrich.cics.exmpcat",
                    "0010".getBytes("Cp1047"),
                    0);
            for (int i = 0; i < result.length; i++) {
                System.out.println(new String(result[i], "Cp1047"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Notes on Example 6-3 (these numbers correspond to the numbers on the right):

1. The generated VsamAccessServiceLocator class “knows” about the service endpoints. We ask it for the default endpoint (the one that is specified in the WSDL).

2. We invoke the list operation.

3. We use the EBCDIC representation of 0010 as key.

4. We use 0 to indicate that we want to retrieve all records having the same or greater key.

5. We print each record by converting the bytes to a string, using EBCDIC encoding.

6. Now, try to run the client program (select the class, and then select Run → Run As → Java Application. We got the output as shown in Example 6-4.

Example 6-4  Error from invoking the Java client: No Secure Sockets Layer (SSL) configuration available

Sep 4, 2008 10:48:57 AM com.ibm.ws.ssl.config.SSLConfigManager
INFO: ssl.disable.url.hostname.verification.CWPKI0027I
WARNING: WSWS3734W: Warning: Exception caught from invocation to
com.ibm.ws.webservices.engine.transport.http.HTTPSender:
WebServicesFault
     faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.generalException
faultActor: null
faultDetail:

at
com.ibm.ws.webservices.engine.transport.security.ConfigSSLProvider.getConfig(ConfigSSLProvider.java:257)
at
com.ibm.ws.webservices.engine.transport.http.HTTPSender$2.run(HTTPSender.java:973)
[... stack trace omitted ...]

An easy way to get around this problem is to use the HTTP endpoint instead of the HTTPS endpoint. Modify the client program by changing the line:
VsamAccess vsam = new VsamAccessServiceLocator().getVsamAccess();
to:
VsamAccess vsam = new VsamAccessServiceLocator().getVsamAccess(httpEndpoint);

Obviously, you need to change the URL to match your host name and HTTP port.

7. Run the program again. This time, it successfully connects to the HTTP endpoint and produces the expected output, as shown in Example 6-5.

Example 6-5   Output from running the VSAM client against the HTTP endpoint

Sep 4, 2008 11:06:24 AM com.ibm.ws.ssl.config.SSLConfigManager
INFO: ssl.disable.url.hostnameverification.CWPKI0027I
0010Ball Pens Black 24pk 010002.900135000
0020Ball Pens Blue 24pk 010002.900006050
0030Ball Pens Red 24pk 010002.900106000
0040Ball Pens Green 24pk 010002.900080000
[... rest of output omitted ...]
8. Alternatively, you can configure SSL for the client, using the SSL configuration that comes with the RDz installation. Modify the launch configuration to invoke the program:

   a. Select **Run → Run** to open the Launch Configuration editor. Select the **Arguments** tab. In VM arguments (Figure 6-13), enter:

   ```
   -Dcom.ibm.SSL.ConfigURL="file:c:/Program Files/IBM/SDP70/runtimes/base_v61_stub/properties/ssl.client.props"
   ```

   Be sure to enclose the URL in double quotation marks if the path name contains blanks.

   ![Figure 6-13 Modifying the client launch configuration for SSL support](image)

   b. Now, change the program back to use the default endpoint and run again. This time, it successfully connects to the HTTPS endpoint.

   **Note:** For more information about SSL configuration, refer to the WebSphere Application Server product documentation (look for the ssl.client.props client configuration file section).
6.8 Creating a wrapper service

At this point, we have a complete Web service that can read and update any VSAM KSDS file on the system, and we have a Java client to invoke that service from anywhere. However, the service is still not easy to use. The client has to know about the exact layout of the records in the file, including the record data types and lengths. It becomes complicated when the data in the file is not encoded as strings but uses native COBOL types, such as packed decimal fields. If the layout of the records ever changes, all clients accessing the file using this service have to be changed accordingly. Therefore, the service is too generic. It exposes not the business view of the data, but a purely technical view.

Therefore, in this section, we add a new layer on top of the existing service implementation that hides the record layout from the clients and exposes the business meaning of the individual fields instead (Example 6-6).

Example 6-6  COBOL level 01 definition of the sample VSAM file

```
01 WS-CAT-ITEM.
   05 WS-ITEM-REF          PIC 9(4).
   05 WS-DESCRIPTION       PIC X(40).
   05 WS-DEPARTMENT        PIC 9(3).
   05 WS-COST              PIC ZZZ.99.
   05 WS-IN-STOCK          PIC 9(4).
   05 WS-ON-ORDER          PIC 9(3).
   05 FILLER               PIC X(20).
```

One of the numerous useful features in the JZOS toolkit is a complete set of field converter classes in the package `com.ibm.jzos.fields`. These field types operate on a byte array and can be used to convert from COBOL or Assembler structures to Java types, and from Java types to COBOL or Assembler structures.

**Note:** Of course, there is always more than one way to do it. In this example, we show you how to use the JZOS record framework and generator. An equally valid alternative is to use the CICS/IMS Java Binding tooling available in RDz.

Now, the field converter classes deal with converting from COBOL (or Assembler) types to Java types, for example, from packed decimal to integer, on an elementary field level. But this is only part of the job, because you still have to manually calculate field offsets and lengths within the COBOL data structures.
Given that COBOL structures can be quite complex, it is a tedious and error-prone job to do that.

The JZOS record generator does precisely that. It uses IBM Enterprise COBOL to compile the copybook and uses a special data set ("SYSADATA") produced by Enterprise COBOL to learn about the types, offsets, and lengths of the data items described by that copybook. It then generates Java classes that use the record converter utility classes of the JZOS run time (Example 6-7 on page 152). We illustrate this process in Figure 6-14.

![Figure 6-14 Generating record classes with the JZOS record generator](image)

As of the time of writing this book, the Record Class Generator is not an official part of the SDK for z/OS. The JZOS version in the SDK is at level 2.1.0, while the Record Class Generator is available in level 2.2.1 and higher. We therefore downloaded the latest AlphaWorks version of JZOS and copied the Jar file containing the Generator to a directory on the z/OS Unix file system.

**Note:** You will find the jar file, `jzos_regen.jar`, in the additional materials for this book. Refer to Appendix B, “Additional material” on page 485.

**Note:** It is not necessary (and not recommended) to replace the JZOS version in the SDK with the AlphaWorks version. All we need for the purposes of this section is the `jzos_regen.jar` file. The generated code will work fine with the older JZOS level that is part of the SDK.
Example 6-7  JCL to invoke the JZOS record class generator

```
//RECGEN JOB ,NOTIFY=&SYSUID,MSGLEVEL=(1,1)
// *
//COBADATA PROC CPYBKLIB=,CPYBOOK=
//COBOL EXEC IGYWC,PARM='LIB,ADATA'
//SYSIN DD DISP=SHR,DSN=ULRICH.JZOS.RECGEN(COBPREFIX)
//  DD DISP=SHR,DSN=&CPYBKLIB(&CPYBOOK)
//  DD DISP=SHR,DSN=ULRICH.JZOS.RECGEN(COBsuffix)
//SYSADATA DD DSN=&ADATA,DISP=(,PASS)
//  PEND
// *
//CATITEM EXEC COBADATA,
//  CPYBKLIB=ULRICH.CICSCAT.COPYLIB,
//  CPYBOOK=CATITEM
//JAVAGEN EXEC JVMPRC50,
//  *   LOGLVL='+T',
//  JAVAACL='com.ibm.jzos.recordgen.cobol.RecordClassGenerator'
//MAINARGS DD *
bufoffset=false
package=com.ibm.itso.sg247669.vsam.catalog
outputDir=/u/ulrich/cobgen
//SYSADATA DD DSN=&ADATA,DISP=(OLD,DELETE)
//STDENV   DD *
JAVA_HOME=/usr/lpp/java/J5.0
PATH="/bin:$JAVA_HOME/bin:
LIBPATH="/lib:/usr/lib":"$JAVA_HOME"/bin":"$JAVA_HOME"/bin/j9vm
export JZOS_JVM_OPTIONS=""
# Add the JZOS alphaWorks jars to the classpath
CLASSPATH="$CLASSPATH":"/u/ulrich/jzos_recgen.jar"
export JAVA_HOME PATH LIBPATH CLASSPATH
```

Notes on Example 6-7:

1. An inline JCL procedure invokes the COBOL compiler. The caller must pass the library and member name for the copybook. Of course, you are free to put this procedure into your own PROCLIB instead of using an inline procedure.

2. Invoke the COBOL compiler with the ADATA option.

3. The input to the compiler is our copybook, wrapped into a dummy prefix and suffix to make it a valid COBOL program. So, the prefix contains dummy IDENTIFICATION and DATA DIVISIONs, and the suffix contains a dummy PROCEDURE DIVISION.

4. The SYSADATA output will be generated into a temporary dataset and passed to the subsequent steps.
5 Call the COBADATA cataloged procedure to invoke the COBOL compiler. You can process more than one copybook by simply duplicating this step and modifying it accordingly. The record generator will produce classes for each structure that it finds in SYSADATA.

6 This step invokes the Record Generator tool, using the JZOS batch launcher. The Record Generator will create one Java class for each copybook. Note: If your installation does not have the JZOS batch launcher procedure (JVMRC50) in the standard PROCLIB, you will have to add a JCLLIB statement after the jobcard.

7 The MAINARGS DD statement can be used to supply command-line arguments to a Java application, in this case, to the Record Generator.

8 This line is an option to the Record Generator to suppress generation of buffer offsets. For the full set of options supported by the Record Generator, refer to the documentation.

9 This line tells the Record Generator into which package to generate the record classes.

10 This line is the output directory for the generated classes.

11 This line references the SYSADATA data set produced by the COBOL compiler.

12 Set up the environment for the Java Virtual Machine (refer to the JZOS documentation for detailed information). You probably want to externalize this information into a partitioned data set (PDS) member; we used an instream dataset here just to be self-contained.

Now, import the generated Java file into the VsamAccessWebservice project. You can use FTP, or if you have RDz, you can use the “Import from remote file system” feature (select File → Import → Other → Existing USS File Filter or File → Import → Other → Remote File System. You need to have a definition for the remote system in advance.)

Next, we create the implementation for the CatalogAccess Web service (Example 6-8 on page 154). As you can see, the code is almost identical to the original generic VsamAccess service, except that instead of byte arrays, it deals with CatalogItems.

Also, as a slight variation, the VSAM file name is no longer passed as a parameter to each method call, but it will be configured when the Web service is deployed.
Example 6-8 The CatalogAccess Web service implementation

```java
package com.ibm.itso.sg247669.vsam.catalog;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.itso.sg247669.vsam.VsamFile;

public class CatalogAccess {

    private String getFileName() throws NamingException {
        Context ctx = new InitialContext();
        return (String) ctx.lookup("java:comp/env/fileName");
    }

    public void insert(CatalogItem item) throws IOException, NamingException {
        VsamFile file = new VsamFile(getFileName());
        try {
            file.write(item.getByteBuffer());
        } finally {
            file.close();
        }
    }

    public void update(CatalogItem item) throws IOException, NamingException {
        [ ... implementation omitted ... ]
    }

    public boolean delete(int itemRef) throws IOException, NamingException {
        VsamFile file = new VsamFile(getFileName());
        try {
            CatalogItem item = new CatalogItem();
            item.setWsItemRef(itemRef);
            return file.delete(item.getByteBuffer(),
                               CatalogItem.WS_ITEM_REF.getOffset(),
                               CatalogItem.WS_ITEM_REF.getByteLength());
        } finally {
            file.close();
        }
    }
}
```
Notes on Example 6-8 on page 154:

1. We no longer expect the VSAM file name as a parameter on the service call. But rather than hard-coding it in the service implementation, we look it up in Java Naming and Directory Interface. We describe this step in more detail later.

2. To perform the actual insert operation, we have to convert the CatalogItem instance that was passed in as a parameter to a byte array. The generated code in the CatalogItem class handles all the conversion details for us.

3. The parameter to the `delete` method is just the item number (as opposed to a complete CatalogItem object).

4. We need to create a “dummy” CatalogItem object and set the item number on that object.

5. Now, we call the `VsamFile.delete()` method, using the dummy CatalogObject converted to a byte array. We use the three-argument version of `VsamFile.delete`, passing the offset and the length of the key field (WS_ITEM_REF). Each field in the original COBOL record has a corresponding static member field in the generated Java class that describes the field, including its offset from the beginning of the buffer, and its length.

Next, we use the Web service wizard again to create a Web service for the CatalogAccess implementation. This time, we tell the Web service wizard to also create a Java client for invoking the service from a stand-alone application.

The procedure is familiar by now: Select the service implementation (the CatalogAccess class), and select **Web services ➔ Create Web Service**. Set the options as shown in Figure 6-15 on page 156. Note that we decided to also create the client proxy at this point into the VsamAccessThinClient project.
After the wizard has finished, we need to modify the deployment descriptor of the Web module to create an *environment entry* that will hold the VSAM file name.

Open the deployment descriptor of **VsamAccessWebservice** (the *web.xml* file in the *WebContent/WEB-INF* folder). Switch to the **Environment** tab, find the Environment variables section and click **Add**. The Add Environment Entry dialog opens (Figure 6-16 on page 157). Enter the name of the environment entry, and select **String** as the variable type. We can add a default value in the Value entry field; in this example, we left it empty so we will have to specify a value after deployment. Click **Finish**.
Chapter 6. Exposing VSAM data as Web Services

Figure 6-16  Add an environment entry to the Web deployment descriptor

The new environment entry now shows up in the deployment descriptor, as shown in Figure 6-17.

Figure 6-17  New environment entry

Save the deployment descriptor and wait until the application has successfully been published to WebSphere.

Now, we have to specify a value for the newly created environment entry. Open the WebSphere Application Server Integrated Console (select the server entry in the Servers view, right-click and select Run administrative console). Select Applications → Enterprise Applications and click SG24-7669. In the Web Module Properties section, click Environment entries for web modules. In the panel that appears, enter the file name of the VSAM file to be accessed by the CatalogAccess service (Figure 6-18 on page 158). Click OK and then click Save to apply the changes.
Now, you can test the new Web service using the Web Services Explorer, or you can create a stand-alone Java client as we just did. Figure 6-19 shows a partial response from invoking the list operation via the Web Services Explorer.

![Figure 6-19 Response from CatalogAccess.list in Web Services Explorer](image)
Chapter 7. Exposing DB2 data as a service

In this chapter, we describe the steps that are involved in exposing DB2 data as a service.
7.1 Overview

While DB2 can act as a Web service requester (as we explain in Chapter 14, “Using DB2 as a Web service requester” on page 455), DB2 made a conscious decision to reuse the Web Services infrastructure that works well that is provided with products, such as WebSphere Application Server, WebSphere Message Broker, WebSphere DataPower, and so on, and focused on the Web Services to DB2 mapping instead of reimplementing everything from scratch.

DB2 development made a conscious decision to not start over, but to leverage the existing Web services infrastructure provided by application servers.

There are several alternatives for how that Web service actually accesses the database. They differ in the amount of coding necessary and in the tooling required:

- Using plain Java DataBase Connectivity (JDBC) or SQLJ
- Using `pureQuery`
- Using Entity EJBs (with either Container-Managed (CMP) or Bean-Managed persistence (BMP))
- Using Data Web Services (DWS)

**Note:** In this section, we focus on exposing DB2 data (in the form of queries) as a service, as opposed to DB2 business logic (in the form of stored procedures). As mentioned elsewhere in this book, this section might be a good starting point, but it is somewhat questionable from an architectural point of view.

In an ideal world (at least from a DB2 point of view), your business logic is already neatly encapsulated behind stored procedures; in fact, the main design point of DWS is to expose stored procedures.

7.2 Using Data Web Services (DWS)

Probably the easiest way to expose DB2 data and business logic as a Web service is using the Data Web Services (DWS) feature of IBM Data Studio Developer.

In this section, we use the following tools:

- IBM Data Studio Developer, Version 1.2 for development
- WebSphere Application Server as the Web service hosting environment
7.2.1 Data Web Services

Data Web Services (DWS) is a new solution to significantly ease the development, deployment, and management of Web services-based access to DB2. DWS lets you take Data Manipulation Language (DML) statements (such as SELECT, INSERT, UPDATE, DELETE, XQuery, and, last but not least, stored procedure calls) and generate Web services from these statements without writing a single line of code. The generated Web services are packaged as a ready-to-deploy Web application (as a .war file), which can easily be deployed to supported application servers.

Highlights of DWS include:

- Creating Web services using DWS requires no programming.
  DWS lets you create Web services using a drag-and-drop interface.
- DWS supports SOAP over HTTP and Web Services Description Language (WSDL) generation.
  DWS automatically generates a WSDL file that contains a description of the Web service.
- No code generation.
  DWS consists of a common metadata-driven run time, and there is no “black box” code that gets generated “under the covers,” which results in a reliable and lightweight application.

7.2.2 Introduction to the example

In this example, we show how to create a simple service that allows clients to query and update one of the sample tables that are part of the DB2 installation.
If the DB2 sample tables are not installed on your system, or if you are not authorized to access them, refer to “DDL to create the DEPT sample table” on page 483.

**Note:** In this example, we show how you can use Data Studio by building a simple query on a DB2 table and generating a Web service from that query. Technically, this approach is good, but from an architectural point of view, it is more likely that you will choose to Web service-enable stored procedures.

### 7.2.3 Setting up the connection to DB2

If you have not already configured your connection to DB2:

1. Open the Data Perspective *(Window → Open Perspective → Data)*.
2. In the Database Explorer view, right-click **Connections** and select **New Connection**.
3. Depending on the DB2 version that you want to use for this sample, expand either DB2 for Linux, Unix, or Windows or DB2 for z/OS. Select **All Versions**.
4. Enter the details for the connection, depending on the type of database. For DB2 for z/OS, you need to enter the location name, the host name of the DB2 server, and the distributed data facility (DDF) port number.

**Tip:** An easy way to find out the values for your installation of DB2 for z/OS is to search for messages DSNL519I and DSNL004I in the DB2 MSTR address space or to issue the -DIS DDF DB2 command.

Additionally, you have to configure the location of the two Jar files that contain the JCC driver and the driver license file. If the Jar files are not already on your workstation, you can download them (via FTP, for example) from the DB2 installation directory on the host (typically, `/usr/lpp/db2/db2/db2910/jdbc/classes`).

Because we are going to access the DB2 sample tables, we need to remove the check from “Retrieve objects created by this user only.”

5. In the bottom left part of the dialog, enter the user name and password. Click **Test Connection** to verify. If the connection test was successful, click **Next**.
6. Because a DB2 for z/OS subsystem typically contains a large number of objects, we want to restrict the list to only those objects in which we are interested. So, on the next panel, we can specify a filter to only show objects in the DSN8910 schema. Click **Selection**, and in the list of available schema names, check **DSN8910**. Click **Finish**.
The new connection now shows up in the Database Explorer view. Expand the tree up to the Schemas level; you will see the DSN8910 schema.

**Important:** IBM Data Studio relies on the use of a set of DB2 stored procedures to extract information from the DB2 catalog. If these procedures are not installed, you will get an SQLException when trying to establish the connection. For details and instructions, refer to:  

### 7.2.4 Creating the SQL statement

Now, we are ready to build an SQL statement. First, we need to create a new Data Development project. In the Data Project Explorer Tab, right-click and select **New → Data Development Project**. Enter **Employee** as the project name. Leave the other settings as is, and click **Next**. On the next window, select **Use an existing connection** and select the connection that you created in the previous step.

Next, we are going to build the SQL statement. In the Data Project Explorer, expand the newly created project and select **SQL Scripts**. Right-click and select **New → SQL Statement**. Enter `getDepartment` in the Statement name field. Depending on your preferences, you can choose to write the SQL statement manually (select **Edit using SQL editor**) or interactively (select **Edit using SQL builder**). The SQL builder is straightforward to use, and although the SQL that we are going to write is quite trivial, we encourage you at this point to try to use it.

Either way, the SQL eventually looks like Example 7-1.

#### Example 7-1  SQL for department lookup by department number

```
SELECT DEPTNAME, MGRNO
FROM DSN8910.DEPT
WHERE DEPTNO = :DEPTNO
```

Now is a good time to test the statement. In the SQL editor pane, right-click and select **Run SQL**. Because the SQL contains a host variable, the tool now prompts you to specify a value for that host variable (Figure 7-1 on page 164). Enter E21 or any other value that appears in the DEPTNO column of the table, and click **Finish**. You see the result of running the query in the Data Output view.
7.2.5 Creating a Web service from the query

At this point, we are ready to create a new Web service for the query. The new Web service will have one operation, which is the query that we just created, with one parameter, which is the variable part of the statement (the :DEPTNO host variable). The steps are:

1. Right-click the Web Services folder in the Data Project Explorer, and select New Web Service. In the dialog that opens, enter EmployeeWebservice as the Web service name and http://sg247669.itso.ibm.com/employee/ as the Namespace URI (Figure 7-2 on page 165). Click Next.

Note: The namespace URI serves to avoid name clashes between XML element names. Although it looks like something that can be accessed via HTTP, it is not (necessarily) the case; it is just meant to be a string that is unique.
2. Now, right-click the SQL statement, \texttt{getDepartment.sql}, and select \textbf{Add to Web Service}. Select the Web service and click the top button (>) to move it from the \textit{Available Web services} to the \textit{Selected Web services list box} (Figure 7-3). Click \textbf{Next}.

3. Because the query will return at most one result, we check \textbf{Fetch only single row for queries} on the next window (Figure 7-4 on page 166). This setting has an impact on the generated WSDL: It will be much simpler because in the response element, we will have just a complex type consisting of two simple
types rather than an unbounded sequence of that complex type. The simpler WSDL in turn affects language binding generation (refer to 12.4, “Generating the CICS artifacts” on page 395).

![Figure 7-4 Fetch only single row for queries](image)

4. The next step is to build the Web service. Right-click **EmployeeWebservice** (the asterisk indicates that the Web service was modified since the last deployment operation) and select **Build and Deploy**. In the following dialog, specify the options to use as shown in Figure 7-5 on page 167. Be sure to select **WebSphere Application Server v6.x** as the Web Server type. Click **Finish**.
5. In the Data Project Explorer view, expand the XML folder and its WSDL subfolder. You now see that a WSDL file for our new Web service has been created.

**Note:** We did not generate bindings for the REST binding, because that does not really add any useful functionality for this scenario (it adds support for HTTP GET invocation, thus allowing browser-based access).

We do not explain the SOAP over JMS method in this book, but if you want to experiment with it, be aware of a minor bug in the DWS tooling (you will see compilation errors in the generated code due to missing libraries on the compiler classpath). Refer to 7.4, “Troubleshooting” on page 175 for the work-around.
7.2.6 Installing the Web service in WebSphere Application Server

Go back to either the Java 2 Platform, Enterprise Edition (J2EE) or the Java perspective. You will see that our new Web service was automatically made available as a dynamic Web project named EmployeeEmployeeWebserviceWeb.

To install the Web service in WebSphere Application Server, simply drag the Web project onto the deployment descriptor of the EAR project (Figure 7-6).

![Figure 7-6 Adding the Employee Web service project to the EAR project](image)

There is one final thing to do before the application can successfully run on WebSphere. Maybe you noticed the yellow warning sign next to the EmployeeEmployeeWebserviceWeb project. Looking at the Problems view, we see the message:

CHKJ2877W: Missing JNDI name for reference jdbc/EmployeeWebservice

This message indicates that DWS is going to access the database using a datasource with the JDBC name jdbc/EmployeeWebservice. We have to map that resource reference to an existing datasource configured in WebSphere either after deployment (via the WebSphere Application Server Integrated Console), or you can specify a default binding in the IBM WebSphere extension deployment descriptor. To do the latter, we have to enable that functionality first. Right-click the Web project and select Properties. Highlight Project Facets and click Add/Remove Project Facets. Check WebSphere Web (Co-existence) and make sure that 6.1 is selected in the list box. Click Finish, and then click OK to close the properties dialog.
Now, open the deployment descriptor of the Web project and change to the References pane. Highlight the resource reference, and enter the Java Naming and Directory Interface (JNDI) name of an existing datasource in the WebSphere Bindings section (Figure 7-7).

**Note:** Without adding the WebSphere project facet, the WebSphere Bindings section is missing from the References pane.

![Figure 7-7 Configuring a default WebSphere binding for the datasource reference](image)

The application is now ready for installation on WebSphere. Right-click the application in the Servers view, and click **Publish**.

### 7.2.7 Testing the Web service

To test the Web service, go back to the Workbench. From the menu bar, select **Run → Launch the Web Services Explorer**. The Web Service Explorer window will now open with an icon bar and three panes: Navigator, Actions, and Status. In the icon bar, click the **WSDL page** icon. In the WSDL URI field, enter the following URI:
http://host.name:port/EmployeeEmployeeWebserviceWeb/services/EmployeeWebservice?WSDL

The *host.name* is the host name of your WebSphere Application Server server, and *port* is the HTTP Transport port for the default_host virtual host. Click Go. The Web Services Explorer now retrieves the WSDL document from WebSphere Application Server, parses it, and updates the Navigator pane to show the services that are declared in that WSDL, and their binding and operation details. Expand the tree in the Navigator pane and click the **getDepartment** operation. The Action pane now allows you to specify the parameters for the getDepartment operation. Enter A00 and click Go. You see output similar to Figure 7-8.

![Web Services Explorer](image)

**Figure 7-8** Invoking the Web service via the Web Services Explorer

To see the actual SOAP response XML document, click **Source** in the Status pane. You will see both the SOAP request and SOAP response documents (Examples Example 7-2 on page 171 and Example 7-3 on page 171).

**Note:** For better readability, the example documents have been slightly reformatted.
**Example 7-2  SOAP request document**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope xmlns:q0="http://sg247669.itso.ibm.com/employee/
 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:getDepartment>
      <DEPTNO>A00</DEPTNO>
    </q0:getDepartment>
  </soapenv:Body>
</soapenv:Envelope>
```

Notes on Example 7-2:

1. The XML namespace declaration is for the application-specific parts of the SOAP response. The namespace URI, `http://sg247669.itso.ibm.com/employee/`, is the one that we assigned earlier when we created the Web service. Note that the Web service explorer uses a randomly generated prefix, `q0` in this case, to refer to that namespace.

2. This name is the SOAP body element. This next part is the application-specific part of the request. Note that unprefixed elements (DEPTNO) inherit the namespace from their immediate parent, in this case, from getDepartment.

**Example 7-3  SOAP response document**

```xml
<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:getDepartmentResponse xmlns:ns1="http://sg247669.itso.ibm.com/employee/
      xmlns=""
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <DEPTNAME>SPIFFY COMPUTER SERVICE DIV.</DEPTNAME>
      <MGRNO>000010</MGRNO>
    </ns1:getDepartmentResponse>
  </soapenv:Body>
</soapenv:Envelope>
```
Notes on Example 7-3 on page 171:

1. Just as in the SOAP request, we have namespace declaration for the application-specific part of the response. The SOAP engine generated a random namespace prefix to refer to our application namespace, in this case, ns1.

2. This part is the application-specific part of the response.

**Note:** We occasionally received the following SQL error when trying to invoke the service after a few minutes of idle time:

```
SQL State = null SQL Code = -4499 SQL Message = [ibm][db2][jcc][t4][2030][11211] A communication error has been detected. Communication protocol being used: TCP/IP.

Communication API being used: SOCKETS. Location where the error was detected: T4Agent.sendRequest().

Communication function detecting the error: OutputStream.flush(). Protocol specific error codes Software caused connection abort: socket write error, *, 0. Message: Software caused connection abort: socket write error Exception message = com.ibm.db2.jcc.c.DisconnectException:

[ibm][db2][jcc][t4][2030][11211] A communication error has been detected. Communication protocol being used: TCP/IP.
```

The next attempt after this error was then successful again.

We found that the reason for this problem was an unrecommended ZPARM setting in our DB2 installation: the CMTSTAT parameter was set to ACTIVE. Refer to 7.4, “Troubleshooting” on page 175 for details.

### 7.2.8 Additional activities

At this point, feel free to explore the DWS tooling by creating additional operations. For example, you can create an SQL statement to update department or employee data, and add that SQL statement to the existing Web service as a new operation.

### 7.3 Exposing a DB2 stored procedure as a service

In this section, we show how to expose a DB2 stored procedure as a Web service. Again, we use DWS and Data Studio. As you will see, this task is just as easy as exposing SQL statements; in fact, the steps are very much the same.
7.3.1 Creating the stored procedure

The stored procedure that we use in this example calculates an employee’s bonus, subject to a base bonus, a rating (1 = excellent to 4 = poor), and the number of years that the employee has been with the company.

In our Employee project in RDz/Data Studio, right-click Stored Procedures and select New → Stored Procedure. Because we are not going to build the procedure interactively but will simply paste the SQL, you can type any value in the Name field. From the Language drop-down list, select SQL-Native. Click Finish. In the editor window that opens, replace the text with the code in Example 7-4, and then, save the file.

Example 7-4  Sample stored procedure for bonus calculation

CREATE PROCEDURE CALCULATE_BONUS
    ( IN EMP_NO     CHAR(6)
      , IN BASE_BONUS DECIMAL(9, 2)
      , IN RATING     SMALLINT
      , OUT EMP_BONUS DECIMAL(9, 2)
    )
LANGUAGE SQL
MODIFIES SQL DATA
QUALIFIER DSN8910
WLM ENVIRONMENT FOR DEBUG MODE DB9HWLM5
BEGIN
    DECLARE YEARS_WITH_COMPANY SMALLINT;

    -- Calculate bonus depending on rating
    CASE RATING
        WHEN 1 THEN -- High performer
            SET EMP_BONUS = BASE_BONUS * 1.30;
        WHEN 2 THEN -- Average
            SET EMP_BONUS = BASE_BONUS;
        WHEN 3 THEN -- Below Average
            SET EMP_BONUS = BASE_BONUS * 0.80;
        WHEN 4 THEN -- Loser
            SET EMP_BONUS = 0;
        ELSE
            SIGNAL SQLSTATE 'EMP99'
                SET MESSAGE_TEXT = 'Invalid rating';
    END CASE;

    -- Extra bonus for every 10 years with the company
    SELECT YEAR(CURRENT DATE) - YEAR(HIREDATE)
INTO YEARS_WITH_COMPANY
FROM EMP
WHERE EMPNO = EMP_NO;
IF MOD(YEARS_WITH_COMPANY, 10) = 0 THEN
  SET EMP_BONUS = EMP_BONUS + 1000;
END IF;

-- Record bonus in database
UPDATE EMP SET BONUS = EMP_BONUS
  WHERE EMP.EMPNO = EMP_NO;
END

The new procedure shows up in the Stored Procedures folder. Select it, right-click and select Deploy (Figure 7-9). Data Studio will install the procedure in DB2, and we are ready for the next step.

Figure 7-9   Deploying the stored procedure to DB2

7.3.2 Creating a Web service for the stored procedure

Now, creating a Web service that exposes the stored procedure is extremely easy. Highlight the procedure in the Stored Procedures folder, select Add to Web Service and go through the wizard that opens. Or simply expand the existing Web Services folder, and drag and drop the procedure to our EmployeeWebservice as we show in Figure 7-10 on page 175.
For consistency, rename the operation to `calculateBonus` by highlighting it and selecting **Rename**. Finally, to install the updated service, follow the steps described in 7.2.6, “Installing the Web service in WebSphere Application Server” on page 168.

**7.4 Troubleshooting**

This section describes work-arounds for problems that we ran into when developing the samples in this chapter.

**7.4.1 Errors in generated code for SOAP over JMS binding**

If you tried to generate code for SOAP over JMS binding in 7.2.5, “Creating a Web service from the query” on page 164, you might have received compilation errors in the generated code (Figure 7-11).

![Compilation errors when generating the SOAP over JMS binding](image)

Figure 7-11   Compilation errors when generating the SOAP over JMS binding
This issue is due to a minor bug in the DWS tooling, which can easily be handled. We need to manually add the WebSphere libraries to the compile classpath of both the Enterprise Java Bean (EJB™) and the JMS project:

1. Open the property settings for the EJB project, select **Java Build Path** and click **Add Library**. In the dialog that opens, select **Server Runtime** and click **Next**. Select **WebSphere Application Server v6.1** and click **Finish**. Finally, click **OK** to close the properties dialog.

2. Perform the same steps for the generated JMS project.

3. Finally, rebuild the two projects. Select **Project → Clean**, check both projects, and click **OK** (Figure 7-12). The compilation errors are now gone.

![Figure 7-12   Rebuilding the projects after correcting the classpath.](image)

### 7.4.2 Stale connection when DWS uses a Type 4 connection

As mentioned in 7.2.6, “Installing the Web service in WebSphere Application Server” on page 168, we saw SQL errors indicating a stale connection when trying to invoke a DWS-generated Web service after idle time and when the application was bound to a data source using a Type 4 JDBC connection.

We found that the reason for this kind of error was an unrecommended ZPARM setting: The CMTSTAT parameter was set to ACTIVE in our DB2 system, which disables thread pooling in the DDF address space. With this setting, DDF will cancel each thread that has been idle for at least IDTHTOIN (“Idle Thread Timeout”) seconds, unless IDTHOIN is set to zero which, you guessed it, is also not recommended.
Refer to *Distributed Functions of DB2 for z/OS and OS/390*, SG24-6952 for an in-depth discussion of these parameters.

### 7.5 Sample material

Refer to Appendix B, “Additional material” on page 485 for instructions on obtaining the workspace with the sample code.

You may also have to refer to Appendix A, “Defining data used in samples” on page 481 for information on defining the DB2 DEPT table.
Exposing IMS data as a service

In this chapter, we describe the steps that are involved in exposing IMS data as a service.
8.1 Introduction

In this example, we show how IMS databases can be accessed directly from a Java 2 Platform, Enterprise Edition (J2EE) environment in WebSphere Application Server for z/OS, hence, giving you the option to create a Web service. IMS Transaction Manager and IMS Connect are not used in this scenario.

You can write Java applications that run in WebSphere Application Server for z/OS and access IMS databases directly when WebSphere Application Server for z/OS and IMS are on the same logical partition (LPAR).

To deploy an application to WebSphere Application Server for z/OS, you must install the IMS DB Resource Adapter (the Java class libraries for IMS) in WebSphere Application Server for z/OS and configure both IMS Open Database Access (ODBA) and the Database Resource Adapter (DRA).

Java Database Connectivity (JDBC) or the IMS hierarchical database interface for Java calls is passed to the IMS DB Resource Adapter, which converts the calls to DL/I calls. The IMS DB Resource Adapter passes these calls to Open Database Access (ODBA), which uses the database resource adapter (DRA) to access the DL/I region in IMS.

8.2 Steps

To use the IMS DB Resource Adapter with WebSphere Application Server for z/OS, you will follow these steps (which we will describe in detail):

1. IMS setup
   To use the IMS DB Resource Adapter to access IMS DB from WebSphere Application Server for z/OS, you first must configure WebSphere Application Server for z/OS to access IMS databases using ODBA. ODBA uses the Database Resource Adapter (DRA) to access IMS databases.

2. WebSphere Application Server for z/OS setup
   This section assumes that you are familiar with WebSphere Application Server Version 6 for z/OS and its WebSphere Application Server Integrated Console:
   - WebSphere Application Server for z/OS Servant Region STEPLIB
   - Installing the IMS DB Resource Adapter
   - Installing the custom service
3. Running the IMS Installing Verification Program (IVP) for the WebSphere Application Server for z/OS Java environment

This topic provides the high-level tasks required to run the IMS IVP applications for the WebSphere Application Server for z/OS Java environment:

- Installing the data source for the IMS IVP application
- Installing the IMS IVP application
- Testing the IMS IVP application

4. Generating IMS database metadata by using the IMS DLIModel utility

In order for a Java application to access an IMS database, it needs information about that database. The DLIModel utility allows you to transform your IMS database information (program specification blocks, database descriptions, and COBOL copybooks) into application independent metadata:

- Creating a DLIModel Project
- Importing a COBOL copybook

5. Running your application on WebSphere Application Server for z/OS:

- Installing the data source for your application
- Deploying your application to WebSphere Application Server for z/OS

### 8.3 IMS Setup

To use the IMS DB Resource Adapter to access the IMS DB from WebSphere Application Server for z/OS, you must first configure WebSphere Application Server for z/OS to access IMS databases using ODBA. ODBA uses the Database Resource Adapter (DRA) to access IMS databases.

To configure WebSphere Application Server for z/OS to access IMS databases, perform the following tasks:

1. If not already done, create a DRA startup table. The DRA startup table module name must have the following naming convention:
   - Bytes 1-3: “DFS”
   - Bytes 4-7: 1-byte to 4-byte ID

   **Recommendation:** The 1-byte to 4-byte ID must be the IMS system ID.

   - Byte 8: “0”

   Example 8-1 on page 182 shows an example of the DRA startup table.
Example 8-1  The DRA startup table (example)

DFSIMSA0 CSECT
*
  DFSRPRP DSECT=NO, X
  CNBA=20, :total FP NBA buffers  X
  DBCTLID=IMSA, :name of DB or DB/DC region  X
  DDNAME=ODBADD, :ddname with dynalloc IMS DB lib X
  DSNAME=IMS10A.SDFSRESL, :dataset name of IMS DB lib  X
  FPBOF=3, :the number of FP OBA per thread  X
  FPBUF=5, :the number of FP NBA per thread  X
  FUNCLV=1, :the DRA level  X
  IDRETRY=0, :the number of retry IDENTIFY  X
  MAXTHRD=10, :the maximam number of DRA thrd  X
  MINTHRD=10, :the minimum number of DRA thrd  X
  SOD=T, :SNAP DUMP sysout class  X
  TIMER=60 :the time btw attempts of DRA
*
END

2. If not already done, link the DRA startup table into a load library. The skeleton JCL that you need to perform this task is shown in Example 8-2.

Example 8-2  The DRA startup table assemble JCL (sample)

//ITSOAM1A JOB (999,POK),'ITSOAM1',NOTIFY=&SYSUID,
//   CLASS=A,MSGCLASS=H,TIME=1439,
//   REGION=5000K,MSGLEVEL=(1,1)
//ASMLK PROC MBR=TEMPNAME,SOUT=H,RGN=512K,SYS2=
//C EXEC PGM=ASMA90,REGION=&RGN,PARM='OBJECT,NODECK'
//SYSLIB DD DSN=IMS10A.&SYS2.SDFSRESL,DISP=SHR
//       DD DSN=SYS1.MACLIB,DISP=SHR
//SYSLIN DD UNIT=SYSDA,DISP=(,PASS),
//       SPACE=(80,(100,100),RLSE),
//       DCB=(BLKSIZE=80,RECFM=F,LRECL=80)
//SYSPRINT DD SYSOUT=&SOUT,DCB=BLKSIZE=1089,
//       SPACE=(121,(300,300),RLSE,,ROUND)
//SYSUT1 DD UNIT=SYSDA,DISP=(,DELETE),
//       SPACE=(CYL,(10,5))
//L EXEC PGM=IEWL,PARM='XREF,LIST',COND=(0,LT,C),REGION=512K
//SYSLIN DD DSN=*.C.SYSLIN,DISP=(OLD,DELETE)
//SYSPRINT DD SYSOUT=&SOUT,DCB=BLKSIZE=1089,
//       SPACE=(121,(90,90),RLSE)
//SYSLMOD DD DSN=IMS10A.&SYS2.SDFSRESL(&MBR),DISP=SHR
//SYSUT1 DD UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),
//       SPACE=(1024,(100,10),RLSE),DISP=(,DELETE)
8.4 WebSphere Application Server for z/OS setup

This section assumes that you are familiar with WebSphere Application Server Version 6 for z/OS and its WebSphere Application Server Integrated Console.

A prerequisite step before continuing is installing the Java class libraries for IMS. Refer to the IMS Version 10 Installation Guide, GC18-9710-00, for more information.

8.4.1 Updating WebSphere Application Server for z/OS Servant Region STEPLIB

Update the JCL for WebSphere Application Server for z/OS by adding the following data sets to the STEPLIB:

- The load library that contains the DRA startup table.
- The SDFSRESL data set. This data set contains the ODBA runtime code.
- The SDFSJLIB data set. This data set contains the DFSCLIB member.

8.4.2 Installing the IMS DB Resource Adapter

After you configure WebSphere Application Server for z/OS to have access to IMS databases, you must install the IMS DB resource adapter in WebSphere Application Server for z/OS.

To install the IMS DB Resource Adapter, perform the following tasks:

1. Install the Resource Archive (RAR):
   a. From the WebSphere Application Server for z/OS WebSphere Application Server Integrated Console, click Resources, expand Resource
Adapters, and then click **Resource Adapters**. A list of resource adapters is displayed.

b. Click **Install RAR**. A dialog for installing the resource adapter is displayed.

c. Select **Server path** and type the path to the imsDBJCA.rar file:

\[\text{pathprefix}/usr/lpp/ims/imsjava10/imsDBJCA.rar\]

d. Click **Next**. A configuration dialog is displayed.

2. Configure the resource adapter by typing the following information:

- **Name**: A name for the resource adapter
- **Classpath**: The Java library for IMS consists of the following JAR files. The path and file names are:
  
  - \[\text{pathprefix}/usr/lpp/ims/imsjava10/imsJDBC.jar\]
  - \[\text{pathprefix}/usr/lpp/ims/imsjava10/imsDBJCA.jar\]
  - \[\text{pathprefix}/usr/lpp/ims/imsjava10/imsjavaBase.jar\]
  - \[\text{pathprefix}/usr/lpp/ims/imsjava10/imsXQuery.jar\]

Refer to Figure 8-1 on page 185 for a snapshot of the WebSphere Application Server Integrated Console panel used for configuring the IMS DB Resource Adapter.
**Figure 8-1** Configuring the IMS DB Resource Adapter

**Important:**

- The path and file names use both upper case and lower case characters.
- The `imsXQuery.jar` file is required only if you intend to use XQuery.
**Important:** In Chapter 8 on Page 120 of the *IMS Version 10 Communications and Connections Guide*, SC18-9703, the name of one of the jar files is spelled incorrectly:

- Correct: `pathprefix/usr/lpp/ims/imsjava10/imsjavaBase.jar`
- Incorrect: `pathprefix/usr/lpp/ims/imsjava10/imsjavabase.jar`

In Figure 8-1 on page 185, click **OK**. The IMS DB resource adapter is listed.

In the messages box, click **Save** to update the master repository with your changes, as shown in Figure 8-2.

![Messages](image)

8.4.3 Installing the custom service

When WebSphere Application Server for z/OS is started, the custom service initializes the ODBA environment. When the server is stopped, the custom service terminates the ODBA environment. After a server is started, every application that is running in the server uses the initialized ODBA environment.

To install the custom service, perform the following steps:

1. Modify the WebSphere Application Server for z/OS server.policy file. The WebSphere Application Server for z/OS server.policy file is in the properties directory (for example, `installdir/profiles/default/properties`). Add the code as shown in Example 8-3.

**Example 8-3  Adding code to server.policy file**

```plaintext
grant codeBase "file:/pathprefix/usr/lpp/ims/imsjava10/-" {  
    //Allows the IMS DB resource adapter and the custom service to read and write environment properties.  
    permission java.util.PropertyPermission "+", "read, write";  
    //Allows the IMS DB resource adapter and the custom service to use //the Java TDLI load library during runtime. permission java.lang.RuntimePermission "loadLibrary.JavTDLI";  
    };
```
The WebSphere Application Server for z/OS server.policy file is stored in the ASCII ISO8859 character set. To edit the server.policy file under OMVS:

a. Change the directory to properties directory:
   
   ```bash
cd installdir/profiles/default/properties
   ```

b. Convert from ASCII to EBCDIC:
   
   ```bash
   iconv -f ISO8859-1 -t IBM-1047 server.policy > servernew.policy
   ```

c. Use the Uniform Symbol Specification `oedit` command to add code to the end of file (refer to Example 8-3 on page 186).

d. Rename the existing server.policy file:
   
   ```bash
   mv server.policy file to server.policy.bu
   ```

e. Convert from EBCDIC to ASCII:
   
   ```bash
   iconv -f IBM-1047 -t ISO8859-1 servernew.policy >server.policy
   ```

f. Set the permissions of the new server.policy file:
   
   ```bash
   chmod 775 server.policy
   ```

2. Install the custom service:

a. In the left frame of the WebSphere Application Server for z/OS Integrated Console, click **Servers**, and then click **Application servers**. A list of application servers is displayed.

b. Click the name of the server on which you want to deploy your custom service.

c. Under **Server Infrastructure**, click **Administration**, and then click **Custom Services**, as shown in Figure 8-3 on page 188. A list of custom services is displayed.
d. Click **New**. A configuration dialog is displayed.

e. Select the **Enable service at server startup** check box. If you do not select this check box, the custom service is not invoked when you start the server.

f. Type the following information:
   - Classname: com.ibm.connector2.ims.db.IMSjdbcCustomService
• Display Name: A name for the custom service
• Classpath: The directory name containing the jar with the IMS JDBC classes, imsjava.jar, and the symbolic link libJavTDLI.so, which points to the data set, IMS.SDFSJLIB, in the WebSphere Application Server for z/OS Servant region STEPLIB. For example:

   pathprefix/usr/lpp/ims/imsjava10

g. Click OK. The custom service is listed (Figure 8-4 on page 190).
h. In the messages box, click Save to update the master repository with your changes.
i. Restart the server in order for the custom service to take effect.

   Important: In the OMS Version 10 IMS Communications and Connections Guide, SC18-9703, there is an error in chapter 8 on page 121. The directory name is correct, but the path and file name are incorrect.
8.5 Running the IMS IVP for the WebSphere Application Server for z/OS Java environment

Run the IMS IVP applications for the WebSphere Application Server for z/OS Java environment in WebSphere Application Server for z/OS to ensure that you have configured IMS and WebSphere Application Server for z/OS properly. This topic provides the high-level tasks that are required to run the IMS IVP applications for the WebSphere Application Server for z/OS Java environment.
8.5.1 Installing the data source for the IMS IVP application

The `DataSource` facility is a factory for connections to a physical data source, or database. A data source is registered with a naming service based on the Java Naming and Directory (JNDI) application programming interface (API). `DataSource` objects have properties that pertain to the actual data source that an application needs to access.

**Note:** You must use the `DataSource` facility, which replaces the `DriverManager` facility, because the `DriverManager` facility is not supported by the J2EE Connection Architecture Specification.

To install the data source for the IMS IVP application, perform the following tasks:

1. In the left frame of the WebSphere Application Server for z/OS Integrated Console, click **Resources**, expand **Resource Adapters**, and then click **Resource Adapters**. A list of resource adapters is displayed.

2. Click the name of IMS DB resource adapter that you chose when you installed the adapter. A configuration dialog is displayed.

3. Under Additional Properties, click **J2C connection factories**.

4. Click **New**. A configuration dialog is displayed.

5. Type the following information:
   - **Name**: Name for the data source
   - **JNDI Name**: `imsjavaIVP`

6. Click **OK**.

   Refer to Figure 8-5 on page 192. The data source is listed in the J2C Connection Factories.

---

**Prerequisites:** Ensure that the base IMS IVPs have been run. The IMS IVPs prepare the DBD for the IVP database, named `IVPDDB2`, and load the IVP database. They also prepare the Java application program specification block (PSB) (named `DFSIVP37`), build application control blocks (ACBs), and prepare other IMS control blocks that are required by the IMS IVP applications for the WebSphere Application Server for z/OS Java environment. For details about how to run the base IMS IVPs, refer to the *IMS V10 Installation Guide*, GC18-9710.
7. Click the name of the data source that you installed.

8. Under **Additional Properties**, click **Custom properties**. Three properties are listed in a table.

9. Click **DatabaseViewName**. A configuration dialog is displayed.
10. In the Value field, type: `samples.ivp.DFSIVP37DatabaseView`

11. Click **OK**.

Refer to Figure 8-6. The properties table displays the DatabaseViewName value that you just entered.

![Figure 8-6 Configure database view name](image)

12. Click **DRAName**. A configuration dialog is displayed.

13. In the Value field, type bytes 4 - 7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, refer to “IMS Setup” on page 181.

14. Click **OK**.

Refer to Figure 8-7 on page 194. The properties table displays the DRA name that you just entered.
Figure 8-7  Configure DRAName

15. In the messages box, click **Save** to update the master repository with your changes.

16. Restart the server in order for the custom service to take effect.

### 8.5.2 Installing the IMS IVP application

This section describes how to deploy the IMS IVP application to WebSphere Application Server for z/OS.

To install the IMS IVP application, perform the following tasks from the WebSphere Application Server for z/OS Integrated Console:

1. Click **Applications**, and then click **Install New Application**. A dialog for installing the application is displayed.
2. Select **Remote file system** and type the path to IMSJavaIVP.ear:

\[
\text{pathprefix}/usr/lpp/ims/imsjava10/samples/ivp/was/imsjavaIVP.ear
\]

3. Click **Next**.

Refer to Figure 8-8.

![Preparing for the application installation](image)

**Figure 8-8  Installing the IMS IVP application (panel one of five)**

4. In the next window, accept the defaults and click **Next**. The **Install New Application wizard** is started. Refer to Figure 8-9 on page 196 for the first panel. Click **Next**.

**Note:** Depending on your specific server configuration, you might have to use the wizard to change certain default values.
5. In the panel titled “Step 2: Map modules to servers,” as shown in Figure 8-10 on page 197, accept the defaults, and click **Next**.
In the “Step 3: Summary panel,” as shown in Figure 8-11 on page 198, click **Finish**. A message is displayed that indicates first that the application is being installed, and then, that the installation was successful.
Figure 8-11 Installing the IMS IVP application (panel four of five)

7. Click **Save** to save to the Master Configuration to update the master repository with your changes, as shown in Figure 8-12.
8.5.3 Testing the IMS IVP application

This topic describes how to test the IMS IVP application in WebSphere Application Server z/OS.

To test the IMS IVP application, perform the following steps using the WebSphere Application Server for z/OS Integrated Console:

1. Click Applications, and then, click Enterprise Applications. The application IMSJava IVP is listed with a red X, which indicates that the application is stopped.

2. Select IMSJava IVP.

3. Click Start.

Refer to Figure 8-13. The Application Status of application IMSJava IVP will turn into a green arrow, which indicates that the application is started.

<table>
<thead>
<tr>
<th>Select</th>
<th>Name</th>
<th>Application Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CustomerServiceGetCustomer_var</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DefaultApplication</td>
<td></td>
</tr>
<tr>
<td>✔</td>
<td>IMSJava IVP</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8-13   Start the IMS IVP application

4. Open a Web browser.

5. Type the Web address:
   http://host_IP_address:port/IMSJavaIVPWeb/IMSJavaIVP.html
   An input Web page opens.

6. Click Run the IVP.

Refer to Figure 8-14 on page 200.
Enabling z/OS Applications for SOA

Figure 8-14 IMS Java Installation Verification Program (IVP)

If WebSphere Application Server for z/OS is configured properly, the IVP shows “The IVP was SUCCESSFUL” and displays the results of checks performed by the IVP. Figure 8-15 on page 201 shows the output in a case where the IVP is successful.

If WebSphere Application Server for z/OS is not configured properly, the IVP shows “The IVP was NOT SUCCESSFUL” and displays the results of checks performed by the IVP.
**IMS Java Installation Verification Program (IVP)**

The IVP was SUCCESSFUL

**Result**

<table>
<thead>
<tr>
<th>Classpath:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>imssqlJavaBase.jar</td>
<td>OK L2007042001</td>
</tr>
<tr>
<td>imssqlJDBC.jar</td>
<td>OK IMS JDBC driver located</td>
</tr>
<tr>
<td>imssqljavaTM.jar</td>
<td>Not Tested Skipped imssqljavaTM.jar c</td>
</tr>
<tr>
<td>imssqlDBJCA.jar</td>
<td>OK IMS DB JCA driver located</td>
</tr>
<tr>
<td>Custom service</td>
<td>OK Custom service ran successfully</td>
</tr>
<tr>
<td>xml-apis.jar</td>
<td>OK IBM JAXP 1.3.8</td>
</tr>
<tr>
<td>xercesImpl.jar</td>
<td>OK XML4J 4.4.12</td>
</tr>
<tr>
<td>xalan.jar</td>
<td>OK XSLT4J Java 2.7.12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Connection:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Connection</td>
<td>OK</td>
</tr>
<tr>
<td>Database Calls</td>
<td>OK</td>
</tr>
<tr>
<td>Close Connection</td>
<td>OK</td>
</tr>
</tbody>
</table>

IVP PASSED

**Figure 8-15** The IVP was successful

---

### 8.6 Generating IMS database metadata using DLIModel utility

In order for a Java application to access an IMS database, it needs information about that database. The **DLIModel utility** allows you to transform your IMS database information (program specification blocks, database descriptions, and COBOL copybooks) into application independent metadata.
Two versions of the IMS DLIModel utility are available:

- An IMS-shipped version that runs in the UNIX System Services and can also be run using the z/OS BPXBATCH utility.
- A Web-download version that runs as a plug-in to Eclipse, WebSphere Developer for zSeries, Rational Application Developer (RAD), or Rational Developer for System z (RDz).

**Recommendation:** IMS Version 10 is the last release of IMS to support the IMS-shipped version of the DLIModel utility. Clients using this version need to migrate to using the DLIModel utility plug-in (Eclipse plug-in).

The plug-in version of the DLIModel utility allows you to generate and modify metadata in an Eclipse application development environment. We recommend this version of the utility for Java application developers using WebSphere Developer for zSeries, Rational Application Developer, and Rational Developer for System z. You can run this utility using your IMS source files as input, eliminating the need for a hand-coded control statement. The plug-in version of the utility also produces a graphical Uniform Modeling Language (UML) model file that makes viewing and modifying metadata easier.

For more information and to download the DL/I Model Utility, go to the following Web site:

http://www-01.ibm.com/software/data/ims/toolkit/dlimodelutility/

### 8.6.1 Creating a DLIModel Project

You can create a new DLIModel project using the DLIModel wizard.

Before you begin, review the requirements and restrictions for using the DLIModel utility.

To create a new DLIModel project:

1. Launch Eclipse or Rational Developer for System z.
2. Click **File → New → Project**, select **DLIModel Utility Project** from the list.
3. Click **Next** to launch the DLIModel wizard.

   Alternately, you can right-click the Package Explorer view, and then, click **New → Project**. If you are unable to locate the Package Explorer view, make sure that you are using the IMS perspective. The Package Explorer view defaults to the left side of the IMS perspective.
4. On the DLIModel Utility Project page, type a project name, type a Java package name, and specify a location in which to save your project:

- Your project name is strictly an internal name used by Eclipse to reference your specific project; this name will not be reflected in the generated metadata or in your application.
- The Java package name indicates into which Java package the generated metadata will be built.
- The Java package name field is optional and can be left blank; however, we recommend that you follow a logical naming scheme based on the company and application type for which the metadata will be used (for example, \texttt{com.mycompany.ims.payroll}).
- The project location is the location on your local hard drive to which your generated files will be saved. By default, the DLIModel wizard saves your project to your Eclipse workspace.

5. \textbf{Optional}: To customize the output options that will be generated into your project, click Show Advanced and check the box associated with the output type that you want to generate. Output options include:

- An XML schema
- A DLIModel database report
- A trace log

6. \textbf{Optional}: If you have an existing IMS control statement that you want to reuse as input to the utility, select Run utility from an IMS control statement. For example, you can reuse the control statements that you wrote to run the version of the DLIModel utility that is shipped with IMS.

Refer to Figure 8-16 on page 204.
7. On the Import IMS PSB and DBD Source page, browse existing projects or your local file system for the folder containing your PSB and DBD source files, then select a PSB and all of the DBD source files referred to by the selected PSB. Refer to Figure 8-17 on page 205.

**Important:**

- It is necessary to download the IMS PSB and DBD source to your local file system. You cannot use the remote file system on z/OS.
- The DBD file name in your local file system must be the same name as the DBD name.
- The Database PSB file names in your local file system must be the same name as the Database PSB names.
8. Generate the DLIModel project files by clicking **Finish**. The following files are created:

- Metadata file (.java)
- Graphical model file (.mdl)
- XML Metadata Interchange™ file (.xmi)
9. After the output files are generated, a Parsing IMS source files completed window opens to confirm that your DLIModel project was created. Click **OK** to close this window.

![Parsing IMS source files completed window](image)

The IMS perspective launches automatically and the generated .java and .mdl files open in the DLIModel editor. You can either exit the DLIModel editor and use the metadata for Java application development, or you can use the DLIModel editor to modify field, segment, or program communication block (PCB) information.

### 8.6.2 Importing a COBOL copybook

You can enhance metadata with additional information about field layouts by importing one or more COBOL copybooks into your DLIModel project.

You can only import a COBOL copybook if you have RAD, RDz, or WebSphere Developer for zSeries.

To import a COBOL copybook, perform the following steps:

1. Right-click the model background or the segment to which you want to add the COBOL field definitions.
2. Select **Import COBOL Fields**, as shown in Figure 8-19 on page 207.
3. Browse your local file system for your COBOL copybook files.
   A COBOL copybook can use any of the following file extensions: .cbl, .ccp, .cpy, or .cob.

4. Click **Select** to select a segment into which to import the COBOL copybook (Figure 8-20 on page 208).

5. Click **Finish** to integrate the COBOL fields into the selected segment or browse the local file system to import additional COBOL copybook files into a different segment.

   **Optional**: If you get an error after clicking **Finish**, verify that your COBOL preferences are set to support data structures only:

   a. Click **Window → Preferences**, and then select **COBOL** from the list on the left of the window.

   b. Select the **More COBOL options** tab and change the file extension support to **Data structures only**.

   c. Click **Apply** to update the changes, and then, reimport your COBOL copybook.
Example 8-4 shows a Java representation of an imported COBOL Copybook.

**Example 8-4  DLIDatabaseView subclass (sample)**

```java
package com.ibm.itso.dlidemo;

import com.ibm.ims.db.*;
import com.ibm.ims.base. *;

public class DFSIVP37DatabaseView extends DLIDatabaseView {

    // This class describes the data view of PSB: DFSIVP37
    // PSB DFSIVP37 has database PCBs with 8-char PCBNAME or label:
    //       PHONEAP

    // The following describes Segment: A1111111 ("A1111111") in PCB:
    // PHONEAP ("PHONEAP")
    static DLITypeInfo[] PHONEAP1111111Array = {
```
new DLITypeInfo("A1111111", DLITypeInfo.CHAR, 1, 10, "A1111111", DLITypeInfo.UNIQUE_KEY),
new DLITypeInfo("IOLINE", DLITypeInfo.CHAR, 1, 37),
new DLITypeInfo("IODATA", DLITypeInfo.CHAR, 1, 37),
new DLITypeInfo("IOLASTNAME", DLITypeInfo.CHAR, 1, 10),
new DLITypeInfo("IOFIRSTNAME", DLITypeInfo.CHAR, 11, 10),
new DLITypeInfo("IOEXTENSION", DLITypeInfo.CHAR, 21, 10),
new DLITypeInfo("IOZIPCODE", DLITypeInfo.CHAR, 31, 7),
new DLITypeInfo("IOFILLER", DLITypeInfo.CHAR, 38, 3)
};
static DLISegment PHONEAPA1111111Segment= new DLISegment
("A1111111","A1111111",PHONEAPA1111111Array,40);

// An array of DLISegmentInfo objects follows to describe the view
// for PCB: PHONEAP ("PHONEAP")
static DLITypeInfo[] PHONEAParray = {
    new DLITypeInfo(PHONEAPA1111111Segment,DLIDatabaseView.ROOT)
};

// Constructor
public DFSIVP37DatabaseView() {
    super("2.0","DFSIVP37", "PHONEAP", "PHONEAP", PHONEAParray);
} // end DFSIVP37DatabaseView constructor

} // end DFSIVP37DatabaseView class definition

**Note:** We had a problem with the original field names in the COBOL Copybook, because they contained minus signs. The SQL parser did not accept these names (and does not seem to support quotation marks). So, we manually changed the field names in the generated class, removing the minus signs.

### 8.7 Running an application in WebSphere Application Server for z/OS

This topic provides the high-level steps that are required to run an application that accesses IMS DB from WebSphere Application Server for z/OS.
8.7.1 Installing the data source for your application

The DataSource facility is a factory for connections to a physical data source, or database. A data source is registered with a naming service based on the Java Naming and Directory (JNDI) API. DataSource objects have properties that pertain to the actual data source that an application needs to access.

**Requirement:** You must use the DataSource facility, which replaces the DriverManager facility, because the DriverManager facility is not supported by the J2EE Connection Architecture Specification.

To install the data source for your application:

1. In the left frame of the WebSphere Application Server for z/OS Integrated Console, click **Resources**, expand **Resource Adapters**, and then, click **Resource Adapters**. A list of resource adapters is displayed.

2. Click the name of the IMS DB resource adapter that you chose when you installed the adapter. A configuration dialog is displayed.

3. Under Additional Properties, click **J2C connection factories**.

4. Click **New**. A configuration dialog is displayed.

5. Type the following information:
   - Name: The name for the data source
   - JNDI Name: The JNDI name for the data source

6. Click **OK**.

   Refer to Figure 8-21 on page 211. The data source is now listed in the J2C connection factories.
7. Click the name of the data source that you installed.

8. Under **Additional Properties**, click **Custom properties**. Three properties are listed in a table.

9. Click **DatabaseViewName**. A configuration dialog is displayed.
10. **Optional:** In the Value field, type the fully qualified DLIDatabaseView subclass name. If you set the subclass name, you must either create a data source for every PSB that an application accesses, or you must override the DLIDatabaseView subclass name in the DataSource object that is within the application by calling the setDatabaseView method and providing the fully qualified name of the subclass. If you do not set the subclass name, you need to create a data source only for each IMS. In the application, define the DLIDatabaseView subclass name in the DataSource object by calling the setDatabaseView method and providing the fully qualified name of the subclass.

11. Click **OK**. The properties table displays the DatabaseViewName value that you just entered (Figure 8-22).

12. Click **DRAName**. A configuration dialog is displayed.
13. In the Value field, type bytes 4 - 7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, refer to 8.3, “IMS Setup” on page 181.

14. Click OK (Figure 8-23). The properties table displays the DRA name that you just entered.

15. In the messages box, click Save to update the master repository with your changes.

16. Restart the server in order for the custom service to take effect.

Figure 8-23  Configure Database DRAName
8.7.2 Developing an application for WebSphere Application Server

You can develop an application program using a development tool, such as Rational Developer for System z (RDz).

The code in Example 8-5 and Example 8-6 is a sample that accesses the IMS Phone book database via the IMS DB Resource Adapter.

**Important:** The database name is the database PCB name, not the IMS database descriptor (DBD) name using SQL.

**Example 8-5  SQL sample**

```java
PreparedStatement stmt = conn.prepareStatement(
    "SELECT IOLASTNAME"
    + " , IOFIRSTNAME"
    + " , IOEXTENSION"
    + " , IOZIPCODE"
    + " FROM PHONEAP.A1111111"
    + " WHERE A1111111 = ?");
```

**Example 8-6  Program code sample**

```java
package com.ibm.itso.sg247669.ims.web;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;
import javax.transaction.HeuristicMixedException;
import javax.transaction.HeuristicRollbackException;
import javax.transaction.NotSupportedException;
```

Important: The database name is the database PCB name, not the IMS database descriptor (DBD) name using SQL.
import javax.transaction.RollbackException;
import javax.transaction.SystemException;
import javax.transaction.UserTransaction;

/**
 * Servlet implementation class for Servlet: HelloIMSServlet
 */
public class HelloIMSServlet extends javax.servlet.http.HttpServlet
implements
    javax.servlet.Servlet {
    /*
     * (non-Java-doc)
     *
     * @see javax.servlet.http.HttpServlet#HttpServlet()
     */
    public HelloIMSServlet() {
        super();
    }
    
    /*
     * (non-Java-doc)
     *
     * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest
     *      request,
     *      HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException,
                      IOException {
        PrintWriter pw = response.getWriter();
        try {
            Context ctx = new InitialContext();
            UserTransaction trn = (UserTransaction) ctx
                .lookup("java:comp/UserTransaction");
            trn.begin();
            DataSource dataSource = (DataSource) ctx
                .lookup("java:comp/env/jdbc/IMS");
            Connection conn = dataSource.getConnection();
            PreparedStatement stmt = conn.prepareStatement("SELECT "
                + " IOLASTNAME"
                + ", IOFIRSTNAME"
                + ", IOEXTENSION"
                + ", IOZIPCODE"
You can test the Java application after it has been deployed to WebSphere Application Server. Figure 8-24 shows the test output.

```java
String lastname = request.getParameter("lastname");
stmt.setString(1, lastname);
ResultSet rs = stmt.executeQuery();
ResultSetMetaData metaData = rs.getMetaData();
response.setContentType("text/plain");
if (rs.next()) {
    for (int idx = 1; idx <= 4; idx++) {
        pw.println(metaData.getColumnName(idx) + " : " +
               rs.getString(idx));
    }
} else {
    pw.println("No matching segment found");
}
trn.rollback();
conn.close();
} catch (Exception e) {
    e.printStackTrace(pw);
}
```

Figure 8-24 Testing the sample program
Exposing data as a service by using IBM WebSphere Message Broker

In this chapter, we show an example to access VSAM and DB2 data directly from WebSphere Message Broker. The service request is sent from the consumer to WebSphere Message Broker. In our example, we retrieve customer information based on a customer number. If the customer number does not exist, an error is produced and passed back to the consumer.

This example is based on the scenario that is described in 3.1, “Accessing VSAM data as a service” on page 28 and “Accessing DB2 on z/OS using WebSphere Message Broker” on page 45.
9.1 Introduction

This example demonstrates how a WebSphere Message Broker message flow can support two consumers. One consumer is an XML over MQ format/protocol, and the second consumer is SOAP over HTTP. WebSphere Message Broker performs the protocol conversion, data transformation, and content-based routing.

You must have a COBOL copybook that describes the format of the input and output messages for the VSAM file.

We developed this example using WebSphere Message Broker Toolkit Version 6.1.0.3 and WebSphere Message Broker Version 6.1 with Fix Pack 3 running on z/OS.

9.2 Steps

We guide you through these steps to access data in VSAM and DB2 using WebSphere Message Broker:

1. Invoke the WebSphere Message Broker Toolkit and create a project.
2. Build a WebSphere Message Broker Message Set for the VSAM record definition. A WebSphere Message Broker Message Set is needed to define the structure of the VSAM file.
3. Build a WebSphere Message Broker message set for the request/reply.
4. Build the Message Flow. The message flow defines the protocol conversion, data transformation, VSAM access, DB2 access, and content-based routing.
5. Describe and modify the node properties. Most nodes need one or several properties defined. The compute nodes need customized extended SQL (ESQL).
6. Deploy the flow to z/OS.
7. Test the flow on z/OS.

9.2.1 Invoking WebSphere Message Broker Toolkit to create a project

The WebSphere Message Broker Toolkit is the graphical “buildtime” environment that is used to develop Broker Message Flows, as well as the artifacts that are needed by flows, for example, the Broker Message Sets. The Toolkit runs in the Linux or Windows environment. After building the flow, it must be deployed to a runtime environment for execution, which we describe later in this chapter.
Start the WebSphere Message Broker Toolkit from the Windows Start menu and enter the workspace name that you want to use, as shown in Figure 9-1.

![Workspace Launcher](image)

**Figure 9-1  Select a workspace in WebSphere Message Broker Toolkit**

The Broker Toolkit includes several perspectives. A *perspective* is a pre-customized set of window panes, organized in a particular way to facilitate the development. For this part of the sample, you must 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, switch to the correct perspective by selecting **Window → Open Perspective → Other → Broker Application Development (default)**. Refer to Figure 9-2.

Alternatively, you can also select the recently used perspectives from the Toolkit tab buttons displayed in the upper right part of the Toolkit window.

![Broker Application](image)

**Figure 9-2  Select Broker Application Development perspective**

The finished flow that is developed in this exercise appears as shown in Figure 9-3 on page 220.
9.2.2 Building a Message Set for the VSAM record definition

In the following steps, we show how to create the Message Set for the VSAM file:

1. Select File → **New** → **Message Set**, as shown in Figure 9-4 on page 221.
Figure 9-4  Create a new Message Set

The “Create a new message set” wizard appears now.

2. Fill in the fields as shown in Figure 9-5 on page 222. You have to enter a 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 must be unique in a Broker. For this example, we use the name itsoxx_zsoa_CopyVSAM_CustRec. Click Next.
The Message set 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.

3. Select the message set data type by dropping down the list of data types and selecting **Binary data**. Clear the check marks in any other boxes and click **Next**. Refer to Figure 9-6 on page 223.
Figure 9-6   Select physical message format

The “Create a new message set” window is shown next. Click **Finish**. Refer to Figure 9-7 on page 224.
The Message set is now generated, and a pop-up Tip window appears as shown in Figure 9-8. Click **OK**.

You now have a window open in your workspace with the just defined message set, as shown in Figure 9-9 on page 225.
In the next step, the message set is defined from a COBOL copybook.

4. Highlight `messageSet.set` and right-click it. Select **New → Message Definition File From → COBOL File**, as shown in Figure 9-10 on page 226.

**Note:** Notice the automatically generated message set ID. You might need it later in the definition of the nodes.

**Tip:** Be careful when noting a message set ID. Zeros (0) and the character “O” in particular can be confused. Better yet, you can later copy the Message Set ID to the clipboard and then paste it into the required field.
Figure 9-10  Creating a new Message Definition File from COBOL

The COBOL copybook to import is shown in Example 9-1.

Example 9-1  VSAMCREC COBOL copybook

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).
5. In the next window, select the COBOL copybook file that represents the VSAM record. In our example, the name of this copybook file is VSAMCREC.cpy.

If the COBOL copybook file has previously been imported into the workspace, select **Select file from workspace**, as shown in Figure 9-11. If not, select **Select file from outside workspace**. The VSAM copybook is already part of the workspace included with the additional material.

Leave all the other field defaults in this window, and click **Next**.

*Figure 9-11 Selecting the COBOL copybook file from the workspace*
The next window is the Structure and message selection window.

6. Highlight **CUSTREC** and click > to move the structure to the Imported structures column. Refer to Figure 9-12.

![Figure 9-12 Selecting a structure from a COBOL copybook](image)

7. Next, check the **CUSTREC** in Imported structures, and click **Next**, as shown in Figure 9-13 on page 229.
8. The next window is Import options (Figure 9-14 on page 230). In this window, you specify the physical properties of the message definition.

Check the following two boxes:
- Create default values from INITIAL VALUEs
- Create facets from level 88 VALUE clauses where possible

Leave the other fields as shown in Figure 9-14 on page 230 (defaults), and click **Finish**.
If you expand the message definition, it looks like the window that is shown in Figure 9-15 on page 231.
9. This is a good time to save the message set by holding down Ctrl+s.

9.2.3 Building a message set for the request/reply

In the following steps, we show how to build a message set for the request/reply. The process is similar to the process that is explained in 9.2.2, “Building a Message Set for the VSAM record definition” on page 220. This time, however, we work with SOAP, and we build our message set from an XML schema. The steps are:

1. Select File → New → Message Set.

2. In the “Create a new message set” window, type a name for the message set. We used itsoXX_zsoa_xsd_CustReqReply in our example. The message set project name is filled by the wizard. Click Next. Refer to Figure 9-16 on page 232.
3. Select the message set data type by dropping down the list of data types and selecting **XML documents**. Check **Web services SOAP**, and clear the check marks from all other boxes. Click **Next**. Refer to Figure 9-17 on page 233.
4. In the Summary information window, click **Finish**. If the Tip pop-up window appears, click **OK**.

At this point, you have two message sets defined: one for the VSAM record and one for the SOAP request/reply message.

The message set will look like the window that is shown in Figure 9-18 on page 234.
The next step is to create a message definition file from the supplied XML schemas, one for the request message and one for the reply message.

5. Highlight `itsoXX_zsoa_xsd_CustReqReply` and right-click it. Select **New → Message Definition File From → XML Schema File**, as shown in Figure 9-19 on page 235.

**Tip:** Be careful when noting a message set ID. Zeros (0) and the character “O” in particular can be confused. Better yet, you can later copy the Message Set ID to the clipboard and then paste it into the required field.
Figure 9-19  Create a new Message Definition File from an XML Schema file

The XML schema file to import is shown in Example 9-2 on page 236. It is the one for the request message.
Example 9-2  CustomerInfoRequest XML Schema file

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="www.ibm.com" xmlns:q0="www.ibm.com">
<xs:element name="CustomerReq">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CustNo" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

6. In the next window, select the XML schema file. If the XML file has previously been imported into the workspace, select Select file from workspace, as shown in Figure 9-20 on page 237. If not, select Select file from outside workspace.

   The XML schema file for the request message is already part of the workspace that is packaged in the additional material for this book. Refer to Appendix B, “Additional material” on page 485.

7. Navigate to the XML file to import, select CustomerInfoRequest.xsd, and click Next.
8. In the next window, check **Elements → CustomerReq**, as shown in Figure 9-21 on page 238, and click **Finish**.
We now follow the same process for the reply message.


The CustomerInfoReply XML Schema file is shown in Example 9-3 on page 239.
Example 9-3  CustomerInfoReply XML Schema file

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="www.ibm.com">
    <xs:element name = "CustomerReply">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="CustNo" type="xs:int"/>
                <xs:element name="LastName" type="xs:string" minOccurs="0"/>
                <xs:element name="FirstName" type="xs:string" minOccurs="0"/>
                <xs:element name="Address1" type="xs:string" minOccurs="0"/>
                <xs:element name="City" type="xs:string" minOccurs="0"/>
                <xs:element name="State" type="xs:string" minOccurs="0"/>
                <xs:element name="Country" type="xs:string" minOccurs="0"/>
                <xs:element name="RetCode" type="xs:int"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

At this point, you have created one new message set and two new message definition files based on XML Schema files: one message definition file for the request message and one message definition file for the reply message.

10. Save everything by pressing Ctrl+s.

In the next steps, we generate the Web Services Description Language (WSDL).

11. Highlight itsoXX_zsoa_xsd_CustReqReply, right-click it, and select Generate → WSDL Definition, as shown in Figure 9-22 on page 240.
12. In the next window, because we already have existing message definitions in the workspace, select **Generate a new WSDL definition from existing message definitions** and click **Next**, as shown in Figure 9-23 on page 241.
Figure 9-23  Generating WSDL

13. In the next window, select the workspace directory, *itsoXX_zsoa_xsd_CustReqReply*, as shown in Figure 9-24 on page 242, and click **Next**.
14. In the next window, specify the WSDL details:
   - Select **Generate as a single WSDL file**
   - Type `www.ibm.com` in the WSDL Namespace
   - Type a name for the Definition Name; we used `itsoXX_db2vsam_wsdl`
   Refer to Figure 9-25 on page 243. Click **Next**.
In the next window, the “Add Operations to the WSDL Definition” window, add the following operations to the WSDL:

- Type `getCustomer` in the Name.
- In Input, select `CustomerReq`.
- In Output, select `CustomerReply`.
- In Fault, select `CustomerReply`.

---

Chapter 9. Exposing data as a service by using IBM WebSphere Message Broker
Click **Next**. Refer to Figure 9-26.

![Diagram](image-url)

**Figure 9-26 Adding operations to the WSDL definition**

16. In the next window, the Configure binding details window:

- Select **SOAP/HTTP** as the bindings.
- In the Port address, type the endpoint of the service. In our sample, we used:

  http://wtscz1.itso.ibm.com:7800/itsoXX/db2VSAM/getCustomerInfo

Click **Next**. Refer to Figure 9-27 on page 245.
A summary of the WSDL that was generated by the wizard is shown now (Figure 9-28 on page 246). Click Finish. Click File → Save.
9.2.4 Building the message flow

In the following steps, we build the message flow for what we want to achieve.

1. Create a new message flow project. From the menu bar, select File → New → Message Flow Project.

2. In the following window, give a name to the project. We used itso_zsoa_DB2VSAM in our sample. Click Next. Refer to Figure 9-29 on page 247.
3. In the next window, specify dependencies of the message flow project. Check the boxes for both *itsoXX_zsoa_CopyVSAM_CustRec* and *itsoXX_zsoa_xsd_CustReqReply*. Click **Finish**. Refer to Figure 9-30 on page 248.
We now create the Message Flow within the project that we have just created.

4. Highlight **itso_zsoa_DB2VSAM → Flows**. Right-click and select **New → Message Flow**. Refer to Figure 9-31 on page 249.
5. In the next window:
   - Type a name for the Message Flow name. We used `itso_zsoa_DB2VSAM`.
   - Clear the check mark from **Use default broker schema** and type `itsoXX` in schema.

   Click **Finish**. Refer to Figure 9-32 on page 250.
Figure 9-32  Specifying more details for the message flow

The canvas opens now with an empty flow, as shown in Figure 9-33 on page 251.
We will now start the process of building the flow, which consists of placing nodes on the canvas, naming them, eventually specifying attributes, and connecting them.

First, we will import the generated WSDL to the canvas, which describes a Web service interface into our message flow. Our purpose is that Web service consumers can invoke this message flow from any place as a Web service.

6. Expand the message set itsoXX_zsoa_xsd_CustReqReply and navigate to itsoXX_db2vsam_wsdIService.wsdl within Deployable WSDL.

7. Drag the itsoXX_db2vsam_wsdIService.wsdl to the canvas by selecting it with the left mouse button and moving it to the canvas while holding down the left mouse button. A wizard opens as soon as you release the mouse button. This wizard is called “Configure Web service usage.”

8. Select **Expose message flow as web service** in the wizard, and click **Next**. Refer to Figure 9-34 on page 252. Leave the rest of the fields to the default.
9. In the next window, Flow Generation Details, select **SOAP nodes**, and click **Finish**. Refer to Figure 9-35 on page 253.
A few nodes have been added to the canvas, as shown in Figure 9-36 on page 254. The nodes added represent a SOAP request, a Web service, and a SOAP reply. This canvas is the base for expanding the flow.
We 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:

a. Click a folder, for example, WebSphere MQ or Routing, to display the possible nodes within that folder.

b. Click a node within the folder, for example, the MQInput node in the WebSphere MQ folder, to select the node.

c. Move the cursor to an empty area in 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 highlighted. You can retype the name of the node.

Continue with this procedure to place all of the nodes required for this flow onto the canvas. The node names can be found in Figure 9-37 on page 255 and in Table 9-1 on page 256.

We explain the customization and function of the major nodes in this flow next. We advise you not to save the flow until you have finished configuring all of the nodes within this section. If you perform a save before you finish, several error messages will display, because the Toolkit detects that several nodes have not (yet) been set up correctly. Figure 9-37 on page 255 provides the details of the nodes and their names. Note that there is no specific order necessary for creating the nodes.
Figure 9-37  All nodes required for the message flow without wiring
### Table 9-1  DB2VSAM flow nodes

<table>
<thead>
<tr>
<th>Node</th>
<th>In folder</th>
<th>Node type</th>
<th>Rename to</th>
<th>Outgoing terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>WebSphere MQ</td>
<td>MQInput</td>
<td>DB2VSAM.XML.REQUEST</td>
<td>Out</td>
</tr>
<tr>
<td>B</td>
<td>Transformation</td>
<td>Compute</td>
<td>Set_DB_target</td>
<td>Out</td>
</tr>
<tr>
<td>C</td>
<td>Routing</td>
<td>RouteToLabel</td>
<td>No rename</td>
<td>None</td>
</tr>
<tr>
<td>D</td>
<td>Routing</td>
<td>Label</td>
<td>CustNo1-10</td>
<td>Out</td>
</tr>
<tr>
<td>E</td>
<td>Routing</td>
<td>Label</td>
<td>CustNo11-20</td>
<td>Out</td>
</tr>
<tr>
<td>F</td>
<td>Routing</td>
<td>Label</td>
<td>CustNoInvalid</td>
<td>Out</td>
</tr>
<tr>
<td>G</td>
<td>Transformation</td>
<td>Compute</td>
<td>Query_DB2</td>
<td>Out</td>
</tr>
<tr>
<td>H</td>
<td>Transformation</td>
<td>Compute</td>
<td>Prepare_VSAM_Query</td>
<td>Out</td>
</tr>
<tr>
<td>I</td>
<td>Transformation</td>
<td>Compute</td>
<td>Invalid_CustNo</td>
<td>Out</td>
</tr>
<tr>
<td>J</td>
<td>Additional IBM VSAM nodes</td>
<td>VSAMRead</td>
<td>VSAM_Lookup</td>
<td>Out</td>
</tr>
<tr>
<td>K</td>
<td>Transformation</td>
<td>Compute</td>
<td>CompleteReply</td>
<td>Out</td>
</tr>
<tr>
<td>L</td>
<td>Routing</td>
<td>Route</td>
<td>Return_to_MQ</td>
<td>Match to MQReply Default to SOAP Reply</td>
</tr>
<tr>
<td>M</td>
<td>WebSphere MQ</td>
<td>MQReply</td>
<td>No rename</td>
<td>None</td>
</tr>
</tbody>
</table>

**Important:** Be sure to specify the node names exactly as shown in Table 9-1 on page 256, especially for the compute nodes. In general, these node names do not matter, but in order to complete this sample, you must use the exact names, because we will refer to imported ESQL code blocks with those exact names later.
10. After you have placed all the nodes on the canvas, wire them as illustrated in Figure 9-38.

In order to “wire” the nodes together:

– Pass the mouse arrow over the outgoing terminal on the right of a node. When the mouse hovers over a terminal, the terminal name is displayed. Be sure that you have the correct terminal (refer to Table 9-1 on page 256).

– Click with the left mouse button to “take” or grab the terminal. Hold the mouse button down.

– Drag the line that now appears until you get to the incoming terminal of the node to which you want to connect the wire. You can now click again with the left mouse button to set the wire destination.

Note: For compute nodes, the procedure differs slightly. On the second step we just discussed, when you click the large outgoing terminal of a compute node, a window appears and you need to select the appropriate outgoing terminal. For this sample, always select the Out terminal, and click OK. You can then proceed by dragging the line.

Figure 9-38  All nodes required for the message flow, wired
9.2.5 Customizing the nodes

In the following steps, we customize the nodes. In specific cases, we only need to specify attributes, in certain cases, we need to customize ESQL code, and in other cases, we need to both specify attributes and customize ESQL code.

Customizing ESQL
ESQL is a descriptive language used in a message flow. Each compute node has default ESQL associated to it when the node is created. In all cases, this default ESQL needs to be customized to suit your needs. If you double-click a compute node, it opens the default ESQL portion associated to that node. All ESQL for the compute nodes in a message flow are kept in one file, and the file name is the message flow name and the .esql file extension.

DB2VSAM.XML.REQUEST node
This node receives a message from an WebSphere MQ queue. We need to describe from which queue this node gets the message:

1. Select the DB2VSAM.XML.REQUEST node, and click it. Beneath the flow editor window, you see the Properties tab. Select the Basic tab. Fill in the Queue name field with a queue name as defined in your WebSphere MQ environment. We used ITSOXX.ZS0A.DB2VSAM.XML.CUST.REQUEST. Refer to Figure 9-39 on page 259.
2. Next, click the side tab marked **Input Message Parsing.** In the Message domain drop-down list box, select **XMLNSC,** as shown in Figure 9-40 on page 260.

This action tells the Broker that, by default, it needs to parse the incoming message with an XML parser that supports namespaces. Note that WebSphere Message Broker also includes other XML parsers, for instance, the machine readable material (MRM) XML parser that requires that the message is predefined with a schema.

For simplicity, we have chosen to use the simpler XMLNS parser for this sample.
Figure 9-40 Input Message Parsing option

Set_DB_Target compute node
To create the ESQL for the Set_DB_Target compute node:

1. Return to the message flow editor by clicking the tab marked itso_zsoa_DB2VSAM.msgflow in the upper right editor pane. Now, double-click the first compute node labeled Set_DB_target (alternatively, you can right-click and select Open ESQL), which brings up the ESQL editor in the upper right panel.

   **Important:** Note that this action brings up the entire file with all the default ESQL code for the entire flow. We now replace each portion for each node one by one. Also, note that each ESQL module gets the name of the label of the node, which is prefixed with the name of the message flow.

2. Replace the default ESQL with the customized ESQL, as shown in Example 9-4.

   **Note:** We have included this code block as part of itso_zsoa_DB2VSAM_flow.esql.txt file in the itso_zsoa_Lab_Materials → Imports folder, which is inside the workspace packaged in the additional material. For the additional material information, refer to Appendix B, “Additional material” on page 485.

**Example 9-4  ESQL for Set_DB_target compute node**

```sql
CREATE COMPUTE MODULE itso_zsoa_DB2VSAM_Set_DB_target
  CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
    /* Determine which database to access for customer information, DB2 or VSAM */
    CALL CopyEntireMessage();
    SET OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelname = CASE
      WHEN InputRoot.XMLNSC.*:CustomerReq.*:CustNo < 11 THEN 'do_db2'
```
WHEN InputRoot.XMLNSC.*:CustomerReq.*:CustNo < 20 THEN 'do_vsam'
ELSE 'do_invalid'
END;
RETURN TRUE;
END;

CREATE PROCEDURE CopyEntireMessage() BEGIN
SET OutputRoot = InputRoot;
END;
END MODULE;

3. Save the ESQL code into the flow by pressing Ctrl+s.

The only code developed is shown in Example 9-5.

Example 9-5  ESQL code developed

SET OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelname =
CASE
    WHEN InputRoot.XMLNSC.*:CustomerReq.*:CustNo < 11 THEN 'do_db2'
    WHEN InputRoot.XMLNSC.*: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 to where the flow will branch in the subsequent RouteToLabel node. This sequence of nodes performs the classic "select" logic that is found in most second generation programming languages.

In addition to this ESQL code, there is a small configuration change that is necessary for the Set_DB_target node.

4. Return to the message flow editor window tab by clicking the tab marked itso_zsoa_DB2VSAM.msgflow, and click the Set_DB_target node one time. In the Properties page shown in Figure 9-41 on page 262, make sure that the Basic tab is selected as the left tab.

5. You must select LocalEnvironment and Message as the compute mode, as shown in Figure 9-41 on page 262. This selection is necessary to ensure that this node passes the local environment data to the following node. The local environment data is additional data that is carried in the flow along with the message. This local environment data is set in the ESQL code displayed in Example 9-5.
The final node in the upper section of this flow is the RouteToLabel node. This node simply performs a “branch” to the label set in the prior Set_DB_target node. No configuration is needed here.

**CustNo1-10, CustNo11-20, and CustNoInvalid Label nodes**

The lower part of this flow begins with three label nodes. Each label node needs to be configured with the appropriate branch point label. The names that are used here must match with the names that are used in the ESQL in “Set_DB_Target compute node” on page 260:

1. Select the node marked **CustNo1-10** and make sure that you have the Properties tab selected, as well as the Basic tab on the left. This node is the branch point when the incoming request is for customer number 1 - 10, in which case, the flow queries DB2.

2. Type the value **do_db2** in the branch Label name field as shown in Figure 9-42.

3. Perform the same customization on the other two label nodes:
   - The **CustNo11-20** node has the branch Label name of **do_vsam**.
   - The **CustNoInvalid** node has the branch Label name of **do_invalid**.
Query_DB2 node

To create the ESQL code for the Query_DB2 node:

1. As in “Set_DB_Target compute node” on page 260, open the ESQL code for the compute node named Query_DB2.

Replace the default ESQL code of this module with the customized ESQL code that is shown in Example 9-6.

Example 9-6  ESQL for Query_DB2 node

```esql
CREATE COMPUTE MODULE itso_zsoa_DB2VSAM_Query_DB2
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
/* Retrieve customer data from DB2 */
CALL CopyMessageHeaders();
SET OutputRoot.XMLNSC.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.ITSO0C.EOTCUST AS A
WHERE A.CUST_NO = InputRoot.XMLNSC.*:CustomerReq.*:CustNo);
/* Verify if DB2 inquiry found the customer record */
IF (Cardinality(OutputRoot.XMLNSC.CustomerReply[]) = 1 ) THEN
  SET OutputRoot.XMLNSC.CustomerReply.RetCode = 0;
ELSE
  SET OutputRoot.XMLNSC.CustomerReply.RetCode = 8;
END IF;
RETURN TRUE;
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;
```

Note: We have included this code block as part of itso_zsoa_DB2VSAM_flow.esql.txt file in the itso_zsoa_Lab_Materials Imports folder, which is inside the workspace packaged in the additional material. For the additional material information, refer to Appendix B, “Additional material” on page 485.
The first line of customized code (SET) performs the actual database lookup with a SELECT statement that is virtually identical to standard SQL, which, of course, explains why the native broker language is called ESQL (Extended SQL). The second line of customized code (IF) simply validates if reasonable data was returned from the DB2 SELECT and assigns an appropriate return code.

In addition to supplying the ESQL code in Example 9-6 on page 263, it is necessary to perform a minor configuration change for this compute node.

2. Return to the Message Flow canvas, and click the DB2_Query node.
3. In the Properties tab, select the Basic side tab. Type the location name of the z/OS DB2 database in the Data Source field. In our sample, we use DB9H.
4. Browse and select itsoXX.itso_zsoa_DB2VSAM_Query_DB2 as the ESQL module.

Refer to Figure 9-43.

Figure 9-43   Query_DB2 properties

Prepare_VSAM_Query node
To create the ESQL for the Prepare_VSAM_Query node:

1. Move to the Message Flow editor window and select the Prepare_VSAM_Query node. Double-click this node, which opens the ESQL code associated with this node.

This node gets executed when 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. The entire ESQL code associated to this node is shown in Example 9-7 on page 265.

The code printed in bold has been added to the default code (the code between BEGIN and RETURN TRUE).
Example 9-7  ESQL code for the Prepare_VSAM_Query node

CREATE COMPUTE MODULE itso_zsoa_DB2VSAM_Prepare_VSAM_Query
  CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
    /* Position and format the VSAM search key where the VSAM node expects to retrieve it */
    CALL CopyMessageHeaders();
    DECLARE Pattern CHARACTER '000';
    SET OutputRoot.XMLNSC.VSAM.Request.Position.Key =
      CAST( CAST(InputRoot.XMLNSC.*:CustomerReq.*:CustNo AS INTEGER) AS CHARACTER
      FORMAT Pattern);
    RETURN TRUE;
  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;

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.

VSAM_Lookup node

To configure this node:

1. Move to the Message Flow editor window and select the VSAMRead node labeled VSAM_Lookup, which must be configured.
2. In the lower Properties page, make sure that the Basic tab is selected.
3. Set the Default File name to the VSAM data set that you plan to use. In our sample, we used `//ITSOXX.CICS.ITSOVSAM` (single quotation marks included). Refer to Figure 9-44.

![Figure 9-44 Default tab of the VSAMRead node](image)

4. Select the **Default** tab and:
   - Set the Message Domain to MRM.
   - Specify the message set ID that you noted earlier in the sample.
   - Type `msg_CUSTREC` for the Message Type.
   - Type `Binary1` for the Message Format.

Refer to Figure 9-45.

![Figure 9-45 Default tab in VSAMRead node](image)
5. Select the **Request** tab and:
   - Type `InputRoot.XMLNSC.VSAM.Request` as the Request Location.
   - Select **KEY_EQ - Key equal** as the Position Mode.
   - Select **Parser String** as the Key Type.

   **Note:** An error message appears on this menu “X RRN: Must be a value greater than 0.” Ignore this message.

Refer to Figure 9-46.

![Figure 9-46 Request tab for the VSAMRead node](image)

6. Select the **Result** tab. Type `OutputRoot.XMLNSC.CustomerReply` as the Output Data Location. Refer to Figure 9-47.

![Figure 9-47 Result tab for the VSAMRead node](image)

**CompleteReply node**

Move to the Message Flow editor window and select the compute node labeled **CompleteReply**. Double-click the node to show its ESQL code. This node uses
the following ESQL that is shown in Example 9-8. The customized (added) code is in bold.

Example 9-8 CompleteReply ESQL code

```
CREATE COMPUTE MODULE itso_zsoa_DB2VSAM_CompleteReply
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        /* For VSAM returned data, check if inquiry found a customer record */
        CALL CopyEntireMessage();
        IF (InputLocalEnvironment.VSAM.Report.Length > 0 ) THEN
            SET OutputRoot.XMLNSC.CustomerReply.RetCode = 0;
        ELSE
            SET OutputRoot.XMLNSC.CustomerReply.RetCode = 8;
        END IF;
        RETURN TRUE;
    END;

    CREATE PROCEDURE CopyEntireMessage() BEGIN
        SET OutputRoot = InputRoot;
    END;
END
```

This code performs a simple verification if the VSAM search returned valid data and sets a return code appropriately.

Return_to_MQ node

The routing logic for this node uses the ReplyProtocol field in the Properties folder. Based on the protocol that is used for the request message, the reply message is routed appropriately. The routing expression used in this node is built with the XPath Expression® Builder wizard. Follow these steps to build the expression for the routing logic for this node:

1. Left-click the Return_to_MQ node.
2. In the Properties window, select the Basic tab, and click Add.
3. In the Add filter table entry window, as shown in Figure 9-48 on page 269, click Edit.
4. In the XPath Expression Builder window, expand Variables $Root, as shown in Figure 9-49.

5. Expand Properties.

6. Double-click ReplyProtocol. The XPath expression is built now, as shown in Figure 9-50 on page 270.
7. In the Operators pane, click = (equal symbol), as shown in Figure 9-51 on page 271.
8. In the XPath Expression field, type ‘MQ’ (including single quotation marks) after the = (equal symbol) just inserted. The expression now looks as shown in Figure 9-52 on page 272.
9. Click Finish.

10. The table entry is built now. Click OK in the Add Filter entry table window.

The expression created is the “match” expression. If the condition in this expression is met, the branch that is connected to the output terminal named match will get executed.

**Invalid_CustNo**

Now, we look at the second branch option. Move to the Message Flow editor window and select the compute node labeled **Invalid_CustNo**, which has the ESQL code that is shown in Example 9-9 on page 273. The added customized code is in bold. In this code, we set a return code of 12, because there is no record found.
Chapter 9. Exposing data as a service by using IBM WebSphere Message Broker

Example 9-9  ESQL code of the Invalid_CustNo node

```sql
CREATE COMPUTE MODULE itso_zsoa_DB2VSAM_Invalid_CustNo
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    /* Provide return code for a request out of range */
    CALL CopyMessageHeaders();
    SET OutputRoot.XMLNSC.CustomerReply.RetCode = 12;
    RETURN TRUE;
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;
```

9.2.6 Deploying and testing the flow

Now that the message and flow definitions are complete, you can create a Broker Archive (.bar) file with the artifacts, deploy it to the Broker run time, and test it. A WebSphere Message Broker uses the .bar file in the same way that WebSphere Application Server uses .war or .ear files; the .bar file is a simple file that contains all the necessary components in a compressed, unified format for easy deploying to a runtime environment. To deploy and test the flow:

1. Select the Broker Administration perspective in the Toolkit in the upper right part of the Toolkit, as shown in Figure 9-53.

![Figure 9-53  Select Broker Administration perspective](image-url)
2. In the Broker Administration pane found in the upper left, right-click the **Broker Archives** folder. Select **New → Message Broker Archive**, as shown in Figure 9-54.

![Figure 9-54 Create New Message Broker Archive](image)

3. This action brings up the Message Broker Archive creation window. You can use any name when filling in the name of a .bar file to create. The .bar file typically has a name that indicates the project. Type **Servers** in the Project field and **DB2VSAM** in the Name field. Click **Finish**. Refer to Figure 9-55 on page 275.
4. A window opens now in which you can describe the contents of the bar file. Select the following check boxes:
   - `itso_zsoa_DB2VSAM.msgflow` in the Message Flows folder
   - `messageSet.set` in the Message Sets folder
     - `/itsoXX_zsoa_CopyVSAM_CustRec/itsoXX_zsoa_CopyVSAM_CustRec/messageSet.set` in the Message Sets folder
   - `messageSet.set` in the Message Sets folder
     - `/itsoXX_zsoa_xsd_CustRecqReply/itsoXX_zsoa_xsd_CustReqReply/messageSet.set` in the Message Sets folder

5. Click **Build message broker archive**. Refer to Figure 9-56 on page 276.
6. A confirmation window is displayed; click OK to dismiss the window, and the .bar editor displays the artifact names, as shown in Figure 9-57 on page 277.

7. Click the Manage tab to see the contents of the bar file. Refer to Figure 9-57 on page 277.
8. 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 Toolkit to the Configuration Manager. This connection only needs to be done once each time that you start the Toolkit.

The Configuration Manager controls and filters access to the Broker. All artifacts are first verified by the Configuration Manager.

9. In the lower left Domain window, right-click the Configuration Manager domain and select **Connect**. Refer to Figure 9-58.

---

**Figure 9-57  Overview of the .bar file contents**

**Figure 9-58  Connecting the Toolkit to the Configuration Manager**
After a few seconds, you see the Broker name, MQA1BRK, appear under the Broker Topology heading. You also see several Broker Execution Groups appear under the Broker name.

Now, right-click the .bar file in the Navigator pane, and select the option **Deploy File** from the pop-up menu, as shown in Figure 9-59.

![Figure 9-59   Start deployment](image)

10. In the pop-up window, select the execution group that you want. Refer to Figure 9-60 on page 279.
Click OK to continue. An information window appears confirming that your .bar file has been deployed. You can click OK to dismiss this window also. Notice a few seconds later that your flow and message set names now appear in the Broker topology display in the Domains pane under your execution group name. Refer to Figure 9-61 on page 280.
You can now test your new flow.

### 9.2.7 Testing the flow

You can test the message flow in various ways:

- Using the Web Services Explorer, which is built into the WebSphere Message Broker Toolkit. You can follow this procedure if you have a SOAP node in your message flow. In that case, your request is a SOAP message. Refer to “Testing using the Web Services Explorer” on page 281.

- Using the RFHUTILC utility, which is a popular tool for testing applications using WebSphere MQ. You can use this tool to send a request message to the input queue for the message flow and then read a reply message from the reply queue of the message flow. Refer to “Testing using RFHUTILC” on page 284.
Using the nettool. The nettool is an open source tool for testing Web applications. In this example, you can use the nettool to send an XML message to the SOAP node of the message flow.

Testing using the Web Services Explorer
To test the message flow with the Web Services Explorer, follow these steps:

1. Click **Window → Open perspective → Other**.
2. Select **Web perspective** and click **OK**.
3. Expand the **itsoXX_zsoa_xsd_CustReq_Reply** folder until you see the WSDL file that is used for the SOAP node.
4. Select the WSDL file and right-click it, which opens a pop-up window.
5. In the pop-up window, select **Web Services → Test with Web Services Explorer**, as shown in Figure 9-62 on page 282.
6. In the Web Services Explorer pane, expand the WSDL Main tree in the Navigator until you get to the `getCustomer` method. Click `getCustomer` and you see an input field for Customer number on the right side. Fill in a value and click **Go**, as shown in Figure 9-63 on page 283.
Figure 9-63  Testing with the Web Services Explorer

7. The output is shown in Figure 9-64 on page 284.
Testing using RFHUTILC

To test the message flow with RFHUTILC, follow these steps:

1. Start the RFHUTILC tool by double-clicking the shortcut. The RFHUTILC window opens. Make sure that the Main tab is selected.

2. Type the Queue Name, which is the name of the queue that you will use for the request message. In our example, this name is ITSOXX.ZSOA.DBSAM.XML.CUST.REQUEST. Refer to Figure 9-65 on page 285.
3. Click **Open File**.

4. Navigate to the file that you are using with test data. In our case, we used a file with XML test data, as shown in Example 9-10.

**Example 9-10  XML test data**

```xml
<CustomerReq>
  <CustNo>5</CustNo>
</CustomerReq>
```

---

Note: The queue name specified here is the queue name that we also specified in the DB2VSAM.XML.REQUEST node in “DB2VSAM.XML.REQUEST node” on page 258.
5. Click the **Data** tab. You now see the test data in raw format. To be able to see the test data in the original (XML) format, click **XML** on the right side of the pane. Refer to Figure 9-66.

![Figure 9-66 XML test data in RFHUTILC](image)

6. Click the **MQMD** tab.

7. In the MQ Message Format field, type **MQSTR**, which indicates STRING.

8. In the Reply to Queue field, type the name of the reply queue. In our example, we used **ITSOXX.ZSOA.DB2VSAM.XML.CUST.REPLY**.

   Refer to Figure 9-67 on page 287.
9. Click the **Main** tab, and click **Write Q**.

10. Change the Queue Name to the name of reply queue. In our example, we used `ITSOXX.ZS0A.DB2VSAM.XML.CUST.REPLY`.

11. Click **Read Q**.

12. Go to the **Data** tab, and you can see the reply message.
   
   Refer to Figure 9-68 on page 288.
Figure 9-68  Output
Exposing an IMS application as a Web service

In this chapter, we describe a sample of an IMS application that is enabled as a Web service using IMS SOAP Gateway and Rational Developer for System z.
10.1 Introduction

To expose an IMS application as a Web service, you first need to create a Web Services Description Language (WSDL) file. A WSDL file is an XML document that describes a Web service. WSDL files are used by other clients (for example, the client that invokes the service) to discover the service and to understand how to invoke the service. The WSDL specifies the location of the service and the operations that the service exposes.

The Enterprise Service Tools (EST) wizard in Rational Developer for System z can be used to generate the WSDL file that is needed to enable your IMS application to run as a Web service provider.

You must have a description of the format of the input and output messages for your IMS application, also known as a copybook.

For using IMS SOAP Gateway, you must have Rational Developer for System z (RDz) Version 7.1.1 or later.

10.2 Steps

To expose an IMS application as a Web service through IMS SOAP Gateway, perform the following steps using RDz. We explain each step in more detail in a separate section:

1. Create a new IMS SOAP Gateway project.
   In this step, we create a project in RDz and import the copybook. The copybook is everything you need to go through the process.

2. Generate the IMS SOAP Gateway Resources.
   In this step, we create a number of artifacts that are needed to deploy the service, such as the WSDL file and the XML converter.

3. Compile the XML converter.
   The XML converter is a COBOL program that needs to be compiled, link-edited, and implemented on z/OS.

4. Enable the IMS application as a Web service.
   After the artifacts have been created, you need to deploy the Web service to the IMS SOAP Gateway.
10.2.1 Create a new IMS SOAP Gateway project

Create a new IMS SOAP Gateway project and import the COBOL copybook:

1. Start Rational Developer for System z, and open the Enterprise Service Tools (EST) perspective.
2. Select a wizard by selecting File → New → Other from the menu. In the Select a wizard panel, expand Enterprise Service Tools and select IMS SOAP Gateway Project.
3. Click Next to open the New IMS SOAP Gateway project wizard.

Refer to Figure 10-1.
4. In the New IMS SOAP Gateway Project wizard:
   a. In the Choose the project name field, type a name for the project, for example, MyProject.
   b. Click Next to open the import source files.
   c. In the Import source files panel, click Remote.
   d. In the Browse For File panel, expand COBOL source library, select the COBOL copybook, and click OK to open the Check Dependencies dialog.

   **Note:** The COBOL copybook is part of the additional material. Refer to Appendix B, “Additional material” on page 485 for instructions to obtain the copybook.

   e. In the Check Dependencies panel, select **Import the selected source only**, and click OK.
   f. Click Finish to create the new IMS SOAP Gateway project.
10.2.2 Generate the IMS SOAP Gateway Resources

Perform the following steps to generate the IMS SOAP Gateway artifacts:

1. Start the Enterprise Service Tools (EST) Wizard Launchpad:
   - In the Navigator view, click **Project → Source** and right-click the imported source file that contains the input and output data structure statement of your application.
   - Select **Generate IMS SOAP Gateway resources**.
   - Click **Start** to open the Language structures page of the wizard.
2. In the Language structures page of wizard:
   a. Change the COBOL Preferences by clicking Change COBOL Preferences. The COBOL Preferences window opens and displays the COBOL preferences in the right pane. In the General tab, expand the Platform list box and select z/OS. Leave other options unchanged. Click OK to close the Preferences window.
   b. Under the Inbound Language Structure tab, select the COBOL structure that is the input structure for your IMS application.
   c. Under the Outbound Language Structure tab select the COBOL structure that is the output structure for your IMS application.
   d. Click Next to open the Generation Options page of the wizard.

Refer to Figure 10-3 on page 295.
3. In the Generation options page of the wizard:
   a. Under the **XML Converter** tab, select or specify the following settings if necessary:
      - Host code page: Select the code page that the host uses.
      - Inbound code page and outbound code page: Leave it at the default value, 1208 Unicode, UTF-8. IMS SOAP Gateway supports only UTF-8.
   b. Under the **WSDL and XSD** tab, in the Service location input field, change the host name and port number to the location of your IMS SOAP Gateway and leave the other fields unchanged.
c. Click **Next** to open the IMS SOAP Gateway Web Service Provider page of the wizard.

Refer to Figure 10-4.
4. In the IMS SOAP Gateway Web Service Provider page of the wizard:
   a. In the Specify IMS Connect Interaction properties section, you specify the parameters relevant for IMS Connect. You can specify or alter these values later using the IMS SOAP Gateway Deployment utility.
   b. Click **Next** to open the File, data set, or member selection page of the wizard.

Refer to Figure 10-5.

*Figure 10-5  IMS SOAP Gateway Web Service Provider page of the wizard*
5. In the File, data set, or member selection page of the wizard, perform the following steps and change fields only when necessary:

- Under the **XML Converters** tab, select **Generate all to driver**. This selection causes all the generated Web service programs (driver, inbound converter, and outbound converter) to be placed in the same file.

- Under the **WSDL and XSD** tab, RDz sets the first seven characters of the COBOL copybook as default values and uses the COBOL copybook name in the endpoint URI as the default value as well.

- In the **WSDL and XSD** tab, also make sure that the check box preceding the input file WSDL file name is selected.

- Click **Finish** to generate the following IMS SOAP Gateway resources:
  - WDSL (.wsdl) file, which is the only file that you need to enable your IMS application as a Web service provider by using IMS SOAP Gateway
  - Correlator file (.xml)
  - The generated file containing the Web service driver and runtime XML converter (.cbl)
  - The input and output data mapping XDS files (.xsd)

Refer to Figure 10-6 on page 299.
File, data set, or member selection
Select the source and targets for the Web Services enablement artifacts.

Interface file container: /Phonebook/Generation/Target
- WSDL file name: DFSIVA34 .wsdl
- Inbound XSD file name: DFSIVA3I .xsd
- Outbound XSD file name: DFSIVA3O .xsd

Overwrite files without warning

Figure 10-6  File, data set, or member selection page of wizard
10.2.3 Compile the XML converter

To start:

1. Upload the generated XML converter to your COBOL source library, which can be any partitioned data set (PDS) set on z/OS with the right format.

2. Compile and link-edit the generated XML converter so that it can be accessed by IMS Connect, which means that the converter needs to be placed in a load library part of the IMS Connect JCL. Note that the XML converter must have alias name suffix “X”.

10.2.4 Enable the IMS application as a Web service

The next steps are needed to define the Web service to the IMS SOAP Gateway. You can use the IMS SOAP Gateway Deployment Utility to perform the following steps:

1. Upload the WSDL file by using RDz using a drag and drop technique, by using FTP from the command line, or by using an FTP tool. Note that you have to upload the WSDL file in binary mode.

2. Start the IMS SOAP Gateway Deployment Utility from the UNIX System Services command line. You can bring up a Telnet emulation from RDz, or alternatively, you can log on to z/OS and go into OMVS mode. When you are in the Uniform Symbol Specification, perform these tasks:


   b. Follow the instructions to enable your IMS application as a Web service from start to finish.

For more information about the IMS SOAP Gateway Deployment Utility, refer to *IMS SOAP Gateway User’s Guide and Reference*, SC19-1290.

The main menu of the utility is shown in Figure 10-7 on page 301.
Figure 10-7   The IMS SOAP Gateway Deployment Utility main menu

c. From the IMS SOAP Gateway Deployment Utility menu, type 1 and press Enter.

d. Type the name and location of the generated WSDL file, as shown in Figure 10-8 on page 302.
Task 1 guides you through a series of steps to publish your IMS application as a Web service.

Note: You must have IMS SOAP Gateway started and a WSDL file created for your IMS application before you enter this task.

Step a: Provide the WSDL file for deployment

To enable your IMS application as a Web Service, you need a WSDL file. You can create a WSDL file for your IMS application manually or use a development tool.

At the prompt, specify the name and location of the WSDL file you have created (e.g. `c:\MyWSDLFile.wsdl`).

If you had deployed the WSDL before or you have copied the WSDL into the IMS SOAP Gateway `wsdl` directory, provide just the name for the WSDL file (e.g. `MyWSDL.wsdl`), the full path is NOT required.

> Provide the name and location of the WSDL file:

/u/itsoam/DFSIVA34.wsdl

Figure 10-8  Provide the WSDL file

---
e. In Provide Connection properties for connecting to IMS, specify the connection and security information for the Web service to create or reuse a connection bundle, as shown in Figure 10-9 on page 303. In this case, we reuse a connection bundle.
Step b: Provide Connection properties for connecting to IMS
---------------------------------------------------------------
This step allows you to create or reuse a connection bundle that consists of properties for connecting to IMS.

Do you want to view all existing connection bundles? (y/n):
y
  Connection Bundle Name : connbundle1
    Host name       : wtsczl.itso.ibm.com
    Port Number            : 6001
    Datastore       : IMSA
    IMS UserID             :
    Group name      :
    SSLKeystoreName       :
    SSLKeystorePassword   :
    SSLTruststoreName       :
    SSLTruststorePassword       :
    SSLEncryption type       :
    calloutTPipes       :

Do you want to create a new connection bundle? (y/n):
n
Provide name of an existing connection bundle:
  connbundle1

Figure 10-9   Provide Connection properties

f. Provide Interaction (Correlator) properties for the Web service by specifying the correlator file name or creating a new correlator file, as shown in Figure 10-10 on page 304. In this case, we create a file.
Step c: Provide Interaction (Correlator) properties for Web service

This step allows you to create or reuse a correlator file that consists of properties for interacting with the IMS application for the Web service.

If you use the IBM WebSphere Developer for zSeries (WDz) tooling, the correlator file is generated for you as part of the process. Otherwise, you will have to create the correlator file.

Do you want to use an existing Correlator file? (y/n):

n
- The SOAP action URN ‘DFSIVA34’ will be the file name.
- The correlator file name is /usr/lpp/ims/imssoap/server/webapps/imssoap/xml/DFSIVA34.xml
- Provide the following correlator properties.

  > Enter the converter name:
  DFSIVA3D
  > Enter the socket timeout value (in milliseconds; default is 5000):
  > Enter the execution timeout value (in milliseconds; default is 5000):
  > Enter the lterm name:
  > Enter the adapter type (default is ""):
  > Enter the connection bundle name (default: connbundle1):
  > Enter the IMS transaction code:
  ITVNO

Do you want to continue with another correlator? (y/n/cancel):

n
- The correlator file has been created.

Do you want to view the correlator file? (y/n):

y

Correlator entry 1 out of 1

Adapter Type : IBM XML Adapter
Converter Name : DFSIVA3D
Connection Bundle Name : connbundle1
Socket Timeout : 5000
Execution Timeout : 5000
Lterm Name :
Transaction Code : ITVNO

No more correlator entry. Press enter to return to the main menu.

Figure 10-10   Provide Interaction (Correlator) properties

g. Deploy the WSDL to IMS SOAP Gateway as shown in Figure 10-11 on page 305. Ensure that IMS SOAP Gateway is already started.
Step d: Deploy the WSDL to IMS SOAP Gateway
--------------------------------------------
This step deploys the WSDL file and makes your IMS application accessible as a Web service.
Note: Ensure that IMS SOAP gateway is already started.
> Do you want to deploy the WSDL file to IMS SOAP gateway? (y/n):
y
Figure 10-11  Deploy the WSDL to IMS SOAP Gateway

3. To verify that the Web service was deployed successfully, open the IMS SOAP Gateway Administrative Console and ensure that the deployed Web service is listed, as shown in Figure 10-12 on page 306.
The Web service is now ready to be called by sending a SOAP request from a Web service consumer. A Web service consumer can be created or generated using the WSDL document that we used earlier.
Building composite CICS services using the Service Flow Feature

In this chapter, we show you how to use the Service Flow Feature to build composite CICS services.
11.1 Introduction

Many enterprises run Customer Information Control System (CICS)-based applications on IBM System z servers to support their business operations. These organizations made huge investments in CICS application code and skills. As these enterprises start adopting a service-oriented architecture (SOA), they face the challenge of embracing new technologies while leveraging existing application assets. Often, they have to decide between rewriting or reusing these applications. In most cases, reusing existing applications in a noninvasive manner provides a more cost-effective and lower impact approach to integrating existing applications into a SOA than rewriting.

The integrated Web Services support of CICS Transaction Server for z/OS Version 3.1 and 3.2 (CICS TS) enables existing CICS communication area (COMMAREA)-based applications to become service providers. The Service Flow Feature (SFF) of CICS TS enables applications and services external to CICS TS to use and integrate existing 3270 applications, COMMAREA, channels and containers, external Web services, and other applications as callable services. SFF consists of the Service Flow Runtime (SFR) component and the Service Flow Modeler (SFM) development tool. SFM is part of IBM Rational Developer for System z Version 7.1 (RDz) and higher.

RDz supports the creation of CICS-based services through its Enterprise Service Tools (EST). EST enables the transformation of existing CICS-based applications into Web services by creating applications that run using the integrated Web Services and Service Flow capabilities of CICS TS.

In this section, we focus on the use of RDz to create a Service Flow application that will be deployed to and run on the SFR. RDz supports the creation of a comprehensive Web service that aggregates data and results from multiple sources, including CICS non-terminal applications, CICS terminal applications, and Web services. This support is provided through Service Flow projects.

A Service Flow project enables you to model a business service from existing CICS applications where one Web service interface is defined and described. This single Web service interface will be backed by multiple CICS-based applications. Consider the use of the single-service project if you only need to create a Web service interface that will be mapped to only one non-terminal or COMMAREA-based application.

Service Flow projects support either bottom-up or meet-in-the-middle development approaches. A bottom-up approach is used when the objective is to create a service from existing applications. A meet-in-the-middle approach is used when both the Web Services Description Language (WSDL) file describing the service and the implementation components representing the application
exist and additional support code that maps the two together needs to be created.

11.1.1 Introducing this sample

This sample will take the *meet-in-the-middle approach* by starting with a WSDL that describes the intended Web service interface. This WSDL describes the structures of the input and output messages associated with the Web service. We will then map the described interfaces to the data and results that will be aggregated from invoking an outbound Web service (CICS as a service requester) and interacting with an existing terminal-based application.

The CICS TS sample Catalog Manager application is the existing terminal-based application that we will use. This application is invoked from a CICS terminal through the EGUI transaction. We will refer to this application as the EGUI application. You can list available catalog items from the EGUI application. You can then place an order for an item. Figure 11-1 shows the menu options.

<table>
<thead>
<tr>
<th>CICS EXAMPLE CATALOG APPLICATION - Main Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select an action, then press ENTER</td>
</tr>
<tr>
<td>Action . . .</td>
</tr>
<tr>
<td>1. List Items</td>
</tr>
<tr>
<td>2. Order Item Number</td>
</tr>
<tr>
<td>3. Exit</td>
</tr>
</tbody>
</table>

Figure 11-1 Example Catalog Application (EGUI)

To place an order, you need to provide the item number, the order quantity, a user name, and the department to charge for the order. Refer to Figure 11-2 on page 310 for the original basic mapping support (BMS) map.
We will create a Service Flow application that will process an order based on an input request that consists of a customer number, an item number, and the quantity to be ordered. The expected output response will be a description of the item ordered, the total cost of the order, the status of the order, and the name associated with the customer number for this order.

This scenario assumes that a list of catalog items has previously been obtained (perhaps from another Web service call or another Service Flow application). Other information required for placing an order will be obtained by invoking a customer lookup Web service. This Web service has been deployed to a WebSphere Application Server.

We will therefore model a Service Flow that consists of two steps. The first step is to invoke a Web service to get customer data based on the customer number that is passed. The second step is to place an order by interacting with the EGUI application. Figure 11-3 shows the flow that we will create in RDz.
11.1.2 Objective

At the end of this sample, you will have an understanding of the capabilities of the Enterprise Service Tools of RDz and the Service Flow Runtime of CICS to transform existing CICS-based applications into Web services with the appropriate level of service granularity.

11.1.3 Prerequisite

In order to perform this sample, access is needed to a z/OS system with a CICS Transaction Server for z/OS region, Version 3.2, with the CICS Service Flow Runtime installed. The sample instructions are written based on access to a z/OS 1.9 system, which is referred to as “z1”. You will also need to have Rational Developer for System z Version 7.1 installed on your workstation. You will also need to have built and deployed a GetCustomer service to WebSphere Application Server, as described in Example 11-1.

Example 11-1  SQL for Customer lookup by Customer Number

SELECT CUST_ADDR1, CUST_CITY, CUST_NO, DEPT, CUST_FN, CUST_LN
    FROM ITSO0C.EOTCUST
WHERE CUST_NO = :CUST_NO

We also assume that you are familiar with using an Eclipse-based development environment, such as RDz.

11.2 Creating a Service Flow project

The first step is to create a Service Flow project in RDz:

1. Start Rational Developer for System z (RDz).

2. On the Workspace Launcher dialog, specify the workspace name that you want to use and click OK. We used C:\Workspaces\sfm as the workspace. You can use Browse if necessary. Refer to Figure 11-4 on page 312.
3. Switch to the **Enterprise Service Tools** perspective (if you are already there, proceed to the next step).

   If the Welcome view is open, close it. Click the Open Perspective icon ( ) and select **Other** from the list.

4. From the Open Perspective dialog, select **Enterprise Service Tools** and click **OK**, as shown in Figure 11-6 on page 313.
You now see a window similar to Figure 11-7.

Notice the EST Project Explorer view, which is where EST-related projects will be organized and where the tools related to Service Flow development are launched.

5. Set preferences for Service Flow projects:
   a. Click Window → Preferences on the menu bar to bring up the Preferences dialog.
b. On the left pane of the Preferences dialog, navigate to the entry for Enterprise Service Tools. Click the + (addition symbol) to expand the subentries. Select **Service Flow Projects**, and on the right pane, be sure that **Do not prompt for variable when generating variable mapping** in the Recording Flow section is selected. Refer to Figure 11-8. By setting this preference, you are choosing to have automatic field names, originating from the BMS source, created for you during flow recording of the terminal interactions. If you choose not to set this preference, you are prompted to supply the field names each time that data is entered on a window or extracted from a window.

![Preferences](image)

*Figure 11-8 Setting preferences for Service Flow Project*

c. There is a set of JCL templates that is used by SFM to generate the jobs that will be used to build and deploy the Service Flow applications. You can modify these templates manually or import these templates from your target host system. In most cases, your host systems programmer has updated these templates to reflect where libraries, such as the COBOL compiler and Language Environment® (LE), are installed.

d. Click the + (addition symbol) beside **Service Flow Projects** to expand its subentries.

e. Select **JCL Templates** and you see a window similar to Figure 11-9 on page 315.
You can edit the templates in place, import from a local directory, or import copies from a host system. Importing from a host system requires that you have defined a system connection using the remote systems capability of RDz and are connected to that remote system. In this sample, we will show you the scenario when importing from a host system.

f. Click **Import from Host**. You see a window similar to Figure 11-10 on page 316. On that dialog, navigate to the remote system that you will use. For this sample, we show a system z1, which when expanded has a set of data sets available. In this instance, we will import the selected members from data set ITSOAM.SFRN.SCIZSAMP. Click **OK** to start importing.
After the import is complete, you see results dialog similar to Figure 11-11 on page 317. You can click **Details** to see additional status information.
g. Click **OK** to save your changes and dismiss the Preferences dialog.

6. Create a Service Flow project. Click anywhere in the EST Project Explorer view. Right-click anywhere in the view and select **New → Service Flow Project** from the context-sensitive menu, as shown in Figure 11-12.

7. In the New Service Flow Project dialog, type **CATAL**, as shown in Figure 11-13 on page 318. Click **Next**.
8. On the Associate service interface dialog, select **Import from a WSDL document** and click **Browse** beside the WSDL location entry field. Navigate to the directory where you downloaded the additional materials for this book and select **processOrder.wsdl**, which is shown in Figure 11-14 on page 319. Click **Next**.
9. In the Define service steps dialog, select **Naming the steps and implementing later**, as shown in Figure 11-15 on page 320. Click **Next**.
10. On the “Identify the steps of the flow” dialog, click **Add twice**.

This action creates two steps with default names of Step0 and Step1, as shown in Figure 11-16 on page 321, to represent the processing nodes in our flow. We will rename the steps to more descriptive names.
11. Click **Step0** and replace the name by typing over it with GetCustomerInfo, as shown in Figure 11-17 on page 322. Click **Step1** and replace it with OrderItem, as shown in Figure 11-17 on page 322. Click **Finish**.
If the Enterprise Service Tools Tip dialog appears, select **Not to show any Tips** and click **OK** to dismiss that dialog.

You see a flow diagram similar to what is shown in Figure 11-18 on page 323.
12. If your flow is not laid out properly, click anywhere in the flow editor. 
Right-click and select **Layout → Left to Right** from the context menu, which 
will give you a layout as shown in Figure 11-19.

Note that the CATAL Service Flow project is now in the EST Project Explorer 
view and that the flow editor is opened showing the flow that you have modeled 
so far.

Before proceeding with further refinement of the flow, let us explore the structure 
of a Service Flow project. You have the items as shown in Figure 11-20.
11.2.1 Understanding the Service Flow project structure

A Service Flow project provides the framework to organize the source and generated files associated with a Service Flow. Various artifacts are stored in a predefined set of folders and subprojects. A folder is used to store a single class or type of artifact. Table 11-1 shows the folders.

Table 11-1 Folders

<table>
<thead>
<tr>
<th>Folder</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flows</td>
<td>Used to store flows where a flow is a sequence of operations, assignments, and conditionals that are connected into a programmed unit that can be invoked.</td>
</tr>
<tr>
<td>Mapping</td>
<td>Used to store mappings that are used to manipulate data while a flow is performed.</td>
</tr>
<tr>
<td>Expressions</td>
<td>Used to store expressions written in ESQL that are used to control the sequencing of a flow.</td>
</tr>
<tr>
<td>Generation</td>
<td>Used to store generation properties files and generated runtime code files.</td>
</tr>
</tbody>
</table>

A subproject contains operations and messages that are used to invoke a particular type of entity or application. Table 11-2 shows these subproject types.

Table 11-2 Subproject types

<table>
<thead>
<tr>
<th>Subproject type</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface definition (CATAL.Interface)</td>
<td>Operations and messages invoking flows</td>
</tr>
<tr>
<td>Terminal applications (CATAL.Terminal)</td>
<td>Operations and messages for invoking terminal or window-based applications</td>
</tr>
<tr>
<td>Nonterminal applications (CATAL.Nonterminal)</td>
<td>Operations and messages for invoking nonterminal (non-window-based or COMMAREA-based) applications</td>
</tr>
<tr>
<td>Outbound Web services (CATAL.OutboundWebService)</td>
<td>Operations and messages for invoking outbound Web services</td>
</tr>
</tbody>
</table>

If you are interested in more details, launch the Help facility and look in Contents for Enterprise Service Tools for Web services and SOA → Developing Service Flow projects.
11.3 Defining the GetCustomerInfo flow step

So far, we have modeled a basic flow for the application that will process a customer order. We identified two steps as part of modeling the flow. These steps are represented in the flow diagram as nodes, called *invoke* nodes.

There are two invoke nodes, *GetCustomerInfo* and *OrderItem*, that correspond to the two steps that are needed to process an order. At this point, these invoke nodes are abstract notions that are not yet associated with any concrete applications. The sample will walk you through describing and associating the invoke nodes with actual applications.

In our discussion of the scenario in the introductory section, we mentioned that the customer information will be obtained by invoking a customer lookup Web service. This Web service is described in a WSDL that is provided. We will associate the WSDL file that is provided with the GetCustomerInfo invoke node.

Because this step will involve an outbound Web service call (outbound from the perspective of CICS, where CICS is acting as a service requester), we will be importing the WSDL describing the Web service into the outbound Web service subproject (*CATAL.OutboundWebService*):

1. In the EST Project Explorer view, select the *CATAL.OutboundWebService* subproject. Right-click it and select *Import CICS Web Service* from the context menu, as shown in Figure 11-21.

![Figure 11-21 Importing a Web service description (WSDL)](image)

2. In the Import Web Service Definitions dialog, click *File System* to import the WSDL. Navigate to the directory where you downloaded the additional materials and select *GetCustomer.wsdl*. Refer to Figure 11-22 on page 326. Click *Next*.
Important: We assume that you have deployed a GetCustomer service to a WebSphere Application Server. The GetCustomer service performs a database lookup of customer information based on a supplied customer number. The table is EOTCUST. The additional materials that you downloaded (refer to Appendix B, “Additional material” on page 485) contain the Data Definition Language (DDL) to create this table and a sample set of data that you load into the table.

Refer to Chapter 7, “Exposing DB2 data as a service” on page 159 for instructions to generate a Web service to access DB2 data from WebSphere Application Server. Make sure that you retrieve the columns in the following order: CUST_ADDR1, CUST_CITY, CUST_NO, DEPT, CUST_FN, and CUST_LN. The resulting SQL needs to look similar to:

```
SELECT CUST_ADDR1, CUST_CITY, CUST_NO, DEPT, CUST_FN, CUST_LN
FROM ITSO0C.EOTCUST
WHERE CUST_NO = :CUST_NO
```

3. In the Select operations dialog, as shown in Figure 11-23 on page 327, note that there is only one operation, getCustomer, defined in the WSDL, and thus, this operation is already selected. Click Finish.
4. Depending on the wsdl file that is imported, you might receive an error message, as shown in Figure 11-24 on page 328, indicating that messages were generated while importing the WSDL. Click Details for that dialog and note that the warning is related to the use of the COBOL reserved words, such as address as an element name, and that the affected elements have been changed, for example, from address to xaddress.
Click **OK** to dismiss the error dialog.

5. In the EST Project Explorer view, expand the **CATAL.OutboundWebService** subproject and its folders.

Notice the additional artifacts shown in Figure 11-25 on page 329 that were created from importing the **GetCustomer.wsdl** file.

The Operations folder contains a single operation, `getCustomer`, which will be used to associate the abstract invoke node `GetCustomerInfo` with an actual outbound Web Service invocation.

The Message folder contains a single message file, `GetCustomerMessages.mxsd`, which is actually an XML schema definition of the input and output messages that were defined in the WSDL that you imported.

The Services folder contains two files: one file is the WSDL that you imported and the other file is a WSBind file that was generated from processing the WSDL file. If an equivalent WSBind file has not already been deployed to CICS TS, this file can be used. Recall that a WSBind file is a CICS runtime artifact that is required when dealing with Web Services. Because this file is for an outbound Web Service call, this WSBind file must be associated with a service requester pipeline in CICS as opposed to a service provider pipeline. We will come back to this topic in the deployment section of this sample.
Figure 11-25  OutboundWebService subproject and associated artifacts

Now that you have imported the WSDL for the customer lookup Web service, you will associate the getCustomer operation with the GetCustomerInfo node by selecting **getCustomer** under **Operations** from the EST Project Explorer view and dragging and dropping this operation onto the GetCustomerInfo invoke node in the flow editor. Notice that the icon for the GetCustomerInfo invoke node changed from  to  .

6. Click anywhere in the flow editor to give it focus. Press Ctrl+s to save your changes.

### 11.4 Defining the OrderItem flow step

We now define the details of the OrderItem flow step for this segment of the sample. The details involve the terminal interactions needed with the EGUI application in order to place an order. We record those terminal interactions as a terminal subflow.

To facilitate the recording of the terminal interactions and the creation of the terminal subflow, we import the BMS map definitions corresponding to the windows with which we will interact.

This segment assumes that you access a CICS system that has the sample Catalog Manager application installed. This sample was created using the CITSO14 region running on z1.
11.4.1 Importing BMS

Import the BMS map definitions for the EGUI application:

1. In the EST Project Explorer view, select the CATAL.Terminal subproject, right-click it, and select Import → BMS from the context menu.

2. In the Import BMS dialog, click File System and navigate to the directory where you downloaded the additional materials (refer to Appendix B, “Additional material” on page 485). Select both dfh0xs1.bms and dfh0xs2.bms. Refer to Figure 11-26.

3. Click Finish.

Three messages are created now, as shown in Figure 11-27 on page 331. Two messages are associated with map definitions (EXMENU and EXORDR) in mapset DFH0XS1, and one message is associated with one map definition (EXINQC) in mapset DFH0XS2.
4. You can hover your cursor over a message and get a rendering of the screen view, as shown in Figure 11-28.

![Figure 11-28 Hovering over a message to get a view of the screen](image)

11.4.2 Defining a host connection

Define a host connection with the z/OS system:

1. Select the **CATAL.Terminal** subproject, right-click it, and select **New → Host Connection**.

2. In the Specify Destination dialog, specify a file name that reflects the z/OS system, for example, **AD08.host**, and click **Next**, as shown in Figure 11-29 on page 332.
3. On the Basic Settings dialog, specify the host name or IP address of your z/OS system. Modify the other settings as needed in order to connect to your z/OS system. Refer to Figure 11-30 on page 333.
4. Click **Finish**.

You see the usual application logon or selection window for your system.

### 11.4.3 Recording 3270 terminal screens

In the following steps, we record a Terminal Flow. The *Terminal Flow recording* is used to define the dependencies between screens and define all input and output fields. Eventually, you can map these fields.

Before starting to record the Terminal Flow, we need to get to the point, that is, the screen, where we want to start the recording. At a minimum, you need to log on to CICS, but most likely, you will need to advance to the main menu of the application. In our case, we need to first get to the main menu of the Catalog Manager Order application.

![Specifying Host name for New Host Connection](image)
Log on to CICS:

1. Select the CICS region that you will use for this sample from your application selection screen. Navigate until you have signed on to your target CICS system to the point where you can enter transactions. For this sample, you need to enter the EGUI transaction. Note that the Enter key is the right Ctrl key.

**Tip:** There is a slide out drawer built into the host editor that can be opened to reveal a keypad for all the automatic initiate descriptor (AID) keys typically found on terminal emulators.

An example of an application selection screen is shown in Figure 11-31.

![Application selection screen](image)

**Figure 11-31  Application selection screen**

2. Start the EGUI application by entering *EGUI* on the screen.

You see the main menu of the EGUI application (Figure 11-32 on page 335). Note that the information area of the host editor indicates that it recognized the screen from the BMS maps that you imported earlier.
We are now ready to start recording the terminal interactions. We will use the Host Editor to record the terminal interactions. The Host Editor has a toolbar, an information area, and a terminal emulation area.

**Important:** Be extremely careful with the following steps. You might want to read through the steps describing the recording process one time before actually performing these steps. Proceed slowly. If you do make a mistake, you might have to restart this process.

The steps to start recording the terminal interactions are:

1. On the Host Editor Toolbar, click the **Start Recording Flow** icon, as shown in Figure 11-33.

2. In the Record a terminal flow dialog, take the default names for the flow name and screen operations file name, and click **Next**, as shown in Figure 11-34 on page 336.

A flow file will be created that contains a graphical representation of the sequence of terminal interactions, as well as the data input and output from both the flow itself and the screen invoke nodes that represent these terminal interactions.
A screen operations file will also be created that contains the screen operations captured during the recording of a sequence of terminal interactions. A **screen operation** is the definition of an input screen and its association with possible output (next) screens based on an interaction (input data and AID key combination) with the input screen.

![New Flow Recording - Dfh0xs1_Exmenu_Exmenu](image)

*Figure 11-34  Specifying name of the flow and screen operations*

3. In the next window, the Associate operation dialog, click **Create New**, as shown in Figure 11-35 on page 337. Make sure that **Update interface operation messages while building flow** is selected. By selecting this option, you choose to have the fields, which are used for data input during flow recording of the terminal interactions, automatically included as input fields for the flow itself. If you elect not to select this option, the fields to be used as input and output for the flow itself need to be defined separately from the recording of the terminal interactions.
4. In the Create Operations File pane, as shown in Figure 11-36 on page 338, take the default names for the Interface Operations file and Operation. An operation is associated with input and output messages. Click New to create one message to associate with this operation. Then, click New again. You must click New twice to create messages for both the input and output.
5. In the Message File and Message Name dialog, take the default message file
name and message name, as shown in Figure 11-37 on page 339. Click OK.
6. Repeat the process to create an output message file. After you finish, the Operations File dialog looks like what is shown in Figure 11-38 on page 340.
7. Click **OK** to apply the changes and dismiss the Operations File dialog. This action brings the focus back on the Associate operation dialog, which shows the Interface Operations file and Operation entry fields filled, as shown in Figure 11-39 on page 341. Click **Next**.
8. On the Select a variable message dialog, click **Create New** to create a new variable message, as shown in Figure 11-40 on page 342.
9. On the Message File and Message Name dialog, take the defaults and click **OK**, as shown in Figure 11-41.
10. Back on the Select a variable message dialog, you see a window similar to Figure 11-42. Click Finish.

![Figure 11-42 Defined variable message](image)

You are now ready to begin the actual recording of the terminal interactions. This recording involves entering input that is captured directly, specifying which fields will have data inserted into them and which fields you will be extracting data from.

**Important:** Be careful with the following steps. Read them slowly to prevent incorrect actions. If you make a mistake, you must start this process again.

To capture data, just enter it as you normally interact with the application:

- To specify that a field will have data inserted into it, you must click the **Insert Data into Screen** icon ( ) on the toolbar first and then select the actual entry field followed by the entry of representative data. This data becomes variable at run time, which differs from data that is inserted into a screen field without clicking the **Insert Data into Screen** icon, which
becomes static hard-coded data used at run time. For example, in the next step, you enter 2, which will become static hard-coded data that will always be used as the value for the Action screen field, while the value 0040 that you specify in subsequent steps is representative data that becomes a variable at run time.

- To specify that data will be extracted from a field, you must click the **Extract Data from Screen** icon ( ) on the toolbar first and then select the actual entry field containing the data to be extracted.

**Important:** *In the following steps, do not press Enter (right Ctrl) unless explicitly mentioned.* Pressing Enter (right Ctrl) at the wrong moment or too early creates errors in your recorded flow, which can only be fixed by recording the flow again from the beginning.

11. Type 2 in the Action field, as shown in Figure 11-43.

12. Click the **Insert Data into Screen** icon on the toolbar, as shown in Figure 11-43.

13. Click the **Order Item Number** entry field *(not the text field).* You see the entry field outlined (Figure 11-44 on page 345). Make sure that the cursor is at the start of the entry field.
14. Type 0040 in the Order Item Number entry field, as shown in Figure 11-45.

15. Press Enter (Ctrl).

After you press Enter, you are brought to the Order details screen as shown in Figure 11-46 on page 346. On this screen, you need to first extract the item description and then the item cost. After specifying the extract, you then need to insert data into the Order Quantity, User Name, and Charge Dept fields.
16. Click the **Extract Data from Screen** icon on the toolbar, as shown in Figure 11-46.

![Figure 11-46: Extracting data from the screen](image)

17. Click the description for item 0040, **Ball Pens Green 24pk**. You see this read-only field framed with brackets (Figure 11-47).

![Figure 11-47: Selecting item description for extraction](image)

18. Repeat the process (steps 16 and 17) to extract the item cost.
After you have extracted the data for the description and the item cost fields, you need to insert data into the three input fields of this screen. Follow the process for inserting data (click the **Insert Data into Screen** icon on the toolbar, select the input field, and enter the representative data) for each of the three input fields. So, you need to repeat the same steps three times. Use Figure 11-48 as a guide for what data to enter.

**Important:** A common error is only clicking the Insert Data into Screen icon one time and then entering three input fields at a time. *If you make this error, you will later discover that SFM cannot perform a proper input field mapping.*

To avoid this error, make sure that you perform all three steps for each field separately: the first time for Order Quantity with a value of 001, the second time for User Name with, for example, a value of wilbert, and the third time for Charge Dept with, for example, a value of mydept.

![Figure 11-48 Specifying Extract and Insert information](image)

After you have finished your extract and insert specifications, press Enter (Ctrl), which brings you back to the main menu with a status message indicating that the order was successfully placed.
19. You must now extract the status message. Follow steps 16 on page 346 and 17 on page 346 to extract the message but this time for the status message. Refer to Figure 11-49.

![Figure 11-49 Extracting the status message](image)

After you have specified the extraction of the status message, you are finished with the recording of the terminal interactions.

20. Click the **Stop Recording Flow** icon on the toolbar, as shown in Figure 11-50 on page 349.
21. After you have stopped the recording, you need to save the flow by clicking the **Save Flow** icon on the toolbar, as shown in Figure 11-51.

![Figure 11-51  Saving recorded flow](image)

Note that an additional flow, `Dfh0xs1_Exmenu_Exmenu.seqflow`, was created in the Flows folder. This flow represents the terminal interactions and will be considered as a subflow to the main flow, `CATAL.seqflow`. Refer to Figure 11-52.

![Figure 11-52  Flows associated with the CATAL Service Flow project](image)

Let us examine the flow that we just recorded.
22. Double-click **Dfh0xs1_Exmenu_Exmenu.seqflow**. It will open the subflow. You will see a window similar to Figure 11-53.

![Figure 11-53](image)

**Figure 11-53  Dfh0xs1_Exmenu_Exmenu.seqflow window**

**Tip:** If your flow is not laid out properly, click anywhere in the flow editor. Right-click and select **Layout → Left to Right** from the context menu.

23. Double-click **variableAssignInput**. You need to verify the Source fields using Figure 11-54.

![Figure 11-54](image)

**Figure 11-54  VariableAssignInput Source window**

**Note:** The Terminal flow that we recorded and saved can also be loaded into the host editor using the Load Flow icon ( ) and then played back using the Play Flow icon ( ) to test the recorded terminal interactions with various input data and to verify the resulting output data.
You only see exactly the following fields:

- ITEM-REF
- ORDR-USERID
- ORDR-DEPT
- ORDR-QUANTITY

24. Close this window and double-click **variableAssignOutput**. You need to verify the Target fields using Figure 11-55.

![Figure 11-55  VariableAssignOutput Target window](image)

You only see exactly the following fields:

- ORDR-DESC
- ORDR-COST
- MSG1

**Important:** If any of the source fields or target fields lists differ from what is shown in Figure 11-54 and Figure 11-55, you might need to go back to the beginning of this section and rerecord the terminal flow. An alternative is to use the mapping editor (as discussed in 11.5, “Mapping the Data Flow” on page 352) to ensure that the correct source mapping inputs and target mapping outputs are used.

25. Close this window.

26. Close **Dfh0xs1_Exmenu_Exmenu.seqflow**.

27. Save this flow if prompted.

Next, you associate the OrderItem invoke node in the main flow with the created subflow.

28. Make sure that the main flow, **CATAL.seqflow**, is opened in the flow editor. Select **Dfh0xs1_Exmenu_Exmenu.seqflow** from the Flows folder in the EST Project Explorer view and drag and drop it onto the OrderItem invoke node of the main flow in the flow editor. You will see the icon of the invoke node change from: 

![icon change](image)
Click anywhere in the flow editor so it has focus. Press Ctrl+s to save your changes. Your main flow now looks like Figure 11-56.

![Image](92x509 to 485x545)

**Figure 11-56** CATAL main flow after defining GetCustomerInfo and OrderItem

The next step is to specify how data will flow through the application.

## 11.5 Mapping the Data Flow

Before we proceed further in this segment, let us review the scenario (refer to Figure 11-56 for reference).

A Web service request will come in with an input request that consists of a customer number, an item number, and the quantity to be ordered. The Web service requester will expect an output response that consists of a description of the item ordered, the total cost of the order, the status of the order, and the name associated with the customer number for this order.

The input request will come into the flow application through the `Receive` node labeled `inProcessOrderRequestMsg`. The output response will be sent from the flow application through the `Reply` node labeled `outProcessOrderResponseMsg`. The two invoke nodes of the flow application expect different input data and will emit different output data. You need to specify how data will flow from one node to another node and also what kind of transformation might need to be performed during the flow (for example, character string data might need to be converted to integer format). You accomplish this transformation by using the mapping capabilities of the Service Flow tools.

Note that in the flow there are two nodes labeled `variableAssignInput` and `variableAssignOutput`. These are `Assign` nodes. An Assign node is associated with mapping routines that map or transform data as it flows from one node to another node. You will specify what those mapping routines look like. In addition, you will also create an Assign node between the two invoke nodes.
11.5.1 variableAssignInput node

Perform the following steps to specify the transformation for the variableAssignInput node:

1. In the flow editor, double-click the variableAssignInput node, which launches the mapping editor. Note that there is an Outline view associated with the mapping editor.

   **Important:** Make sure that variableAssignInput is the node that has the focus in the Outline view, because multiple mapping routines are stored in one mapping file.

2. In the Outline view, click variableAssignInput, as shown on Figure 11-57 on page 354. Note that the mapping editor has two sections called Source and Target.
3. In the Source section of the mapping editor, click the + (addition symbol) to the left of inProcessOrderRequest, which will show you the elements associated with it: accountNumber, itemNumber, and quantity. We will associate accountNumber with the input to getCustomer.

Click anywhere in the Target section of the mapping editor to give it focus. Right-click and select Add Message Mapping Output from the pop-up menu.

4. On the Select a Message dialog, expand CATAL → CATAL.OutboundWebService → Messages and select
GetCustomerMessages.mxsd → getCustomerSoapInputMsg. Click OK. Refer to Figure 11-58.

![Select a Message](image1)

Figure 11-58  Selecting Target message for variableAssignInput

When you expand getCustomer, the mapping editor looks like Figure 11-59.

![Source and Target mappings for variableAssignInput](image2)

Figure 11-59  Source and Target mappings for variableAssignInput

5. Select accountNumber from the Source section. There is a Properties view at the bottom portion of RDz. As you scroll through the properties for accountNumber, note that it is defined as type string. Refer to Figure 11-60.

![Properties for a mapping element](image3)

Figure 11-60  Properties for a mapping element
6. Select **CUST_NO** from the Target section. Note that the Properties view shows it defined as type `int`, which tells us that transformation is needed to associate `accountNumber` with `CUST_NO`.

7. To associate (or map) `accountNumber` with `CUST_NO`, select `accountNumber` it from the Source section, then drag and drop it onto `CUST_NO` in the Target section.

Notice the additional arrow decorators beside the elements to indicate that a mapping exists. Note also the Target value in the Overview section of the mapping editor. You need to modify that target value so that the correct type of `int` will be assigned.

![Figure 11-61  Mapped elements for variableAssignInput](image)

8. Click twice on the target value (highlighted) for `CUST_NO` in the Overview section of Figure 11-61. Notice the `...` (ellipsis) button that appears at the end of the target value. Click this button to launch the Compose Mapping Expression editor. Open the Compose Mapping Expression editor by clicking `...`, as shown in Figure 11-62.

![Figure 11-62  Editing the mapping](image)

Note that the target value for `CUST_NO` is a straight assignment from `accountNumber`. From the previous discussion, you know that there is a type mismatch between those two elements. With the Compose Mapping Expression editor, you can apply a set of functions to source values to perform data transformation. There is a miscellaneous function called `CAST` that can be used to force a `string` type into an `int` type.
9. In the Value of target field area, modify
   `s_inProcessOrderRequestMsg.inProcessOrderRequest.accountNumber` so
   that it now reads:
   
   ```sql
   CAST((s_inProcessOrderRequestMsg.inProcessOrderRequest.accountNumber
   ) AS INTEGER)
   ```

   Refer to Figure 11-64 on page 358.
10. Click OK to dismiss the editor and apply the changes.
11. Press Ctrl+s to save your changes.

11.5.2 variableAssignOutput Node

Perform the following steps to specify the transformation for the variableAssignOutput node:

1. From the Outline view, select variableAssignOutput to specify its mapping routine. Notice that the Source section is empty while the Target section shows outProcessOrderResponseMsg (highlighted), as shown in Figure 11-65.

2. Click anywhere in the Source section to give it focus. Right-click and select Add Message Mapping Input from the context-sensitive menu.

3. You will select three sources for your input. The input request message contains the order quantity that you will need to compute the total cost for the order. The response message from the outbound Web service call to the customer lookup service contains the first and last names associated with the customer number that was given. Lastly, the response from invoking the EGUI application contains the item description, the unit cost for the item, and the status of the order.
4. On the **Select a Message** dialog, expand **CATAL → CATAL.Interface → Messages** and select **processOrderMessages.mxsd → inProcessOrderRequestMsg**, as shown in Figure 11-66 and click **OK**.

![Figure 11-66   Selecting inProcessOrderRequestMsg as one input for variableAssignOutput](image)

5. Repeat this process of adding a message mapping input *two* more times:

   - For the second time, expand **CATAL → CATAL.OutboundWebService → Messages** and select **GetCustomerMessages.mxsd → getCustomerSoapOutputMsg**.

   - For the third time, expand **CATAL → CATAL.Interface → Messages** and select **o_Dfh0xs1_Exmenu_Exmenu.msxd → o_Dfh0xs1_Exmenu_Exmenu**.

You see a window similar to what is shown in Figure 11-67 on page 360 after adding the three message mapping inputs.
6. Next, you use the information in Figure 11-67 and Table 11-3 as a guide to map between source and target elements. Use drag and drop (click the Source element, hold the left mouse button, drag the variable to the Target element, and release the left mouse button) to perform the mapping of all five variables. We will edit the mappings after the initial mapping is done.

Table 11-3  Mappings for the variableAssignOutput node

<table>
<thead>
<tr>
<th>Source element</th>
<th>Target element</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDR-DESC</td>
<td>itemDescription</td>
</tr>
<tr>
<td>ORDR-COST</td>
<td>totalCost</td>
</tr>
<tr>
<td>MSG1</td>
<td>orderStatus</td>
</tr>
<tr>
<td>CUST_FN</td>
<td>firstName</td>
</tr>
<tr>
<td>CUST_LN</td>
<td>lastName</td>
</tr>
</tbody>
</table>

We will now further refine the target values for totalCost, firstName, and lastName.

For totalCost, we compute the value as the product of the item unit cost (ORDR-COST) and order quantity (quantity). Because both of these elements are defined as type string, we need to convert these element types to type int before they can be multiplied. We use the CAST function that we used in the previous mapping.
For the `firstName` element, the response from the Web service call actually has this value starting at offset 2. For the `lastName` element, the value starts at offset 3. We use the string function SUBSTRING to extract 20 characters starting at the appropriate offsets for those elements.

**Note:** The Service Flow Modeler tool in RDz does not currently support nillable attributes in WSDL files. To compensate, we are using the substring function to ignore the byte positions that indicate whether an attribute is nillable or not. This support will be addressed in a future release of RDz. At the time of writing this book, RDz V7.5 has still not addressed this situation.

Figure 11-68 shows what the mapping routine looks like after performing the initial mapping.

![Initial Mapping routine for variableAssignOutput](image)

Figure 11-68  Initial Mapping routine for variableAssignOutput

7. In the Overview section of the mapping editor, select `totalCost`, right-click, and select **Edit Mapping** from the context-sensitive menu. In the Value of target field area, replace `s_o_Dfh0xs1_Exmenu_Exmenu."ORDR-COST"` with `CAST((s_o_Dfh0xs1_Exmenu_Exmenu."ORDR-COST") AS FLOAT) *`
CAST((s_inProcessOrderRequestMsg.inProcessOrderRequest.quantity) AS INTEGER).

Click OK to apply the changes.

8. Select the target element **firstName** from the Overview section, right-click, and select **Edit Mapping** from the context-sensitive menu. Replace s_getCustomerSoapOutputMsg.getCustomerResponse.CUST_FN with SUBSTRING((s_getCustomerSoapOutputMsg.getCustomerResponse.CUST_FN) FROM 2 FOR 20)

Click OK to apply the changes.

9. Select the target element **lastName** from the Overview section, right-click, and select **Edit Mapping** from the context-sensitive menu. Replace s_getCustomerSoapOutputMsg.getCustomerResponse.CUST_LN with SUBSTRING((s_getCustomerSoapOutputMsg.getCustomerResponse.CUST_LN) FROM 3 FOR 20)

Click OK to apply the changes.

The target values in the Overview section now look as shown in Figure 11-69.

![Figure 11-69 Target values after having customized mappings](image)

10. Press Ctrl+s to save the changes.

11. Close the **CATAL.seqmap**.

### 11.5.3 Assign node

You must map the flow from the getCustomer node to the Dfh0xs1_Exmenu_Exmenu node:

1. A quick way to create the mapping is to select the connector on the **CATAL.seqflow** between those two nodes, right-click it, and select **Initialize Mapping** from the context-sensitive menu. Complete this action now, which brings up a dialog, as shown in Figure 11-70 on page 363.
2. Check `inProcessOrderRequestMsg` in the Source column, and click **OK**.

   The CATAL.seqmap now looks like what is shown in Figure 11-71 on page 364.
3. Use Table 11-4 as a guide for your initial mapping by dragging and dropping from the source element to the related target element.

Table 11-4  Mappings

<table>
<thead>
<tr>
<th>Source element</th>
<th>Target element</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemNumber</td>
<td>ITEM-REF</td>
</tr>
<tr>
<td>quantity</td>
<td>ORDR-QUANTITY</td>
</tr>
<tr>
<td>dept</td>
<td>ORDR-DEPT</td>
</tr>
<tr>
<td>CUST_LN</td>
<td>ORDR-USERID</td>
</tr>
</tbody>
</table>

The CATAL.seqmap now looks as shown in Figure 11-72 on page 365.
We will further refine the target values for ORDR-DEPT and ORDR-USERID.

We will use a substring of CUST_LN as the target value for ORDR-USERID. As discussed previously, the value of the CUST_LN source element starts at offset 3. Because ORDR-USERID is eight characters in length, we will use the SUBSTRING function to extract eight characters starting from offset 3.

4. In the Overview section of the mapping editor, select ORDR-DEPT, right-click it, and select **Edit Mapping**. In the Value of target field area, replace `s_getCustomerSoapOutputMsg.getCustomerResponse.DEPT` with `SUBSTRING((s_getCustomerSoapOutputMsg.getCustomerResponse.DEPT) FROM 1 FOR 8)`.

5. Click **OK** to apply the changes.

6. In the Overview section of the mapping editor, select ORDR-USERID, right-click it, and select **Edit Mapping**. In the Value of target field area, replace `s_getCustomerSoapOutputMsg.getCustomerResponse.CUST_LN` with `SUBSTRING((s_getCustomerSoapOutputMsg.getCustomerResponse.CUST_LN) FROM 3 FOR 8)`.

7. Click **OK** to apply the changes.

8. Press Ctrl+s to save your changes.

The mapping for the Assign node Assign looks like Figure 11-73 on page 366.
9. Close the mapping editor and select the flow editor. Because there were changes made to the flow, press Ctrl+s to save your changes.

Your updated main flow needs to look like Figure 11-74.

![Updated CATAL main flow](image)

10. Click **File → Save all**.

### 11.6 Generating the runtime artifacts

After modeling the Service Flow application, the next step is to generate the runtime artifacts that are needed to build, deploy, and run it. You generate the runtime artifacts through the specification of generation properties. A generation properties file is associated with a Service Flow.

You have created two flows: a main flow for the Catalog Order (CATAL) and a subflow that represents the terminal interactions with the EGUI application. Therefore, you will need to create two generation properties files to govern the generation of the runtime artifacts for those components of the Service Flow application:

1. In the EST Project Explorer view, select the **CATAL.seqflow** file, right-click it, and select **New Generation Properties file**.

   On the Generation Properties - New File dialog, specify a File name of **CATAL_generation_props.wsdl**, as shown in Figure 11-75 on page 367, and click **Next**.
2. On the Generation Properties - Runtime selection dialog, select **CICS Service Flow Runtime for CICS TS 3.2**, and click **Finish** as shown in Figure 11-76.

Figure 11-76   Selecting a target run time for CATAL generation
3. The Generation Properties editor (Figure 11-77 on page 369) opens with Generate Runtime Code disabled, because there are additional items that must be specified:

   a. Make sure that the following values are specified:
      
      • For the Request Name and Transaction ID, use IT01.
      • Specify a Program Name of CATAL.
      • Select **Generate Internal Data Structures**.

   b. Select **Web Services for CICS** as the specification for Generate Web Service Files. You need this selection, because you will want to access the Service Flow application via a Web service interface.

   c. Specify an End Point URI that maps to your CICS system, typically of the form: http://hostname:tcpipservice_port/local_URI, where hostname resolves to the z/OS system where a CICS TS region is running, tcpipservice_port is the port number specified by a TCPIPSERVICE resource on that CICS region, and local_URI is the resource path that will be used in an URIMAP resource.

      For example, we used http://wtscz1.itso.ibm.com:3014/itso/CATAL.

   d. Make sure that the WSBind and WSDL file names are CATAL in uppercase.

   e. Specify a Uniform Symbol Specification directory on your target z/OS system for WSDL HFS File Path, for example, /u/itso14/itso/wsd1.

      Refer to Figure 11-77 on page 369.
4. In the Outline view (left lower corner), click **getCustomer** to specify the generation properties for the Outbound Web service invoke node. Refer to Figure 11-78 on page 370 for details:

- Specify a Uniform Symbol Specification directory on your z/OS system for the Remote WSBind file location. This directory is a pickup directory associated with a service requester pipeline in your CICS region. For example, we used /u/itso14/itso/wspickup/requester.
- Specify a Uniform Symbol Specification directory for Remote WSDL file location, for example, /u/itso14/wsd1.
5. In the Outline view, click **Dfh0xs1_Exmenu_Exmenu (invalid)** to specify the generation properties for the Terminal Invoke node. Use Figure 11-79 on page 371 as a guide:

   - Specify a flow type of **Link3270 Bridge**.
   - Specify a Request Name and Transaction ID of **IT11**.
   - Specify a Program Name of **ORDIT**.
   - Select **Generate Internal Data Structures**.
   - The Initial PFKey setting must be **CLEAR**, and the Startup Transaction Data must be **EGUI**, which is the transaction ID that will initiate the sample catalog manager application.
6. Press Ctrl+s to save your changes. Note that Generate Runtime Code is now enabled.

Currently, when you generate the runtime code for a main flow, it does not generate the runtime code for the associated subflows, which means that you will have to re-specify the settings for the terminal subflow.


8. In the EST Project Explorer view, select CATAL → Flows → Dfh0xs1_Exmenu_Exmenu.seqflow file, right-click it, and select New Generation Properties file.
9. On the Generation Properties - New File dialog, specify a File name of OrderItem_generation_props.wsdl. Select **Reuse properties from most recently saved generation properties files**. Click **Next**.

10. Specify **CICS Service Flow Runtime for CICSTS 3.2** as the target run time, and click **Finish**. Your OrderItem Generation Properties pane looks like Figure 11-80.

![Figure 11-80 Specifying OrderItem generation properties](image)

11. Press Ctrl+s to save your changes. Note that Generate Runtime Code is now enabled.

12. Close the Generation Properties editor for all open windows.
You are now ready to generate the runtime artifacts that are needed to deploy a Service Flow application. These artifacts include COBOL programs, copybooks (not generated if “Generate Internal Data Structures” is selected in the generation properties), the WSBind file, and the jobs that are needed to define resources and build the Service Flow application. You have the option of generating these artifacts directly to the target z/OS system to facilitate the build and deployment. This option will also ensure that CICS Web Services artifacts, such as the WSBind file, are copied correctly to the appropriate Uniform Symbol Specification directory.

When you created the generation properties files, these files were placed in a new Generation folder under the CATAL Service Flow project. There will be a CICS SFR 3.2 subfolder, because you specified the CICS Service Flow Runtime for CICS TS 3.2 as the target run time.

13. In the EST Project Explorer view, select the CATAL → Generation → CICS SFR 3.2 → CATAL → CATAL_generation_props.wsdl file, right-click it, and select Generate Runtime Code. Alternatively, if you had the generation properties file opened, you can click Generate Runtime Code.

14. On the Generation Properties - File selection dialog, specify your z/OS user ID as the Job Control user ID and Job Control account.

15. For the deployment libraries HLQ, specify your user ID. The generation wizard assumes that the data sets <HLQ>.USER.JCLLIB, <HLQ>.USER.SRCLIB, <HLQ>.USER.COPYLIB, and <HLQ>.USER.LINKLIB exist, because these data sets will be referenced in certain jobs that the generation wizard generates.

Also, select Have SFR manage resource definition creation, which lets the Service Flow run time install the required resource definitions (TRANSACTION, PROGRAM, and PROCESSTYPE) as it installs Service Flows. Refer to Figure 11-82 on page 374.
16. To simplify this sample, do not ask the generation wizard to deploy to a remote target location, so leave that box unselected. Click **Finish**.

**Tip:** Using the remote connectivity facilities of RDz greatly simplifies the task of building and deploying the required development and runtime artifacts to the target z/OS and CICS system. If the remote facilities are configured and activated on your site and you are connected to the target system, the Service Flow runtime generation wizard guides you through selecting the target destinations on the remote system.

You see a Runtime Code Generation Complete dialog indicating that the generation was successful. Click **OK** to dismiss the dialog.

If you click Details, you see the same output as shown in Figure 11-83 on page 375.
17. Repeat the generation process for the OrderItem generation properties file.

In the EST Project Explorer view, notice the additional artifacts generated for CATAL, as shown in Figure 11-84 on page 376.
Figure 11-84  Generated runtime artifacts for CATAL and OrderItem

Table 11-5 on page 377 describes what has been generated for the main flow (CATAL).
<table>
<thead>
<tr>
<th>Generated artifact</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#IT01.jcl</td>
<td>This job defines the appropriate PROGRAM, TRANSACTION, and PROCESSTYPE resources to the target CICS system. You might want to modify the name of the resource group. If you had checked the box to have SFR manage resource definitions, this job might be skipped.</td>
</tr>
<tr>
<td>CATAL.cbl</td>
<td>This COBOL program represents the flow application (navigator, adapter).</td>
</tr>
<tr>
<td>CATAL.jcl</td>
<td>This Job compiles and link-edits CATAL into a load module.</td>
</tr>
<tr>
<td>CATAL.wsbind</td>
<td>This CICS Web service WSBind file describes the Web service interface to CATAL. It must be deployed to a pickup directory associated with a service provider pipeline.</td>
</tr>
<tr>
<td>CATAL.wsdl</td>
<td>This WSDL describes the CICS-based Web service. This WSDL is typically used to generate the client application that will access the flow application via the Web service interface. You will use this WSDL to test the flow using the built-in Web Services Explorer in RDz.</td>
</tr>
<tr>
<td>IT01.sfp</td>
<td>This file is the Service Flow properties file for the CATAL flow application. This file describes the modeled Service Flow and contains the definition of the server adapters, programs, and transactions that are required for the Service Flow to run. This file must be deployed to a Service Flow deployment directory on the Uniform Symbol Specification hierarchical file system (HFS) of the target z/OS system.</td>
</tr>
</tbody>
</table>

On successful generation of the subflow, you get the following four additional artifacts (Table 11-6 on page 378).
Table 11-6  Generated components for the subflow

<table>
<thead>
<tr>
<th>Generated artifact</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#IT11.jcl</td>
<td>This job defines the appropriate PROGRAM, TRANSACTION, and PROCESSTYPE resources to the target CICS system for the OrderItem terminal subflow.</td>
</tr>
<tr>
<td>IT11.sfp</td>
<td>This file is the Service Flow properties file for the ORDIT subflow application.</td>
</tr>
<tr>
<td>ORDIT.cbl</td>
<td>This COBOL program represents the terminal subflow (Link3270 Bridge adapter) that interacts with the EGUI application.</td>
</tr>
<tr>
<td>ORDIT.jcl</td>
<td>This job compiles and link-edits ORDIT into a load module.</td>
</tr>
</tbody>
</table>

Note that there are no WSBind and WSDL files generated, because you did not specify to generate Web Service files, because this subflow application will be driven from the main flow.

18. You can browse the generated artifacts so you are familiar with them.


The next logical tasks, which we will discuss, are to modify the jobs, submit the jobs, copy the WSBind file to a CICS Web Services pickup directory, copy the sfp files to the Service Flow deployment directory, activate the newly defined CICS resources, and perform a CICS pipeline scan to pick up the new WEBSERVICE and URIMAP resources that are associated with CATAL. After you complete these actions, you can test the flow.

**Tip:** If you have a remote system definition for your z/OS system in the Remote Systems view, you can customize the Enterprise Service Tools perspective to include the Remote Systems view. From that view, you can connect to the host and enable the remote deployment facility when you generate the runtime artifacts. Being connected to your target z/OS system also facilitates the submission of the jobs to define and build the required CICS runtime resources. Also, note that RDz has host emulation support that enables you to connect to a target CICS system and perform tasks, such as activating (installing) resources via CEDA and performing PIPELINE scans. You might also want to investigate the use of the Application Deployment Manager (ADM), which is a new facility in RDz Version 7.
11.7 Deploying generated artifacts

We will discuss the tasks that you need to accomplish to build and deploy the generated artifacts to the target z/OS system at a high level. You can either use RDz or a combination of ftp and TSO/ISPF to perform most of these tasks:

1. Move or copy the COBOL files (CATAL.cbl and ORDIT.cbl) to a COBOL source data set on z/OS.

2. Move or copy the JCL files (#IT01.jcl, CATAL.jcl, #IT11.jcl, and ORDIT.jcl) to a JCL source data set on z/OS.

3. Move or copy the WSBind file (CATAL.wsbind) to a Uniform Symbol Specification pickup directory on z/OS that is associated with a CICS service provider pipeline. Be sure that the copy is done in binary mode.

4. Move or copy the Service Flow properties files (IT01.sfp and IT11.sfp) to a Uniform Symbol Specification deployment directory on z/OS that is associated with the Service Flow run time.

5. Move or copy the service requester WSBind file (at CATAL.OutboundWebService → Services → GetCustomer.wsbind) to a Uniform Symbol Specification pickup directory on z/OS that is associated with a CICS service requester pipeline. Be sure that the copy is done in binary mode.

6. Modify the compile and link-edit jobs for CATAL and ORDIT as appropriate and run these jobs. Copy the corresponding load modules to a load library that is part of the CICS DFHRPL concatenation.

7. Optionally, modify and submit the resource definition jobs (#IT01.jcl and #IT11.jcl). Be sure to add the resource group to your CICS startup resource list. Note that while SFR can manage the resource definitions, these definitions are not persisted in the CICS system definition data set (CSD).

8. Perform pipeline scans (CEMT PERFORM PIPELINE ... SCAN) to pick up the WSBind files that you just copied and to create the associated WEBSERVICE and URIMAP resources to CICS.

Assumption: The appropriate TCPIPSERVICE and PIPELINE (both service provider and requester) resources have been defined and installed.

9. Install the Service Flows that you just deployed using the CMAN transaction.

10. Test the Service Flow application by using the Web Services Explorer in RDz.
11.8 Testing the Service Flow application

If you have successfully completed all the previous steps, you have reached the point to test the application. We assume that you have:

- Installed the Service Flows
- Installed the new resources (TRANSACTION, PROCESSTYPE, and PROGRAM)
- Activated the new WEBSERVICE resource by scanning the appropriate PIPELINEs
- Deployed the Service Flow applications to a load library that is part of the CICS DFHRPL concatenation

You are ready to test this new CICS-based service.

For this segment, you are interacting with the CICS TS region on z/OS:

1. In the Enterprise Service Tools perspective in the EST Project Explorer view, select CATAL.wsdl, right-click it, and select **Web Services → Test with Web Services Explorer**, which launches the Web Services Explorer in a Web browser.

   You see a window similar to what is shown in Figure 11-85 on page 381.
2. In the Navigator pane, click the + (addition symbol) to the left of **DFHMADPLHTTPSSoapBinding**, which shows you an operation named DFHMADPLOperation.

3. Select **DFHMADPLOperation**. The Actions pane of the Web Services Explorer gives you entry fields for the input request message:
   a. Enter IT01 for the request name (dfhmahr_requestname).
   b. Enter 00000011 for the account number (you can try 1 through 12; remember to enter all eight digits).
   c. Enter 0050 for the item number (you can try 0010 through 0060; remember to enter four characters, which means include leading zeros).
   d. Enter 001 for the quantity (you have to enter three characters, which means include leading zeros).

4. Click **Go** to invoke the Web service request.
Figure 11-86 Invoking the CATAL Web Service request

Upon successful invocation, the Status pane of the Web Services Explorer shows you a window similar to Figure 11-87 on page 383.
11.9 Summary

You have seen how you can use the Enterprise Service Tools (EST) capabilities of Rational Developer for System z (RDz) Version 7 to build a new Service Flow application that aggregates the data and results from various applications to present a single interaction to a Web service client. The CICS Service Flow Runtime provides the infrastructure to enable the aggregation of these applications, which can include non-terminal COMMAREA applications, terminal based-applications, and outbound Web service invocations.

11.10 Additional material

We included the workspace with the developed application in the additional material. Refer to Appendix B, “Additional material” on page 485 for instructions.
Integrating an external Web service with a CICS application

In this sample, we demonstrate how a CICS application can be easily integrated with external Web services. The assumption is that we already have external Web Services hosted in an application server, an enterprise service bus (ESB), or other middleware and that existing CICS applications will benefit from reusing the functionality that is implemented in these external Web services.

We use Rational Developer for System z (RDz) to create the CICS Web service artifacts and deploy them on z/OS.

We follow a top-down approach where we use the Web Services Description Language (WSDL) file of the existing Web service to create a COBOL copybook and a CICS service requester program.

This chapter includes the following topics:
- Introduction to the sample
- Prerequisites
- Extracting the WSDL from the existing Web Service
- Generating the CICS artifacts
- Updating the generated CICS service requester program
- Creating a test CICS client program
- Deploying the application to z/OS
- Testing the CICS Service requester program
- Handling complex CICS COMMAREA structures
- Summary
12.1 Introduction to the sample

We have an existing Web service that when provided with an input for department number, it returns the department name and the manager number. The CICS application needs to get this department name and the manager number for its business process. In this sample, we lead you through the following steps:

- Extracting the WSDL from the existing Web service
- Using the WSDL to generate the CICS artifacts
- Updating the generated CICS service requester program
- Creating a test CICS client program
- Deploying on z/OS
- Testing the CICS service requester program
- Handling complex CICS COMMAREA structures

Note: In case you do not yet have an existing external Web service, refer to Chapter 14, “Using DB2 as a Web service requester” on page 455 to create a Web service, which is basically the same Web service as the Web Service that we use in this sample.

12.2 Prerequisites

This sample has the following prerequisites:

- Rational Developer for System z (RDz) V7.1
- CICS Transaction Server Version 3.1 or 3.2
- An external Web service

12.3 Extracting the WSDL from the existing Web Service

The starting point for the scenario used in this sample is an existing WSDL file. One of the techniques to obtain an existing WSDL file is to extract it from a URL of an external Web service. We extract it from Rational Developer for System z. We also test the WSDL using the Web Services Explorer in RDz to ensure that the Web service is accessible, and we verify the results of an invocation. Generally, a successful test with the RDz Web Services Explorer is evidence that
the Web service is in working order and that the WSDL is the correct WSDL for that specific Web service.

Perform the following steps to obtain and test the WSDL:
1. Launch Rational Developer for System z (RDz).
2. Complete the following steps:
   – Create a workspace.
   – Create a z/OS connection to your z/OS host system.
3. In the Enterprise Service Tools (EST) perspective, launch the Web Services Explorer, as shown in Figure 12-1.

4. The default window that gets launched after you click Web Service Explorer is the Universal Description, Discovery, and Integration (UDDI) Main window. You need to, however, launch the WSDL Main window. Click the WSDL page, as shown in Figure 12-2 on page 389.
5. Click **WSDL Main**, as shown in Figure 12-3.

![Figure 12-3 Web Services Explorer WSDL Main window](image)

6. The URL of our external Web service is:
   

   Copy this URL link in the WSDL URL box and suffix it with `?WSDL`, as shown in Figure 12-4 on page 390.
Note: Obviously, you will need to use your own specific host name and port number instead of the values that we used.

7. Click Go. The service that we will integrate is getDepartment. Refer to Figure 12-5 on page 391.
8. Click **getDepartment**. Refer to Figure 12-6 on page 392.
9. Enter A00 in the DEPTNO field, and click **Go**. The Web service Explorer invokes the service and displays the result of the invocation in its Status pane (refer to Figure 12-7 on page 393).
The next step will be to extract the WSDL file from the Web service, which will be used to generate the CICS artifacts in the forthcoming steps.

Click the URL link beneath WSDL Main in the Navigator pane. Refer to Figure 12-8 on page 394.
Scroll down the Other Actions pane and find the Import WSDL links. You can either import to the workbench directly or import to your local disk on your PC and then add the WSDL file to your project in the RDz workspace. Refer to Figure 12-9 on page 395.
You have extracted the WSDL file. We will use this WSDL file to generate CICS Web service artifacts in the following steps.

### 12.4 Generating the CICS artifacts

In the following steps, we create the artifacts that are needed for CICS:

1. In the EST perspective, create a new CICS Web Services project and import the WSDL file that you extracted in the previous steps (Figure 12-10 on page 396).
2. Right-click `EmployeeWebservice.wsdl`, and select **Generate CICS Web Services for CICS resources.** Refer to Figure 12-11 on page 397.
3. This action launches the Enterprise Service Tools wizard. In the Application mode drop-down list box, select **Service Requestor** (Figure 12-12).

4. Click **Start**. You see the Application Properties tab as is shown in Figure 12-13 on page 398.
5. Switch to the **Service Properties** tab and ensure that you select the Web service that you want to integrate, that is, **getDepartment**, in our example. Click **Next**. Refer to Figure 12-14 on page 399.

![Web Services for CICS wizard: Application Properties tab](image-url)
6. In the next pane, you can accept the defaults. Refer to Figure 12-15.
7. Click **Finish**.

**Note:** You might get an error if your WSDL contains a complex type sequence with maxOccurs or minOccurs attributes. RDz does not support WSDL if either maxOccurs or minOccurs is 0, because of a COBOL language limitation where array structures cannot be dynamic and must have a fixed number. This situation is not the case with XML data structures. You might get this error if you have this complex type in the WSDL file as shown in Example 12-1.

---

**Example 12-1  Employeewebservice.log**

Error Details:
--------------

com.ibm.cics.wsd1.CICSWSDLException: DFHPI9013E Schema model groups with maxOccurs or minOccurs not equal to 1 are not supported. Problem found for type: "getDepartmentResponse".

at com.ibm.cics.wsd1.ws2ls.OperationToICM.getICM(Unknown Source)
at com.ibm.cics.wsd1.ws2ls.ws2ls.doStuff(Unknown Source)
at com.ibm.cics.wsd1.ws2ls.ws2ls.run(Unknown Source)
at com.ibm.cics.wsd1.common.CICSWebServicesAssistant.callWSAssistant(Unknown Source)
at com.ibm.cics.wsd1.common.CICSWebServicesAssistant.DFHWS2LS(Unknown Source)
at com.ibm.etools.xmlent.batch.util.cwsa.CWSAOperation.run(Unknown Source)
at com.ibm.etools.xmlent.batch.cwsa.gen.WS2LSSetGen.runWebServicesAssistant(Unknown Source)
at com.ibm.etools.xmlent.batch.cwsa.gen.WS2LSSetGen.generate(Unknown Source)
at com.ibm.etools.xmlent.batch.processing.BPServiceImplementationWrapper.createInterpretiveImplementation(Unknown Source)
at com.ibm.etools.xmlent.batch.processing.BPServiceImplementationWrapper.processServiceImplementation(Unknown Source)
at com.ibm.etools.xmlent.batch.processing.BPProjectWrapper.process(Unknown Source)
at com.ibm.etools.xmlent.batch.processing.BatchProcess.run(Unknown Source)
at com.ibm.etools.xmlent.ui.operations.BatchProcessorWizardOperation.runBatchProcessor(Unknown Source)
at com.ibm.etools.xmlent.ui.operations.BatchProcessorWizardOperation.run(Unknown Source)
at org.eclipse.jface.operation.ModalContext$ModalContextThread.run(Unknown Source)
Caused by: com.ibm.cics.schema.ICMException: DFHPI9013E Schema model groups with maxOccurs or minOccurs not equal to 1 are not supported. Problem found for type: "getDepartmentResponse".

at com.ibm.cics.schema.impl.ICMImplFromWSDL.getEffectiveElement(Unknown Source)
at com.ibm.cics.schema.impl.ICMImplFromWSDL.getElementName(Unknown Source)
at com.ibm.cics.schema.impl.ICMImplFromWSDL.processAList(Unknown Source)
at com.ibm.cics.schema.impl.ICMImplFromWSDL.processParticle(Unknown Source)
at com.ibm.cics.schema.impl.ICMImplFromWSDL.processComplexType(Unknown Source)
at com.ibm.cics.schema.impl.ICMImplFromWSDL.processTypeDef(Unknown Source)
at com.ibm.cics.schema.impl.ICMImplFromWSDL.processElementDeclaration(Unknown Source)
at com.ibm.cics.schema.impl.ICMImplFromWSDL.processParticleIgnoreCardinality(Unknown Source)
at com.ibm.cics.schema.impl.ICMImplFromWSDL.processRepeatICMEntry(Unknown Source)
at com.ibm.cics.schema.impl.ICMImplFromWSDL.processAnArray(Unknown Source)
at com.ibm.cics.schema.impl.ICMImplFromWSDL.processParticle(Unknown Source)
at com.ibm.cics.wsdl.ws2ls.OperationToICM.processMessage(Unknown Source)

... 15 more

DFHPI9013E Schema model groups with maxOccurs or minOccurs not equal to 1 are not supported. Problem found for type: "getDepartmentResponse".

If this error occurs, you need to edit the WSDL file and update the MINOCCURS to 1 and also ensure that MAXOCCURS is 1. Also, find the word "unbounded" and change it to 1 as highlighted in Figure 12-16 on page 402 to ensure that only one elementary item is generated. In the case of COMMAREA arrays, just update both the fields with the number of OCCURS that you want in your COBOL copybook. Then, regenerate the CICS Web service artifacts. When you try to regenerate, you will get a warning message as shown in Figure 12-17 on page 402. Click OK, and then follow the steps as explained earlier in Figure 12-11 on page 397 to Figure 12-15 on page 399.
The following CICS Web service artifacts are generated (Figure 12-18 on page 403):

- Inbound COBOL copybook
- Outbound COBOL copybooks
- Template Service requester program
- WSBind files
8. You can browse the input copybook, EmploI01.cpy, by double-clicking it. It has the Department number, which is the key field in the request structure. Refer to Figure 12-19.
9. You can also browse the output copybook, EmploO01.cpy, by double-clicking it. It has the Department name and Manager number fields, which are the response structure from the external Web service. Refer to Figure 12-20.

![Web Services Explorer](image)

**Figure 12-20  EmploO01.cpy**

### 12.5 Updating the generated CICS service requester program

Next, you need to modify the generated COBOL Web service requestor program.

Browse the **Employee.cbl** that got generated, as shown in Example 12-2 on page 405. You have to update the program to populate the required request and response language structures and update several error handing messages. Look for the comments “Populate Request Language Structure” and “Process Response Language Structure” in the generated COBOL source as shown in Example 12-2 on page 405. Those places are the only two places where code needs to be added according to your data structures, as well as updating conditions for error handling.
**Example 12-2 Generated CICS Service requester program EMPLOYEE.cbl**

```
IDENTIFICATION DIVISION.
*Begin Identification Division
PROGRAM-ID. 'EMPLOYEE'.
  AUTHOR. WD4Z.
  INSTALLATION. 8.0.1.V200806191806.
  DATE-WRITTEN. 9/10/08 10:14 AM.
*End Identification Division
DATA DIVISION.
*Begin Data Division
WORKING-STORAGE SECTION.
*Begin Working-Storage Section
  * *************************************************************
  *        Operations Available On The Remote Web Service
  * *************************************************************
  1 OPERATION-NAME-1.
    2 PIC X(13) USAGE DISPLAY
        VALUE 'getDepartment'.
*End Working-Storage Section
LOCAL-STORAGE SECTION.
*Begin Local-Storage Section
  * *************************************************************
  *                    Program Work Variables
  * *************************************************************
  1 SOAP-PIPELINE-WORK-VARIABLES.
    2 WS-WEBSERVICE-NAME PIC X(32).
    2 WS-OPERATION-NAME PIC X(255).
    2 WS-CONTAINER-NAME PIC X(16).
    2 WS-CHANNEL-NAME PIC X(16).
    2 COMMAND-RESP PIC S9(8) COMP.
    2 COMMAND-RESP2 PIC S9(8) COMP.
*Specify A URI To Override The Web Service Description
  1 URI-RECORD-STRUCTURE.
    2 FILLER PIC X(10).
    2 WS-URI-OVERRIDE PIC X(255).
  * *************************************************************
  *                     Language Structures
  * *************************************************************
  1 LANG-EMPLOI01.
      COPY EmploI01.
  1 LANG-EMPLOO01.
      COPY EmploO01.
*End Local-Storage Section
```
LINKAGE SECTION.
*Begin Linkage Section
*End Linkage Section
*End Data Division
PROCEDURE DIVISION
.
*Begin Procedure Division
MAINLINE SECTION.
* -------------------------------------------------------------

* Initialize Work Variables
* -------------------------------------------------------------

INITIALIZE SOAP-PIPELINE-WORK-VARIABLES.
INITIALIZE URI-RECORD-STRUCTURE.
* -------------------------------------------------------------

* Container 'DFHWS-DATA' Must Be Present When A Service Requester Program Issues An EXEC CICS INVOKE WEBSERVICE Command. When The Command Is Issued, CICS Converts The Data Structure That Is In The Container Into A SOAP Request. When The SOAP Response Is Received, CICS Converts It Into Another Data Structure That Is Returned To The Application In The Same Container.
* -------------------------------------------------------------

MOVE 'DFHWS-DATA' TO WS-CONTAINER-NAME
* -------------------------------------------------------------

* Channel Passed To The Web Service Call
* -------------------------------------------------------------

MOVE 'SERVICE-CHANNEL' TO WS-CHANNEL-NAME
* -------------------------------------------------------------

* WEBSERVICE resource installed in this CICS region
* -------------------------------------------------------------

MOVE 'EmployeeWebservice' TO WS-WEBSERVICE-NAME
* -------------------------------------------------------------

* Operation To Invoke On The Remote Web Service
* -------------------------------------------------------------

MOVE OPERATION-NAME-1 TO WS-OPERATION-NAME
* -------------------------------------------------------------

* Populate Request Language Structure
* -------------------------------------------------------------

INITIALIZE LANG-EMPLOI01
* ....
* ....
EXEC CICS PUT CONTAINER(WS-CONTAINER-NAME)
   CHANNEL(WS-CHANNEL-NAME)
   FROM(LANG-EMPLOI01)
END-EXEC
PERFORM CHECK-CONTAINER-COMMAND

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

EXEC CICS GET CONTAINER(WS-CONTAINER-NAME)
   CHANNEL(WS-CHANNEL-NAME)
   INTO(LANG-EMPLOO01)
END-EXEC
PERFORM CHECK-CONTAINER-COMMAND

EXEC CICS RETURN
END-EXEC

CHECK-CONTAINER-COMMAND.
EVALUATE COMMAND-RESP
WHEN DFHRESP(CCSIDERR)
CONTINUE
WHEN DFHRESP(CONTAINERERR)
12.6 Creating a test CICS client program

Next, we show how to create a CICS client program that links to the generated COBOL program.

Create a simple CICS program that sends a basic mapping support (BMS) map to accept Department number as input and send Department name and Manager number as output. This CICS program will link to the previously explained CICS Web service requester program (EMPLOYEE) to invoke the external Web service program. Subsequently, the output received back from the external Web service will be passed back via the EMPLOYEE Web service requestor program and the CICS client program to the BMS map.

Note: We have included the complete updated source code and all other relevant sources in the additional material section of this book. Refer to Appendix B, “Additional material” on page 485 for instructions.
In our sample, we have named the CICS client program DEPTXGUI. This program is also included in the additional material appendix for this book.

Figure 12-21 shows the BMS map.

![CICS EXAMPLE GET DEPARTMENT APPLICATION](image)

- GetDepartment Menu
  - Using External WebService

Enter Dept. number, then press ENTER

Department No :
Department Name :
Manager Number :

F3=EXIT

Figure 12-21  CICS test client program

12.7 Deploying the application to z/OS

After all of the components have been prepared, we can deploy the solution to z/OS and run the application on z/OS. You need to perform the following tasks:

1. Compile and link-edit the CICS client program DEPTXGUI on z/OS.
2. Compile and link-edit the CICS Service requester program EMPLOYEE on z/OS.
Note: We assume that you have completed all of the necessary setup for CICS Web Services support, including the hierarchical file system (HFS) directories and the CICS TS resource definitions, such as PIPELINE and TCPIPSERVICE. Refer to Application Development for CICS Web Services, SG24-7126, for more details about how to set up these CICS resources.

3. Copy the generated Employeewebservice.wsbind file to the pickup directory of your CICS Service requester pipeline. You can view the WSBind file using the WSBind Viewer, which is shown in Figure 12-22.

![Figure 12-22 WSBind Viewer](image-url)
4. Perform a PIPELINE SCAN by entering the CEMT PERFORM PIPELINE(requester_pipeline_name) SCAN command. Make sure that the Response is NORMAL, as shown in Figure 12-23.

```cemt
PERFORM PIPELINE(ITSOREQ) SCAN
STATUS: RESULTS
Pip(ITSOREQ ) Sca
```

Figure 12-23  CEMT PERFORM PIPELINE SCAN

SYSID=WRKS  APPLID=CITS013
RESPONSE: NORMAL  TIME:  15.30.17  DATE: 03.20.09
PF 1 HELP  3 END  5 VAR  7 SBH  8 SFH  9 MSG  10 SB  11 SF

5. Additionally, check in the CICS Spool log to see whether the Web service was generated successfully (Figure 12-24 on page 412).
6. Check in the CICS region by typing `CEMT INQ WEBSERVICE`.

7. Navigate to the `EmployeeWebservice` and press Enter to see the Web service details, as shown in Figure 12-25 on page 413.
12.8 Testing the CICS Service requester program

To test the application, perform the following steps:

1. Type the Transaction ID associated with the CICS client program DEPTXGUI, which is RGUI in our example. Refer to Figure 12-26 on page 414.
2. Enter an existing Department number, such as A00, and press Enter. You see the result that was obtained from the Web service displayed on the window (Figure 12-27 on page 415).
12.9 Handling complex CICS COMMAREA structures

COBOL copybook structures are not known for dynamically creating data or interface structures as you can with XML data structures. The COBOL copybook needs to have fixed length records in case of arrays. The COBOL copybook has to know the number of times that a structure will repeat or occur. In case of XML data structures, however, you have the option to create dynamic structures by defining the option “unbounded” in the maxoccurs field and the minoccurs field as 0. In terms of COBOL then, the array can have 0 to any number of occurrences depending on a parameter that can be set by the application; however, RDz will not support generation of CICS Web Services artifacts in cases with such dynamic structures. To be able to process WSDL documents with dynamic data
structures, you need to edit the WSDL file first and update the maxoccurs and minoccurs fields to the actual number of OCCURS in your copybook.

For example, if you want your response structure to have an OCCURS clause for a response structure of 15 times in your copybook, just edit the WSDL file and update the maxoccurs and minoccurs to 15 as shown in Figure 12-28.

![Figure 12-28 XML structure with complexType](image)

After updating the WSDL file and subsequently generating the COMMAREA in RDz, you will notice output as shown in Figure 12-29 on page 417.
12.10 Summary

In this chapter, we have shown how an existing external Web service can be easily used from a CICS application by adopting a top-down approach. There might be situations where you have to write one or several wrapper programs to bridge the differences between the outside Web service and the already existing CICS programs. It is also possible that at the same time of integrating external Web services you want to refactor or re-architect existing CICS applications. Changes in both the WSDL of the external Web service and the existing CICS programs might be necessary to produce a flexible solution. This approach of bridging is referred to as the “meet-in-the-middle” approach.

Important: In this sample, we assume that the CICS application makes a direct call to a Web service implemented in an application server elsewhere. It also looks as though CICS service requestors might need to be adjusted to the external WSDL, or both the WSDL and the CICS service requestor might need to be adjusted to create a fit. Note, however, that discrepancies between service requestors in CICS and external Web services can also be perfectly resolved by an ESB.
Chapter 13. Integrating an external Web service into an IMS application

There has been an increasing need for the IMS business functions to access new business logic running outside the IMS environment. IMS supports access to external Web services by means of asynchronous callout support. With this functionality, users can develop new applications or extend their existing IMS applications to access new business logic or data outside IMS and use state-of-the-art Web Services techniques. Web Services calls are routed from IMS via the IMS SOAP Gateway.

Rational Developer for System z (RDz) provides wizards that generate the XML converter and the correlator file that are needed to enable your IMS application to run as a Web Service consumer.

In this chapter, we discuss the following topics:

- Prerequisites
- Outbound Web Services overview
- Development and deployment steps
13.1 Prerequisites

You must have the following prerequisite software to develop and deploy this scenario:

- IMS Version 10 or later
- IMS SOAP Gateway Version 10.1 or later
- Rational Developer for System z (RDz) Version 7.1.1 or later

13.2 Outbound Web Services overview

This sample explains the callout to a Request-Response type of Web service. The process is depicted graphically in Figure 13-1.

The steps in Figure 13-1 are (the numbers in the diagram correspond to the numbers in the list):

1. Waiting for callout message.
When IMS SOAP Gateway starts a callout thread, IMS SOAP Gateway establishes a TCP/IP connection with IMS Connect and sends a request from the hold queue (the tpipe) to IMS Connect. The tpipe name is obtained from the callout connection bundle file.

2. Start IMS application 1.
   A client starts IMS application 1 DFSIVA98.

3. Send the callout message to IMS Connect.
   IMS application 1 inserts into the *Alternate Program Control Block (ALTPCB)* using the Open Transaction Manager Access (OTMA) destination routing descriptor SOAPGWAY. The descriptor has the IMS Connect cross-system coupling facility (XCF) member name (*TMEMBER*), the hold queue name (*TPIPE*), the XML Adapter name (*ADAPTER*), and the XML converter name (*CONVERTR*). The XML converter is generated by using Rational Developer for System z.

   OTMA reads the descriptor SOAPGWAY that has a tpipe name of IMSS0AP and sends the callout message to the asynchronous hold queue IMSS0AP. The adapter name (HWSXMLA0) and converter name (DFSIVA9D) specified in the OTMA destination routing descriptor are copied to the OTMA header of the callout message.

4. Callout message transformation from COBOL application data to XML.
   IMS Connect retrieves the existing callout message from IMSS0AP as part of the request processing for IMS SOAP Gateway. IMS Connect reads the OTMA header for the adapter name and the converter name. And, it calls the adapter HWSXMLA0 and the converter DFSIVA9D for XML processing. The converter DFSIVA9D performs the callout message transformation from COBOL application data to XML.

5. Send the callout message to IMS SOAP Gateway.
   After the converter converts the callout message successfully, IMS Connect sends the callout request message back to IMS SOAP Gateway.

6. Receive the callout message.
   IMS SOAP Gateway receives the callout request message in XML and parses the message to retrieve the service and payload data. IMS SOAP Gateway loads the corresponding correlator file and retrieves the Web service information and properties for invoking the Web service.

7. Invoke the Web service.
   IMS SOAP Gateway sends the callout message to the Web service using the SOAP/HTTP protocol.
8. Send response message.
   The Web service runs and generates a response message and sends it back to IMS SOAP Gateway.

   IMS SOAP Gateway adds the transaction code for IMS application 2, IVT99; the adapter name, HWSXMLA0; and the converter name, DFSIVA9D (all retrieved from the correlator file) and sends the response message to IMS Connect.

10. Response message transformation from XML to COBOL application data.
    IMS Connect receives the response message and loads the response converter, DFSIVA9D, for XML to bytes conversion. IMS Connect takes the transaction code value and adds it to the appropriate place (LLZZTRCDDATA) in the message.

11. Start IMS application 2.
    IMS invokes IMS application 2 DFSIVA99 to process the output data.

12. Send response message to initiating client.
    The IMS application 2 sends a response message to the initiating client.

### 13.3 Development and deployment steps

In the following sections, we explain the steps to develop and deploy a solution with an IMS Web service consumer. We use Rational Developer for System z for most of the steps:

1. Step 1: Modify or create the IMS application for callout requests.
   You must modify or create your IMS application for sending a callout request and receiving a response message back from an external Web service.

2. Step 2: Define an OTMA destination routing descriptor for callout messages.
   The *OTMA destination routing descriptor* enables an IMS application to route callout requests to services that are accessible to the IMS SOAP Gateway without the need to code routing exits.

   Generate the mapping between the data structures in your IMS application and the external Web service, and generate the XML converter program by using Rational Developer for System z.
4. Step 4: Create a connection bundle for the callout function.
   Create a connection bundle for the callout function that describes the connection properties for accessing IMS. This bundle is created by using the IMS SOAP Gateway Deployment utility.

5. Step 5: Create a correlator file for the callout function.
   Create a correlator file for the callout function that describes the information to invoke the outbound Web service and return the response message back to IMS by using the IMS SOAP Gateway Deployment utility.

   Use the IMS SOAP Gateway Deployment utility to deploy your IMS application.

7. Step 7: Start the callout threads.
   You must start a callout thread after you deploy your callout client.

13.3.1 Step 1: Modify or create the IMS application for callout requests

You must modify or create your IMS application in order to place the callout request on the OTMA asynchronous hold queue or route the request to an IMS Connect destination. In the example, the IMS application inserts into the alternate program control block (ALTPCB) using the OTMA destination routing descriptor SOAPGWAY.

**Important:** The destination of ALTPCB is the destination name of the OTMA destination routing descriptor, not the tpipe name.

The COBOL code for sending the callout request is shown in Example 13-1.

Example 13-1  Example of COBOL code for sending the callout request

```cobol
01  EXTERNAL-SERVICE.
   02  EX-DEST   PICTURE X(8) VALUE 'SOAPGWAY'.
   CALL 'CBLTDLI' USING CHNG, ALTPCB, EX-DEST.
   MOVE LTERM-NAME TO IN-LTERM-NAME.
   CALL 'CBLTDLI' USING ISRT, ALTPCB, INPUT-MSG.
   CALL 'CBLTDLI' USING PURG, ALTPCB.
```

**Note:** The checks on the DL/I status code and the error processing are omitted.
You must modify or create the IMS application program to correlate the response to the original input transaction. It will likely require the application program to include a type of data to be used for correlation purposes in the outgoing callout request message and then to look for that same data in the response message. If a reply must be sent back to an LTERM, the LTERM name must also be stored in the database or copied in the callout message. The following code samples show examples of COBOL copybook data structures that might be used for the callout request and the callout response. The callout request data structure includes a correlator field, IN-LTERM-NAME, to send correlation data to the external Web service, as shown in Example 13-2.

Example 13-2  Example of COBOL copybook for the callout function

```
01  INPUT-MSG.
   02  IN-LL          PICTURE S9(3) COMP.
   02  IN-ZZ          PICTURE S9(3) COMP.
   02  IN-TRXCD       PICTURE X(8).
   02  IN-FILL        PICTURE X(2).
   02  IN-COMMAND     PICTURE X(8).
   02  TEMP-COMMAND REDEFINES IN-COMMAND.
      04  TEMP-IOMCD    PIC X(3).
      04  TEMP-FILLER   PIC X(5).
   02  IN-LAST-NAME   PICTURE X(10).
   02  IN-FIRST-NAME  PICTURE X(10).
   02  IN-EXTENSION   PICTURE X(10).
   02  IN-ZIP-CODE    PICTURE X(7).
   02  IN-LTERM-NAME  PICTURE X(8).
```

When the response is returned, the IN-LTERM-NAME field of the COBOL copybook data structure contains the same correlation data, which the IMS application can use to relate the response to the initial request, if needed (Example 13-3 on page 425).
Example 13-3  Example of COBOL copybook for the callout response

01 INPUT-MSG.
   02 IN-LL     PICTURE S9(3) COMP.
   02 IN-ZZ     PICTURE S9(3) COMP.
   02 IN-TRXCD  PICTURE X(8).
   02 IN-BLANK  PICTURE X(1).
   02 IN-LTERM-NAME PICTURE X(8).
   02 INPUT-LINE  PICTURE X(85).
   02 INPUT-DATA REDEFINES INPUT-LINE.
      04 IN-MESSAGE PIC X(40).
      04 IN-COMMAND PIC X(8).
      04 IN-DATA-TYPE.
         06 IN-LAST-NAME PIC X(10).
         06 IN-FIRST-NAME PIC X(10).
         06 IN-EXTENSION PIC X(10).
         06 IN-ZIP-CODE PIC X(7).
   02 IN-SEGMENT-NO PICTURE X(4).

To route a response message to a terminal, the IMS application that issues the callout request can capture the LTERM name of the terminal. Upon receipt of the response message, the IMS application that processes the response can then identify the originating terminal by the LTERM name. The LTERM name can either be included as part of the data of the callout request and the response messages or the LTERM name can be saved in a temporary database that is accessible to the IMS application that processes the response.

After the response is returned, the IMS application can return the response to the initiating terminal by making a CHNG call and ISRT the response to the LTERM name captured by the IMS application program that issued initial callout request.

### 13.3.2 Step 2: Define an OTMA destination routing descriptor for callout messages

Use OTMA destination routing descriptors to allow OTMA clients, such as IMS Connect, or non-OTMA clients, such as SNA terminals or printers, to define message switch destinations from the ALTPCB. Instead of coding OTMA routing exit routines (DFSYPRX0 and DFSYDRU0), you can externalize the message switch definitions in the DFSYDTx member of the IMS.PROCLIB data set. The x on DFSYDTx is the IMS nucleus suffix.

Specify the ADAPTER and the CONVERTER values. The ADAPTER value must be set to HWSXMLA0 if you want XML data transformation to be performed by IMS.
Connect. Example 13-4 shows an example of the OTMA destination routing descriptor, which spans multiple lines.

**Example 13-4   Example of OTMA destination routing descriptor for callout message**

```
D SOAPGWAY TYPE=IMSCON T MEMBER=HWS10A TPIPE=IMSSOAP
D SOAPGWAY ADAPTER=HWSXMLA0 CONVERTR=DFSIVA9D
```

Note that:

- The descriptor for SOAPGWAY routes messages to IMS Connect T MEMBER HWS10A with tpipe IMSSOAP.
- The ADAPTER value must be set to HWSXMLA0 if you want XML data transformation to be performed by IMS Connect by using the IMS Connect XML adapter.
- The XML converter name is DFSIVA9D.

**Important:** The tpipe that you use for the IMS SOAP Gateway callout function must not be shared with another callout function, such as the callout function that is used in the IMS TM resource adapter.

If the initiating client is an SNA terminal, you must code OTMA destination routing descriptors or OTMA routing exit routines, because the response message is not routed to an SNA terminal but OTMA. An OTMA destination routing descriptor sample is shown in Example 13-5.

**Example 13-5   OTMA destination routing descriptor for SNA terminal**

```
D ltermname TYPE=NONOTMA
```

For more information, refer to the OTMA descriptors topic in *IMS Version 10 Communications and Connection Guide*, SC18-9703, and the DFSYD T x PROCLIB member in *IMS Version 10 System Definition Reference*, GC18-9966.

**13.3.3 Step 3: Generate artifacts with RDz**

Rational Developer for System z (RDz) provides a wizard that generates the correlator file and the XML converter that are needed to enable your IMS application to run as a Web services consumer.
For Rational Developer for System z to perform data mapping and create the mapping files between your IMS application and the Web service that you are accessing, you must have the file describing the data structure in your application and the Web service's Web Services Description Language (WSDL) file on your workstation:

- For inbound message mapping, the source files are WSDL, XML, or XSD files.
- For outbound message mapping, the source files are CBL files.

The artifacts generated are:

- The correlator file in XML (.xml)
- The XML converters in COBOL (.cbl)

**Note:** In our environment, we were not able to edit the mapping between the data structure of the IMS application and the Web service using RDz Version 7.1.1.2. In the following steps “Step 3a: Generate mapping data” on page 427 and “Step 3b: Generate Web service files” on page 439, we used RDz Version 7.1.0 instead. We did not test this scenario using RDz Version 7.5, and it is possible that this problem is fixed in Version 7.5.

**Step 3a: Generate mapping data**

Rational Developer for System z provides wizards that map the data structures between your IMS application and the Web service that the IMS application is accessing. In this step, we describe how the mapping is generated:

1. Start Rational Developer for System z and open the z/OS Projects perspective.
2. Create a new local COBOL project:
   a. Select **File → New → Project**.
   b. Expand the **Workstation COBOL or PL/I** folder.
   c. Select **Local Project**, and click **Next**.

Refer to Figure 13-2 on page 428.
3. In the next window, type a name for the project, for example, MyProject.
   Click Finish. A new project called MyProject is now available in your workspace.

4. Create a folder for your source files:
   a. In the z/OS Projects view, right-click the name of the new project that you just created.
   b. Select New → Folder. The New Folder wizard opens.
   c. In the New Folder wizard:
      • Select the project that you just created.
      • In the Folder name input field, type the name that you want to use for the new folder, for example, MySource.
      • Click Finish. The folder is created.

Refer to Figure 13-3 on page 429.
5. Copy your source files from a directory on your workstation into the proper directory in the Navigator view:
   
   - Open the Navigator view:
     
     - Select Window → Show View → Other. The Show View wizard opens.
     
     - In the Show View window, expand General and select Navigator.
     
     - Click OK. The Navigator view opens.
   
   - In the Navigator view:
     
     - Expand your project, for example, MyProject.
Select the directory for the project, for example, **MySource**. The directory has the same name and path as the new folder in the z/OS Projects view (for example, MyProject/MySource).

- Open the directory on your workstation that contains the COBOL copybook.
- Drag the Web service WSDL file from the workstation directory to the new folder (for example, **MySource**) in the Navigator view.

![Figure 13-4 Copy your source files](image)

6. Right-click the COBOL copybook and select **Enable Enterprise Web Service**. The Enterprise Service Tools Wizard Launchpad opens.

7. In the launchpad, as shown in Figure 13-5 on page 431, select the following settings:
   - Host runtime: **IMS SOAP Gateway**
   - Development scenario: **Map to an Existing Service Interface (meet-in-middle)**
   - Application mode: **Web Service Requestor**
   - Conversion type: **Compiled XML Conversion**

8. Click **Start**. Refer to Figure 13-5 on page 431. The New XML to COBOL Mapping wizard opens, as shown in Figure 13-6 on page 432.
9. In the New XML to COBOL Mapping page of the wizard, as shown in Figure 13-6 on page 432:

   – In the Mapping source field, the name of the copybook that you right-clicked is selected.

     If this name is not the file that you want to use for the source, use **Browse** to choose another source file.

     **Important**: If you specify a WSDL, XSD, or XML file for this field, the wizard changes to inbound mapping mode.

   – In the Mapping target field, specify the WSDL file for the Web service to which your IMS application is calling out.

     **Important**: If you specify a high-level language file (such as a .cbl file) for this field, the wizard changes to outbound mapping mode.

   – Click **Next**.

     Refer to Figure 13-6 on page 432.
10. In the Root XML Element and Language Structure Selection page of the wizard, as shown in Figure 13-7 on page 433:
   - The fields are automatically filled in based on the COBOL copybook source and the WSDL target file that you specified.
     Adjust the values if they are not what you want by selecting from the selection drop-down lists.
   - Click **Next**.

Refer to Figure 13-7 on page 433.
11. On the New XML to COBOL Mapping Session page of the wizard, as shown in Figure 13-8 on page 434:

- In the Mapping file folder field, specify the path for the folder in which you want the new outbound mapping session file to be created.
- In the Map file name field, specify a name for the new outbound mapping session file.
- Click Finish.

Refer to Figure 13-8 on page 434. A mapping session file with a mapping extension is created in the file folder that you specified.
12. Double-click the mapping session file that you want to edit. The mapping editor opens. In the mapping editor:

- For each mapping that you want to create, drag an element in the source data to an element in the target data. The editor displays a connecting line between the source element and the target element to indicate that a mapping exists.

- When you are finished creating the mapping, close the mapping editor. Refer to Figure 13-9 on page 435.
13. Repeat steps 9 on page 431 to 12 on page 434 to create the inbound message mappings from the Web service to the COBOL copybook. However:

- Instead of starting from a COBOL copybook, right-click the WSDL file of the Web service to which you are calling out and select **Enable Enterprise Web service**.

- In the New XML to COBOL Mapping wizard, as shown in Figure 13-10 on page 436:
  - For the Mapping source field, specify the WSDL file for the Web service
  - For the Mapping target field, specify the COBOL copybook.

The two mapping session files now contain the mappings that you created.
Figure 13-10  New XML to COBOL Mapping Session for inbound data mapping

Figure 13-11 on page 437 shows the Root XML Element and Language Structure Selection pane of the wizard for the inbound data mapping with the defaults.
Figure 13-11  Root XML Element and Language Structure Selection for inbound data mapping

Figure 13-12 on page 438 shows the New XML to COBOL Mapping Session pane of the wizard for the inbound data mapping.
Figure 13-12  New XML to COBOL Mapping Session for inbound data mapping

Figure 13-13 on page 439 shows the Mapping Editor for the inbound data mapping.
Step 3b: Generate Web service files

Use the Map an Existing Service Interface (meet-in-middle) wizard in Rational Developer for System z to generate the Web services files to enable your IMS application to run as a Web service consumer.

You must have already performed the data mapping task by using Rational Developer for System z as described in “Step 3a: Generate mapping data” on page 427.

In Rational Developer for System z, select the mapping session files that you have just created:

1. Right-click anywhere in the view that contains the mapping files.

   If you want the meet-in-middle wizard to create a bidirectional Web service, select an inbound mapping session file and an outbound mapping session file, as shown in Figure 13-14 on page 440.
2. Right-click the mapping files and select **Generate Conversion Code** from the pop-up menu, as shown in Figure 13-14. The Enterprise Service Tools Wizard Launchpad opens, as shown in Figure 13-15 on page 441.

3. In the launchpad, choose the following values:
   - In the Host runtime field, select **IMS SOAP Gateway**.
   - In the Application mode field, select **Web Service Requestor**.

4. Click **Start**. The Map an Existing Service Interface (meet-in-middle) wizard opens.
5. In the Generation of conversion code page of the Map to an Existing Service Interface (meet-in-middle) wizard, verify that you have selected the correct mapping session files. Click Next.

6. In the Generation options page of the Map to an Existing Service Interface (meet-in-middle) wizard, in the XML Converters tab, as shown in Figure 13-16 on page 442:
   - In Specify identification attributes, specify attributes if necessary.
   - In Specify character encodings, select or change the following settings if necessary:
     - Host code page:
       Select the code page that the host uses.
     - Inbound code page and outbound code page:
       Default is 1208 Unicode, UTF-8. IMS SOAP Gateway supports only UTF-8, and therefore you cannot change this value.
   - Click Next.

Refer to Figure 13-16 on page 442.
7. On the IMS SOAP Gateway Web Service Requestor page, you can leave the
defaults. You can use the IMS SOAP Gateway Deployment utility to create
these correlator properties later. Click **Next**. Refer to Figure 13-17 on
page 443.
8. In the File, data set, or member selection page (Figure 13-18 on page 444) of the wizard, specify the following fields if necessary:

- For the converter file container, specify the folder and subfolder in which you want the converter file to be created.

- For the converter driver file name, specify the name in which you want the converter programs to be generated.
Click **Finish**. The following files are generated in the directories and filenames that you specified:

- The file containing the Web service driver and runtime XML converter (\*.cbl).
- The correlator file (\*.xml).

![Image of file, data set, or member selection](image)

**Figure 13-18  File, data set, or member selection**

**Note:** Ensure that **Generate all to driver** is selected. This selection causes all the generated Web service programs (driver, inbound converter, and outbound converter) to be placed in the same file.
Step 3c: Deploy artifacts to IMS Connect

After the artifacts have been generated, you can upload the artifacts that are required for IMS Connect to z/OS:

1. Upload the generated XML converter to your COBOL library on z/OS.
2. Compile and link-edit the generated XML converter so that it can be accessed by IMS Connect.

Note: The XML converter must have “X” as the alias name suffix.

13.3.4 Step 4: Create a connection bundle for the callout function

In this step, a connection bundle is created that describes the connection properties for access to IMS by using the IMS SOAP Gateway Deployment utility. The connection bundles are stored in the connbundle.xml file. You must create a new connection bundle or modifying an existing connection bundle to specify:

- The connection properties for IMS Connect
- The names of the pipes that hold the callout requests that are sent by your IMS application

If a response message is expected from the Web service, you might have to create additional connection bundles or reuse an existing connection bundle to specify connection properties for sending the output response message.

To create a connection bundle:

1. Start the IMS SOAP Gateway Deployment utility on z/OS by running iogdeploy.sh in the installation_dir/deploy directory by using the Uniform Symbol Specification shell Terminal Emulator in RDz.
2. Enable your IMS application to access an external Web service.
   - Select Task 10: Enable your IMS application to access an external Web service.
3. Create a connection bundle for callout:
   a. From the callout menu, select Task 9: Create, Update, Delete or View connection bundle for callout.
   b. At the prompt, enter c to create a connection bundle. You will be prompted to enter the name for the bundle.
   c. When you are prompted, enter the following information:
      - Name for the connection bundle
      - IMS host name or IP address
- IMS port number (default is 9999)
- IMS datastore name
- IMS user ID (optional)
- IMS password (optional)
- IMS group name (optional)

d. If you want to set Secure Sockets Layer (SSL) properties, at the prompt where the question is asked, enter y. At the prompt, enter the following information:

- SSL keystore name
- SSL keystore password
- SSL truststore name
- SSL truststore password
- SSL encryption type

e. At the prompt, enter the tpipe name. If there are multiple tpipes, separate the names with a comma.

Figure 13-19 on page 447 shows the dialog to create a connection bundle.
Enable your IMS application to access an external Web service (callout):

Callout tasks:
- Task 2: Start all callout threads
- Task 3: Stop all callout threads
- Task 4: Obtain status of all callout threads
- Task 5: Start a specific callout thread
- Task 6: Stop a specific callout thread
- Task 7: Update IMS SOAP Gateway callout properties
- Task 8: Create, Update or View correlator properties for callout
- Task 9: Create, Update, Delete or View connection bundle for callout
- Task 10: Deploy a callout client
- Task 11: Undeploy a callout client
- Task 12: Return to main menu

> Enter your selection here:

9

Task 9 allows you to group a set of connection properties as a connection bundle for easy management and reuse. You can use this task to create, delete or update connection bundles for Callout.

> Enter 'c' to create a new connection bundle, 'u' to update an existing one, 'd' to delete an existing one or 'v' to view all connection bundles:

c
- Provide the following connection bundle properties.

> Provide a name for the connection bundle:

outbundle1

> Provide IMS host name or IP address:

wtscz1.itso.ibm.com

> Provide IMS port number (default: 9999):

6001

> Provide IMS datastore name:

IMSA

> Provide IMS userid (optional):
The password will not appear.

> Provide IMS password (optional):

> Provide IMS group name (optional):

> Do you want to set SSL properties? (y/n):

n

> Provide Callout TPipes: (Use comma separator for multiple tpipe names)

IMSSOAP

- The connection bundle 'outbundle1' has been created.

> Do you want to view the connection bundle? (y/n):

n
Update correlator files if needed.
13.3.5 Step 5: Create a correlator file for the callout function

You can create a correlator file by using the IMS SOAP Gateway Deployment utility. A **correlator file** provides the information to invoke the outbound Web service and return the response message back to IMS. The information includes:

- The WSDL file name for the Web service to be invoked
- The timeout value for waiting for a response from the Web service
- The IMS transaction code and the connection bundle name for returning the callout response

To create a correlator file:

1. Start the IMS SOAP Gateway Deployment utility on z/OS by running `iogdeploy.sh` in the `installation_dir/deploy` directory.

2. Create a correlator file for the callout function:
   a. Select **Task 10: Enable your IMS application to access an external Web Service**.
   b. From the callout menu, select **Task 8: Create, Update or View correlator properties for callout**.
   c. At the prompt, enter `c` to create a correlator file.
   d. At the prompt, enter the callout WSDL name. The file name must have `wsdl` as the extension. The file must exist and use valid schema.
   e. If you expect a response from the Web service that you are calling, answer `y` when you are asked if you want to update the response callout options. Enter the information as you are prompted.

Figure 13-20 on page 449 and Figure 13-21 on page 450 show the dialog to create a correlator file.
Enable your IMS application to access an external Web service (callout):

Callout tasks:
- Task 2: Start all callout threads
- Task 3: Stop all callout threads
- Task 4: Obtain status of all callout threads
- Task 5: Start a specific callout thread
- Task 6: Stop a specific callout thread
- Task 7: Update IMS SOAP Gateway callout properties
- Task 8: Create, Update or View correlator properties for callout
- Task 9: Create, Update, Delete or View connection bundle for callout
- Task 10: Deploy a callout client
- Task 11: Undeploy a callout client
- Task 12: Return to main menu

> Enter your selection here:

8

Task 8 helps you create, update or view the IMS correlator properties for a callout client.

> Enter 'c' to create a new correlator file, 'u' to update existing file or 'v' to view correlator file:

8

- Provide the following callout correlator properties.

> Provide the name of the WSDL file of the external web service that you want to access (e.g. MyWSDLFile.wsdl):

**PHONEBK.wsdl**

The name of the WSDL file will be used as the WSID (Web services identifier) for this callout client. It will also be the name of the correlator file being created.

Correlator file name is:

/\usr/\lpp/ims/imssoap/server/webapps/imssoap/xml/PHONEBK.xml

> Do you want to set properties to handle the response from the callout web service? (y/n):

y

- Provide the following correlator properties.

> Enter the converter name:

**DFSIVA9D**

> Enter the socket timeout value (in milliseconds; default is 5000):

> Enter the execution timeout value (in milliseconds; default is 5000):

> Enter the lterm name:

> Enter the adapter type (default is ""):

> Enter the connection bundle name:

**connbundel1**
13.3.6 Step 6: Deploy artifacts to IMS SOAP Gateway

Use the IMS SOAP Gateway Deployment Utility to deploy the artifacts for your IMS application. This task is also known as *deploying the callout client*.

Before you start, make sure that you have the following files ready:

- WSDL file of the target Web service
- Correlator file that was created by using either Rational Developer for System z Version 7.1 or later, or Task 10-8: Create, Update or View correlator properties for callout in the IMS SOAP Gateway Deployment Utility

Perform the following steps:

1. Start the IMS SOAP Gateway Deployment Utility by running `iogdeploy.sh` in the `installation_dir/deploy` directory.

2. Deploy a callout client:
   a. Select **Task 10: Enable your IMS application to access an external Web Service**
   b. From the callout menu, select **Task 10: Deploy a callout client**.
   c. Enter the following information when you are prompted:
      - Name and the location of the WSDL file of the Web service that you want to access
      - Name and the location of the correlator file

Figure 13-22 on page 451 shows the dialog to deploy the callout client.
Enable your IMS application to access an external Web service (callout):

Callout tasks:
Task 2: Start all callout threads
Task 3: Stop all callout threads
Task 4: Obtain status of all callout threads
Task 5: Start a specific callout thread
Task 6: Stop a specific callout thread
Task 7: Update IMS SOAP Gateway callout properties
Task 8: Create, Update or View correlator properties for callout
Task 9: Create, Update, Delete or View connection bundle for callout
Task 10: Deploy a callout client
Task 11: Undeploy a callout client
Task 12: Return to main menu

> Enter your selection here:
10

Task 10 deploys the callout client to make an external web service accessible to an IMS application.

At the prompt, specify the name and location of the WSDL file of the external Web service.
(e.g. c:\MyWSDLFile.wsdl).

If you had deployed the WSDL before or you have copied the WSDL into the IMS SOAP Gateway wsdl directory, provide just the name for the WSDL file (e.g. MyWSDL.wsdl), the full path is NOT required.

> Provide the name and location of the WSDL file:
PHONEBK.wsdl

At the prompt, specify the name and location of the correlator file that you have created.
for the Web service (e.g. c:\MyCrr.xml).

If you had created the correlator file before or you have copied the correlator file into the IMS SOAP Gateway xml directory, provide just the name of the correlator file (e.g. MyCrr.xml). The full path is NOT required.

> Provide the name and location of the correlator file:
PHONEBK.xml
- Callout client was successfully deployed.

Figure 13-22  Deploy a callout client (sample)
13.3.7 Step 7: Start the callout thread

You must start a callout thread after you deploy your callout client.

**Important:** This task operates at the thread level. The behavior of this task depends on the thread policy that has been set. If the policy is to have one thread per tpipe, the start and stop action affects only that particular tpipe. If the thread policy is set to one thread per connection bundle, the start and stop action affects all of the tpipes that are associated with that connection bundle.

To start the callout thread:

1. Start the IMS SOAP Gateway Deployment Utility on z/OS by running `iogdeploy.sh` in the `installation_dir/deploy` directory.
2. Start a specific callout thread:
   a. From the utility’s main menu, select **Task 10: Enable your IMS application to access an external Web Service**. The callout menu displays.
   b. From the callout menu, select the appropriate task.
      To start a specific resume tpipe thread, select **Task 5: Start a specific callout thread**.
   c. When you are prompted, enter the connection bundle name to start the callout thread.
   d. If the thread policy is one thread per tpipe, you are prompted to enter the name of the tpipe where the callout messages are held. Enter the tpipe name.
   e. The callout thread for the connection bundle and tpipe name that you specified are started. To verify that a resume tpipe thread has started, select **Task 4: Obtain status of all callout threads from the callout menu**.

Figure 13-23 on page 453 shows the dialog to start a callout thread.
Enable your IMS application to access an external Web service (callout):

Callout tasks:
Task 2: Start all callout threads
Task 3: Stop all callout threads
Task 4: Obtain status of all callout threads
Task 5: Start a specific callout thread
Task 6: Stop a specific callout thread
Task 7: Update IMS SOAP Gateway callout properties
Task 8: Create, Update or View correlator properties for callout
Task 9: Create, Update, Delete or View connection bundle for callout
Task 10: Deploy a callout client
Task 11: Undeploy a callout client
Task 12: Return to main menu

> Enter your selection here:
5
Your current thread policy is set to one thread per connection bundle. If this thread is stopped, polling will stop for all the tpipes in that connection bundle.

> Provide the connection bundle name that contains the tpipe:
**outbundle1**


Enable your IMS application to access an external Web service (callout):

Callout tasks:
Task 2: Start all callout threads
Task 3: Stop all callout threads
Task 4: Obtain status of all callout threads
Task 5: Start a specific callout thread
Task 6: Stop a specific callout thread
Task 7: Update IMS SOAP Gateway callout properties
Task 8: Create, Update or View correlator properties for callout
Task 9: Create, Update, Delete or View connection bundle for callout
Task 10: Deploy a callout client
Task 11: Undeploy a callout client
Task 12: Return to main menu

> Enter your selection here:
4

Below is the status of callout threads:
Thread **outbundle1** is running!

*Figure 13-23  Start a specific callout thread (sample)*
Your environment is now ready to use, and you can test the IMS Web service requestor application.
Using DB2 as a Web service requester

This chapter demonstrates how to consume Web services in DB2 applications using the SOAP user-defined functions (UDFs) in DB2 Version 9 for z/OS.
14.1 Overview

DB2 Version 9 for z/OS can act as a Web service consumer, allowing you to easily integrate external Web services in your DB2 applications.

The DB2 support for Web services invocation comes in the form of four user-defined functions (UDFs). Actually, it is conceptually only one function, with four variations differing only in the return type and the type of the last parameter, with a choice of either VARCHAR or CLOB (Figure 14-1 on page 457). Which one you choose depends on the amount of data that you want to pass to a Web Service call and the amount of data that you expect will be returned from the Web service call.

The UDFs take the following parameters:

- **ENDPOINT_URL: VARCHAR(256)**
  This parameter is the endpoint information, in the form of a URL, of the Web service to invoke. You typically retrieve this information from the WSDL.

- **SOAP_ACTION: VARCHAR(256)**
  This parameter specifies a SOAP action URI reference. Depending on the Web server, this parameter might not be needed. If it is required, the value to be used is defined in the Web Services Description Language (WSDL) of the Web service. If not, pass an empty string.

- **SOAP_INPUT: VARCHAR(32672) or character large object (CLOB)**
  This parameter is the complete SOAP request message in the form of an XML document.

Depending on the expected size of the SOAP return message, you use either the SOAPHTTPNV or the SOAPHTTPNC variant. For SOAPHTTPNV, the output is returned as VARCHAR(32672); for SOAPHTTPNC, it is returned as CLOB.

**Note:** There are two other UDFs with similar names: SOAPHTTPPC and SOAPHTTPPV. In contrast to SOAPHTTPNC and SOAPHTTPNV, these functions do not accept and return a complete SOAP request and response. They only return the request (or response) body. These functions are considered deprecated.

Figure 14-1 on page 457 shows the four UDFs.
In addition, DB2 Version 9 for z/OS has rich functionality to parse XML documents, to extract parts of an XML document and convert it to an SQL type, to convert XML data to tabular format, and to convert tabular format data to XML data.

In the following sections, we show a step-by-step example how to use these functions.

```
DB2XML.SOAPHTTPNV (  
    ENDPOINT_URL VARCHAR(256),  
    SOAP_ACTION VARCHAR(256),  
    SOAP_INPUT VARCHAR(32672)  
  ) RETURNS VARCHAR(32672)

DB2XML.SOAPHTTPNV (  
    ENDPOINT_URL VARCHAR(256),  
    SOAP_ACTION VARCHAR(256),  
    SOAP_INPUT CLOB(1M)  
  ) RETURNS VARCHAR(32672)

DB2XML.SOAPHTTPNC (  
    ENDPOINT_URL VARCHAR(256),  
    SOAP_ACTION VARCHAR(256),  
    SOAP_INPUT VARCHAR(32672)  
  ) RETURNS CLOB(1M)

DB2XML.SOAPHTTPNC (  
    ENDPOINT_URL VARCHAR(256),  
    SOAP_ACTION VARCHAR(256),  
    SOAP_INPUT CLOB(1M)  
  ) RETURNS CLOB(1M)
```

Figure 14-1 The DB2-supplied UDFs for outbound SOAP request support

**Note:** If these UDFs seem to be missing in your DB2 installation, ask your DB2 administrator to run the relevant SQL statements in the DB2 installation job DSNTIJSG.
14.2 Introduction to the example

In this section, we are going to call the getDepartment Web service that we created and deployed in 7.2, “Using Data Web Services (DWS)” on page 160. Admittedly, this example might seem a bit contrived, because after all, the service accesses data that resides on the same DB2 system that calls the service.

The steps are:
1. Creating the SOAP request document
2. Creating the SQL to invoke the SOAP UDF
3. Parsing the SOAP response message
4. Creating a wrapper UDF

14.3 Creating the SOAP request document

First, we must build the SOAP request message in XML form. Now, rather than writing the XML document manually, we can use the Web Services Explorer tool to build a prototype request message and use this message as a starting point.

In fact, we have already done that in 7.2.7, “Testing the Web service” on page 169. Here is the SOAP request again (Example 14-1, slightly reformatted for better readability).

Example 14-1  SOAP request for calling the UDF (Version 1)

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope xmlns:q0="http://sg247669.itso.ibm.com/employee/
 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:getDepartment>
      <DEPTNO>A00</DEPTNO>
    </q0:getDepartment>
  </soapenv:Body>
</soapenv:Envelope>
```

For easier readability and maintenance, we can improve and simplify the request document slightly (Example 14-2 on page 459).
Example 14-2  SOAP request for calling the UDF (Version 2)

```xml
<?xml version="1.0" encoding="UTF-8" ?>
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ews:getDepartment>
      <DEPTNO>A00</DEPTNO>
    </ews:getDepartment>
  </soapenv:Body>
</soapenv:Envelope>
```

Notes on Example 14-2 (these numbers correspond to the numbers on the right of the example):

1. Instead of sticking with the random q0 namespace prefix, we changed it to ews ("Employee Web Service").
2. Because the XML Schema and XML Schema-instance namespaces are not used in this document, we can simply omit the respective declarations.
3. Here, we use the ews prefix to refer to our application-specific elements.

14.4  Creating the SQL to invoke the SOAP UDF

Now, we are ready to create the SQL to invoke our Web service. It will call the SOAPHTTPNV UDF, because both the request and the response SOAP documents will easily fit into VARCHAR(32672) with plenty of room to spare.

Open the Data Perspective and create a new Data Development Project (or use the existing project that we created in 7.2, “Using Data Web Services (DWS)” on page 160).

Right-click SQL Scripts and select New → SQL or XQuery Script. Select SQL Editor as the tool for script creation. In the editor that opens, paste the following SQL statement (Example 14-3).

Example 14-3  SQL to call SOAPHTTPNV

```sql
SELECT DB2XML.SOAPHTTPNV(
  ebservice',
  '',
  '<?xml version="1.0" encoding="UTF-8" ?>'
```
Notes on Example 14-3 on page 459:

1. The endpoint URL.
2. The SOAP action. We can leave this parameter blank (but not NULL) in our example.
3. The complete SOAP request document.
4. The correlation name is optional, but it improves readability.

Now, run the statement (by either right-clicking in the Data Projects Explorer view and selecting Run SQL, or by right-clicking in the editor and selecting Run SQL from there).

The result from running the query appears in the Data Output view (Figure 14-2).

To better inspect the output, click the ellipsis button next to the partial output in the SOAP_RESPONSE column. A pop-up window opens (Figure 14-3 on page 461).
As you might have noticed, this output is not the complete output; it is not even a well-formed XML document, because Rational Application Developer by default limits the displayed size of character columns (and the maximum number of result rows to be fetched from a query).

So, in order to see the complete output, we have to modify the size limit. Select Window → Preferences → Data → Output. Set “Maximum bytes to retrieve for character or binary column” to a bigger value, for example, 1000. Run the query again and click the ... detail button again. The output now is displayed in an editor window as an XML tree (Figure 14-4). You can switch to the Source pane to see the actual XML text.
14.5 Parsing the SOAP response message

Now that we have successfully invoked the Web service from DB2, the next challenge is to parse the SOAP response. There are two possibilities:

- Parse the SOAP response XML document in your application program.

  This option is fairly easy to do in Java, but it is much harder in COBOL. IBM Enterprise COBOL does have an XML PARSE verb; however, this verb supports only the Simple API for XML (SAX) style (that is, event-driven style) of XML parsing, making it much harder to navigate the XML document than with tree-style XML parsing.

  **Note:** Additionally, Enterprise COBOL V3.x did not have XML namespace support. This support was introduced with Enterprise COBOL V4.1.

- Use DB2 XML support to parse the SOAP response.

  DB2 comes with a rich set of functionality to create, validate, and parse XML documents. In this section, we use this functionality to parse the SOAP response message from the outbound Web service call, extracting only the parts in which we are interested. By using the DB2-provided XML support, we avoid having to parse the SOAP response in the application program itself.

  Refer to Example 14-4.

  **Note:** A full explanation of the XQuery syntax is not in the scope of this book, but you can obtain more information in *DB2 Version 9.1 for z/OS XML Guide*, SC18-9858, and *DB2 Version 9.1 for z/OS SQL Reference*, SC18-9854.

**Example 14-4** Querying the SOAP response using XMLQUERY

```sql
SELECT XMLQUERY(
'declare namespace soapenv = "http://schemas.xmlsoap.org/soap/envelope/";
declare namespace ews = "http://sg247669.itso.ibm.com/employee/";
soapenv:Envelope/soapenv:Body/ews:getDepartmentResponse/DEPTNAME/text()'
) PASSING XMLPARSE(DB2XML.SOAPHTTPNV(
'',
',
',<xml version="1.0" encoding="UTF-8" ?>
```

462 Enabling z/OS Applications for SOA
Notes on Example 14-4 on page 462:

1. The first argument to XMLQUERY is the XQuery string, optionally preceded by namespace declarations.

   **Note:** The first argument to XMLQUERY must be a string literal. In the listing, we split that string literal into three lines for better readability; in the source code, however, it has to appear as one single line. Using CONCAT will not work, because CONCAT is a function, and again, it has to be a string literal.

2. As just mentioned, the XQuery string can be (and usually needs to be) preceded by namespace declarations of the form:

   ```
   declare namespace prefix = "namespace_uri";
   ```

   Using this syntax, we declare the namespaces for the SOAP envelope and for our own application-specific elements.

3. This expression is the proper XQuery expression. Reading from right to left, we see that it selects the text of the DEPTNAME element, of getDepartmentResponse, in the SOAP body, in the SOAP envelope.

4. The second argument to XMLQUERY is the document to be queried. We pass the result of the SOAPHTTPNV invocation as in Example 14-3 on page 459, converting it to a parsed XML document using the XMLPARSE function.

5. We give the result the correlation name of DEPT_NAME.

   **Tip:** In this example, the result of the XQuery expression was a string that was implicitly casted to a VARCHAR. It gets slightly more complicated if you want to process the result as a number. You can process the result as a number by using the XMLCAST function:

   ```
   SELECT XMLCAST(XMLQUERY(...) AS DECIMAL(19, 4))
   ```
14.6 Creating a wrapper UDF

At this point, the query that we have just developed can be included in an application program. However, for better readability and maintenance, it is much cleaner to hide the details of the SOAPHTTPNV call and the parsing of the SOAP response from the application. Therefore, we encapsulate the query using a wrapper UDF, as shown in Example 14-5.

Example 14-5  Creating the wrapper UDF

```sql
CREATE FUNCTION DEPARTMENT_NAME(DEPTNO VARCHAR(3))
  RETURNS VARCHAR(40)
  LANGUAGE SQL
  CONTAINS SQL
  EXTERNAL ACTION
  NOT DETERMINISTIC
RETURN
XMLCAST(
  XMLQUERY(
    'declare namespace soapenv = "http://schemas.xmlsoap.org/soap/envelope/";
declare namespace ews = "http://sg247669.itso.ibm.com/employee/";
soapenv:Envelope/soapenv:Body/ews:getDepartmentResponse/DEPTNAME/text()'
PASSING XMLPARSE(DB2XML.SOAPHTTPNV(
    'http://wtscz1.itso.ibm.com:8128/EmployeeEmployeeWebServiceWeb/services/Employ eeWebService'
    ,
    />
      <DEPTNO>" || DEPTNO || '</DEPTNO>'
    </ews:getDepartment>
    </soapenv:Body>
  </soapenv:Envelope>'))
AS VARCHAR(40)
)
```

Notes on Example 14-5:

1. The DEPARTMENT_NAME function takes a single argument of type VARCHAR(3).
2. It returns VARCHAR(40).
It is written in SQL, contains SQL, and takes an action outside of DB2 (namely, the SOAP call).

Alternatively, the UDF might have been declared as being DETERMINISTIC, because it returns the same result for the same input parameters (unless someone modifies the data in the underlying table, of course).

As just described, we have to use the XMLCAST function to change the result of XMLQUERY to our function's return type.

During the development of the SQL statement, we hard-coded the department number that we are querying. Of course, here we have to use the value that was passed as a parameter.

This part completes the XMLCAST function invocation.

### 14.6.1 Invoking the wrapper UDF

With all the bits and pieces in place, invoking our Web service via SQL is simple (Example 14-6):

```sql
SELECT DEPARTMENT_NAME('A00') AS DEPT_NAME
FROM SYSIBM.SYSDUMMY1
```

In fact, an application that uses this UDF to retrieve the department name does not even know or care that under the covers, a Web service is called. And, even more importantly, the operation endpoint does not need to be hardcoded in the application program but is centrally maintained. Obviously, this approach results in better maintainability.

### 14.6.2 Retrieving multiple columns from the SOAP response

In the previous examples, we eventually returned only a single value from the call to SOAPHTTPNV. You might have a situation where you need to use several parts of the SOAP response; in our example, you might need both the department name and the manager's ID. It is not a good solution to create two UDFs, because the Web service will be invoked twice.

We present a solution in Example 14-7 on page 466. It uses the XMLTABLE function to convert an XML document (namely, the SOAP response) to tabular format.
**Example 14-7   Converting the SOAPHTTPNV output to tabular format**

WITH RESPONSE AS (  
    SELECT DB2XML.SOAPHTTPNV(  
        'http://wtscz1.itso.ibm.com:8128/EmployeeService/services/EmployeeWebService'  
      ,  
        '<?xml version="1.0" encoding="UTF-8" ?> '  
      ||  
      ||  
        ' xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">'  
      ||  
        '<soapenv:Body>'  
      ||  
        '  <ews:getDepartment>'  
      ||  
        '    <DEPTNO>A00</DEPTNO>'  
      ||  
        '  </ews:getDepartment>'  
      ||  
        '</soapenv:Body>'  
      ||  
        '</soapenv:Envelope>'  
    ) AS SOAP  
    FROM SYSIBM.SYSDUMMY1  
)  
SELECT T.*  
FROM RESPONSE, XMLTABLE(  
    XMLNAMESPACES('http://employee.sg247669.itso.ibm.com' AS "ews",  
                  'http://schemas.xmlsoap.org/soap/envelope/' AS "soapenv"),  
    '$resp/soapenv:Envelope/soapenv:Body/ews:getDepartmentResponse'  
    PASSING XMLPARSE(DOCUMENT SOAP) AS "resp"  
    COLUMNS "DEPT_NAME" VARCHAR(255) PATH 'DEPTNAME',  
           "MGR_NO" VARCHAR(255) PATH 'MGRNO'  
) AS T

Notes on Example 14-7:

1. We use a common table expression to capture the SOAP response.
2. Return all columns from the result table that is returned by the XMLTABLE function.
3. The XMLTABLE function does the work. It expects a namespace declaration, an XQuery expression to select the list of elements that corresponds to the result rows, and a COLUMNS expression that declares the columns to be returned.
4. The XQuery expression selects the element list to be turned into rows.
5. Data from DEPT_NAME column comes from the DEPTNAME child element of each row.
6. Likewise for the MGR_NO column.
7. The correlation name used in 2.
**Note:** Unfortunately, as of DB2 Version 9 for z/OS, there is no way to wrap this SQL as a UDF as easily as in the previous section; currently, SQL functions can be scalar functions only. You have to create a simple external table UDF in COBOL or another supported language.
Using an external Web service in batch applications

In this chapter, we explore two possibilities for consuming Web services in batch applications, specifically:

- Using DB2 outbound Web services
- Calling a Java client from Enterprise COBOL

Because we have already explained DB2 outbound Web services in detail in Chapter 14, “Using DB2 as a Web service requester” on page 455, we only discuss this scenario briefly in this chapter. Our focus in this chapter is on the option to use Enterprise COBOL.
15.1 Introduction

So far, we have only discussed calling Web services from a managed environment, that is, from an application server, such as WebSphere Application Server or CICS Transaction Server. You might be in a situation where you have to invoke an existing Web service from a stand-alone environment, such as a batch application.

Unfortunately, at the time of writing, there is no predefined solution to call a Web service using only pure COBOL.

Therefore, there are really three options:

- Use DB2 outbound Web services, which we briefly described in “Using DB2 outbound Web services” on page 470 and explained in more detail in Chapter 14, “Using DB2 as a Web service requester” on page 455.
- Use WebSphere MQ, with or without an enterprise service bus (ESB). We do not discuss this scenario in this book, because this scenario basically comes down to inserting a message on an MQ queue from a batch job, converting this messages to a SOAP request in an ESB, subsequently converting the SOAP response message back to a plain MQ message, and handing it over again to the batch program. This method is quite a round-trip, and in most batch environments, there is only enough of a window of available time to perform such a round-trip once or twice, but not much more.
- Call Java from COBOL. We focus on this solution in this chapter, because we have not discussed this solution before in any other book. In this solution, we invoke a Java class from the COBOL batch program, and we let this Java class take care of sending and receiving SOAP requests. We discuss this approach in “Calling a Java client from Enterprise COBOL” on page 471.

15.2 Using DB2 outbound Web services

Probably the easiest way to call a Web service from a COBOL (or PL/I) batch application is to use DB2 outbound Web services support, which we discuss in Chapter 14, “Using DB2 as a Web service requester” on page 455. We do not even provide an example here, because to an application program, the Web service invocation is just a regular embedded SQL statement. Specifically, if the SOAP UDF invocation is “wrapped” as a custom user-defined function (as we recommend in 14.6, “Creating a wrapper UDF” on page 464), the Web service call is virtually indistinguishable from calling a scalar function.
15.3 Calling a Java client from Enterprise COBOL

The solution to invoke a Web service from a batch program that we discuss in this section is to use the COBOL/Java integration features that are in IBM Enterprise COBOL. They provide seamless interoperability between those languages, allowing Java objects to be created and used in COBOL without the need to cope with the intricacies of using the Java Native Interface (JNI™) functions. Almost all the details of Java Virtual Machine (JVM) creation, method invocation, and so on are automatically handled by the COBOL compiler and runtime, and you need to use JNI only for special tasks, such as array manipulation and string conversion.

In this section, we use the Java proxy classes that we generated for the VSAM catalog access Web service (6.7, “Creating a Java client for the VsamAccess service” on page 144 and 6.8, “Creating a wrapper service” on page 150).

We exported the generated Java proxy to a Jar file in the z/OS UNIX file system, using the “Export to remote JAR” wizard (File → Export → Other → Remote JAR File). Refer to Figure 15-1.

![Figure 15-1 Exporting the Java client to a remote JAR file](image-url)
We use the classes in that JAR file from a COBOL program (Example 15-1).

**Example 15-1   COBOL program invoking a Java Web service client**

```
Process pgmname(longmixed),lib,thread,dll
*******************************************************************************
Identification Division.
Program-id. "CallWS" is recursive.
Author. Ulrich Seelbach.
*******************************************************************************
Environment Division.
Configuration Section.
Source-Computer. IBM-System-z with debugging mode.
Repository.
  Class CatalogAccessSrvLocator
    is "com.ibm.itso.sg247669.vsam.catalog.CatalogAccessServiceLocator"
  Class CatalogAccess
    is "com.ibm.itso.sg247669.vsam.catalog.CatalogAccess"
  Class CatalogItem
    is "com.ibm.itso.sg247669.vsam.catalog.CatalogItem"
  Class CatalogItemArray
    is "jobjectArray:com.ibm.itso.sg247669.vsam.catalog.CatalogItem"
  Class JavaException
    is "java.lang.Exception"
  Class jstring
    is "java.lang.String"

*******************************************************************************
Data Division.
Working-Storage Section.
01 serviceLocator Usage object reference CatalogAccessSrvLocator.
01 catalogService Usage object reference CatalogAccess.
01 listResult Usage object reference CatalogItemArray.
01 item Usage object reference CatalogItem.
01 exc Usage object reference JavaException.
01 jstr Usage object reference jstring.
01 listResultLength Pic S9(9) comp-5.
01 Item-Data.
  10 Ref-ID Pic S9(9) comp-5.
  10 Description Pic X(40).
01 listIdx Pic S9(9) comp-5.
Linkage Section.
```
Copy JNI suppress.

******************************************************************
Procedure Division.
Mainline Section.
Perform Init-JNI
D    Display "Getting service locator"
    Invoke CatalogAccessSrvLocator new
        returning serviceLocator
    Perform Exception-check
D    Display "Getting service endpoint"
    Invoke serviceLocator 'getCatalogAccess'
        returning catalogService
    Perform Exception-check
D    Display "Invoking list operation on service"
    Invoke catalogService 'list'
        using by value 0
        0
        returning listResult
    Perform Exception-check
D    Display "Getting result length"
    Call GetArrayLength
        using by value JNIEnvPtr
            listResult
        returning listResultLength
    Perform Exception-check
    Display listResultLength " record(s) returned."
    Display "----------------------------------------"
Perform Display-all-items
Move 0 to Return-Code
Stop run
.
Init-JNI Section.
D    Display "Initializing JNI"
    Set Address of JNIEnv to JNIEnvPtr
    Set Address of JNINativeInterface to JNIEnv
.
Display-all-items Section.
Perform varying listIdx from 0 by 1
    until listIdx = listResultLength
    Call GetObjectArrayElement
        using by value JNIEnvPtr
            listResult
            listIdx
        returning item
Enabling z/OS Applications for SOA

Perform Exception-check
Perform Display-item
End-perform
.

Display-item Section.
Invoke Item 'getWsItemRef'
  returning Ref-ID
Invoke Item 'getWsDescription'
  returning jstr
Call 'GetStringPlatform'
  using by value JNIEnvPtr
    jstr
    address of Description
    length of Description
    0
Perform Exception-check
Display Ref-ID ": " Description
.

Exception-check Section.
Call ExceptionOccurred
  using by value JNIEnvPtr
  returning exc
If exc not equal to null then
Call ExceptionClear
  using by value JNIEnvPtr
Display "Java exception caught"
Invoke exc 'printStackTrace'
Move 16 to Return-Code
Stop run
End-if
.

Notes on Example 15-1 on page 472:
1 The “Repository” paragraph identifies all classes that are going to be used in
the program and defines the association between the external class name
(that is, the fully qualified Java class name in this case) and the name that is
used in the program.
2 Declare references to instances of various classes.
3 Initialize the JVM environment special registers.
Create a new instance of class CatalogAccessSrvLocator and assign a reference to that instance to serviceLocator, which is equivalent to the following Java statement:

```java
serviceLocator = new CatalogAccessServiceLocator();
```

Check for exceptions thrown from the previous statement.

Invoke method `getCatalogAccess` on serviceLocator and assign the result to catalogService, which is equivalent to:

```java
catalogService = serviceLocator.getCatalogAccess();
```

This call is the actual Web service call. It invokes the `list` method with two parameters and assigns the result to listResult, which is equivalent to the following Java statement:

```java
listResult = catalogService.list(0, 0)
```

The result from the list invocation is a Java array. To find out the length of that array, we call the respective JNI method.

Iterate over all elements of the result array.

Get the element at position listIdx, using a JNI function.

To display an item's data, we need to invoke the respective getter method. Method `getWsItemRef` returns a Java int, which maps to a COBOL PIC9(5) item (with a USAGE clause specifying BINARY or one of its variants) without any special conversion.

Method `getWsDescription`, however, returns a Java string.

We need to convert the Java string to EBCDIC using a JNI function.

Display the current item.

Check whether the last Java method call (or JNI call) caused an exception.

If so, print the stacktrace and terminate the program.

To compile the program, we used the following shell script (Example 15-2 on page 476). It copies the source code from the MVS source library to a z/OS UNIX file, and then, it invokes the COBOL compiler and the binder from z/OS UNIX via the `cob2` command.

**Note:** In order to compile the program in the traditional way, from MVS batch, you have to copy the `JNI.cpy` copybook from the z/OS UNIX filesystem to an MVS data set. The COBOL compiler cannot process z/OS UNIX files when invoked from MVS batch.
Example 15-2  Shell script to compile the COBOL program

```bash
#!/bin/sh
IGY_HOME=/usr/lpp/cobol
JAVA_HOME=/usr/lpp/java/J5.0

cp "/'ulrich.source.cobol(callws)" ./callws.cbl

$IGY_HOME/bin/cob2 -g \
   -o "/'ulrich.load.lib(callws)" \
   -I$IGY_HOME/include \
   callws.cbl \
   $JAVA_HOME/bin/j9vm/libjvm.x \
   $IGY_HOME/lib/igzcvjava.x
```

To run the program, we used the following JCL, as shown in Example 15-3.

Example 15-3  JCL to invoke the COBOL/Java sample program

```bash
//COBWS JOB 'COBOL -> Java -> Web service',SEELBACH, //       NOTIFY=&SYSUID,MSGLEVEL=(1,1),REGION=0M //CALLWS EXEC PGM=CALLWS, // PARM='/_CEE_ENVFILE=/u/ulrich/cobenv),POSIX(ON),XPLINK(ON)' //STEPLIB DD DISP=SHR,DSN=ULRICH.LOAD.LIB //       DD DISP=SHR,DSN=CEE.SCEERUN //       DD DISP=SHR,DSN=CEE.SCEERUN2 //       DD DISP=SHR,DSN=BB061Z1.SBBOLD2 //       DD DISP=SHR,DSN=BB061Z1.SBBLOAD //       DD DISP=SHR,DSN=BB061Z1.SBBGLOAD //JAVAOUT DD PATH='/tmp/ulrich.javaout', //       PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=(SIRUSR,SIWUSR) //JAVAERR DD PATH='/tmp/ulrich.javaerr', //       PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=(SIRUSR,SIWUSR)
```

Notes on Example 15-3:

1. Specify the Language Environment (LE) parameters for the program. 
   _CEE_ENVFILE_ points to a z/OS UNIX file containing environment variable 
   settings. POSIX(ON) and XPLINK(ON) are mandatory for COBOL main 
   programs calling Java.

   For more information, refer to _Enterprise COBOL for z/OS Programming 
   Guide_, SC23-8529.

2. In addition to our own load library, we need the Language Environment run 
   time and several WebSphere native libraries in the STEPLIB concatenation. 
   The native libraries are used by the WebSphere thin client library.
Set up z/OS UNIX files for Java System.out output.

Set up z/OS UNIX files for Java System.err output.

The following example (Example 15-4) shows the LE environment file referenced by the previous JCL. Note that this file is not a shell script but more like a properties file, so you cannot use the familiar LIBPATH=$LIBPATH:/what/ever idiom. Also, each environment setting must be on one unbroken line with no internal blanks.

The most important setting in the LE environment file is the CLASSPATH for the Java Virtual Machine, referencing the Web service client JAR file that we created and exported to the host earlier (refer to page 471) and the Web Services thin client run time.

Using the COBJVMINITOPTIONS variable, you can pass options to the JVM that is started by the COBOL run time. In this case, we specify a Java system property pointing to the SSL configuration file, similar to what we did in 6.7, “Creating a Java client for the VsamAccess service” on page 144 on page 149.

Note: Actually, we cheated a little bit. We were unable to successfully connect to the Secure Sockets Layer (SSL) endpoint, apparently due to SSL configuration problems, so we changed the default endpoint to HTTP before exporting the Jar file (by modifying the value of catalogAccess_address in class CatalogAccessServiceLocator).

However, without the com.ibm.SSL.ConfigURL setting, the connection attempt failed at an earlier stage, exactly as it did in 6.7, “Creating a Java client for the VsamAccess service” on page 144 on page 149.

The program produced the following output, on SYSOUT (Example 15-5 on page 478), and therefore, was a success.
Example 15-5  Output from running the COBOL/Java Web service client

Initializing JNI
Getting service locator
Getting service endpoint
Invoking list operation on service
Getting result length
0000000020 record(s) returned.

-----------------------------------------------
0000000020: Ball Pens Blue 24pk
0000000030: Ball Pens Red 24pk
0000000040: Ball Pens Green 24pk
0000000050: Pencil with eraser 12pk
[... omitted ...]
0000000210: Clear sticky tape 5pk

-----------------------------------------------
Appendixes
Defining data used in samples

In this appendix, we show the JCL for creating and loading the catalog VSAM file and the Data Definition Language (DDL) to create the DEPT sample table.
JCL for creating and loading the catalog VSAM file

Example A-1 is the JCL to create and load the catalog VSAM file that we use in several examples in this book. To run the JCL, edit the jobcard and change the @dsindex@ placeholder to a valid high-level qualifier for the dataset.

Example: A-1 JCL to create and load the catalog VSAM file

```
//DFH$ECAT JOB ,NOTIFY=&SYSUID
//********************************************************************
//*                                                                   *
//* JCL NAME = DFH$ECAT                                               *
//*                                                                   *
//* DESCRIPTIVE NAME = Web Services example application               *
//*                                                                   *
//* FUNCTION =                                                        *
//*    This is sample JCL to build the catalog data file required     *
//*    by the base application of the web services example            *
//*    application                                                    *
//*                                                                   *
//********************************************************************
//DEFKSDS EXEC PGM=IDCAMS,REGION=64M,COND=(0,LT)
//SYSPRINT DD SYSOUT=*                                               *
//AMSDUMP  DD SYSOUT=*                                               *
//SYSIN DD *

    DELETE @dsindex@.EXMPLAPP.EXMPCAT PURGE CLUSTER
    SET MAXCC=0
    DEFINE CLUSTER (NAME(@dsindex@.EXMPLAPP.EXMPCAT)-
                    TRK(1 1) -
                    KEYS(4 0) -
                    RECORDSIZE(80,80) -
                    SHAREOPTIONS(2 3) -
                    INDEXED -
                    ) -
        DATA (NAME(@dsindex@.EXMPLAPP.EXMPCAT.DATA) -
            ) -
        INDEX (NAME(@dsindex@.EXMPLAPP.EXMPCAT.INDEX) -
            ) -

    //*
    //CPYKSDS EXEC PGM=IDCAMS,REGION=64M,COND=(0,LT)
    //SYSPRINT DD SYSOUT=*                                               *
    //AMSDUMP  DD SYSOUT=*                                               *
    //SYSIN DD *

    REPRO INFILE(INFILE) OUTFILE(OUTFILE)
    //OUTFILE DD DSN=@dsindex@.EXMPLAPP.EXMPCAT,DISP=SHR
```
DDL to create the DEPT sample table

Example A-2 is a trimmed-down version of the DDL to generate the DEPT sample table. We removed foreign key constraints to other tables in the DB2 sample database to make it self-contained. Be sure to verify the BUFFERPOOL and STOGROUP settings.

Example: A-2  Simplified DDL to create the DEPT sample table

```sql
CREATE TABLE DEPT (  
    DEPTNO CHAR(3) NOT NULL,  
    DEPTNAME VARCHAR(36) NOT NULL,  
    MGRNO CHAR(6) WITH DEFAULT NULL,  
    ADMRDEPT CHAR(3) NOT NULL,  
    LOCATION CHAR(16) WITH DEFAULT NULL  
)  
AUDIT NONE  
DATA CAPTURE NONE  
CCSID EBCDIC;
```
CREATE UNIQUE INDEX XDEPT1 ON DEPT (DEPTNO ASC)
    NOT CLUSTER
    NOT Padded
    USING STOGROUP SYSDEFLT
        PRIQTY 12
        SECQTY -1
        PCTFREE 10
        BUFFERPOOL BP0
        DEFER NO
        COPY NO
        PIECESIZE 2097152 K;

ALTER TABLE DEPT ADD CONSTRAINT DEPTNO PRIMARY KEY (DEPTNO);
Additional material

This book refers to additional material that can be downloaded from the Internet as described next.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks publications Web server. Point your Web browser at:

ftp://www.redbooks.ibm.com/redbooks/SG27669

Alternatively, you can go to the IBM Redbooks publications Web site at:

ibm.com/redbooks

Select the Additional materials and open the directory that corresponds with the IBM Redbooks form number, SG247669.

System requirements for downloading the Web material

There are no specific requirements to download and store the material on your workstation. To be able to use, however, the materials in your workstation tools,
such as Rational Developer for System z, you need to meet the minimum system requirements for the specific tool that you plan to use.

For our sample development, we ran all tools inside a VMWare image. If you plan to do the same thing, we recommend 3 GB of RAM or more. Allocate about 50% of RAM to your VMWare image, and leave the other 50% for your primary Windows systems. Note that a VMWare image can grow in size considerably. You have to expect a minimum size of about 7 GB of disk space for a very small image with one tool up to more than 30 GB for an image with a complex development environment.

Using the Web material

The additional Web material that accompanies this book is explained in the following sections. The material is supplied per chapter/scenario.

Exposing VSAM data as Web Services

The additional material for this scenario is packaged in a zip file. It includes z/OS components, such as JCL, a Java jar file needed for JZOS, and a workspace that can be used in a shared development environment with both the Data Studio Developer Version 1.2 Toolkit and Rational Developer for System z.

The name of the zip file is:
Exposing VSAM and DB2 as a Web service.zip

Exposing data as a service by using IBM WebSphere Message Broker

The additional material for this scenario is supplied as a single file, namely, a workspace file for the WebSphere Message Broker Toolkit Version 6.1.03. This workspace file, when loaded, includes the entire solution.

The name of the workspace file is:
Exposing data as a Web service using WMB.zip

Exposing DB2 data as a service

The additional material for this scenario is packaged in one zip file. It includes z/OS components, such as JCL, a Java jar file needed for JZOS, and a workspace that can be used in a shared development environment with both the Data Studio Developer Version 1.2 Toolkit and Rational Developer for System z.
The name of the zip file is:
Exposing VSAM and DB2 as a Web service.zip

Exposing IMS data as a service

All components referred to in the chapter are included in a zip file. This zip file includes both z/OS components, such as the database descriptor (DBD) and the program specification block (PSB), and workstation components, such as workspace file for RDz.

The name of the zip file is:
Exposing IMS data as a Web service.zip

Exposing an IMS application as a Web service

All components referred to in the chapter are included in a zip file. This zip file includes both z/OS components and workstation components, such as a workspace file for RDz.

The name of the zip file is:
Exposing an IMS application as a Web service.zip

Integrating an external Web service with a CICS application

The material for this scenario includes:

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>README.txt</td>
<td>Instructions to set up sample</td>
</tr>
<tr>
<td>External WebService</td>
<td>Source code of the application that needs to be integrated</td>
</tr>
<tr>
<td>CICS source.zip</td>
<td></td>
</tr>
<tr>
<td>ExternalWebService</td>
<td></td>
</tr>
<tr>
<td>CICS_RDz</td>
<td>Workspace containing RDz-generated artifacts</td>
</tr>
<tr>
<td>Workspace.zip</td>
<td></td>
</tr>
</tbody>
</table>

Integrating an external Web service into an IMS application

The material for this scenario includes:
Using DB2 as a Web service requester

The additional material for this scenario is packaged in one zip file. It includes z/OS components, such as JCL, a Java jar file needed for JZOS, and a workspace that can be used in a shared development environment with both the Data Studio Developer Version 1.2 Toolkit and Rational Developer for System z.

The name of the zip file is:
Exposing VSAM and DB2 as a Web service.zip

Using an external Web service in batch applications

The additional material for this scenario is packaged in the same zip file as the data-related scenarios. The name of the zip file is:
Exposing VSAM and DB2 as a Web service.zip
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 publications

For information about ordering these publications, refer to “How to get IBM Redbooks publications” on page 490. Note that several of the documents referenced here might be available in softcopy only:

- Application Development for CICS Web Services, SG24-7126
- Implementing CICS Web Services, SG24-7206
- Securing Access to CICS Within an SOA, SG24-5756
- DB2 for z/OS and OS/390: Ready for Java, SG24-6435
- WebSphere Service Registry and Repository Handbook, SG24-7386
- Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB, SG24-7369
- Implementing an ESB using IBM WebSphere Message Broker V6 and WebSphere ESB V6 on z/OS, SG24-7335
- WebSphere for z/OS V6 Connectivity Handbook, SG24-7064
- WebSphere Business Integration Message Broker Basics, SG24-7090
- Distributed Functions of DB2 for z/OS and OS/390, SG24-6952

Other publications

These publications are also relevant as further information sources:

- The Value of IBM System z and z/OS in Service-Oriented Architecture, REDP-4152
- C/C++ Run-Time Library Reference, SA22-7821
- C/C++ Programming Guide, SC09-4765
- Enterprise COBOL for z/OS Language Reference, SC23-8528
How to get IBM Redbooks publications

You can search for, view, or download IBM Redbooks publications, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy IBM Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads
ibm.com/support

IBM Global Services
ibm.com/services
Index

A
Access Services 16
ALTERNate Program Control Block (ALTPCB) 421
Application Portfolio Manager (APM) 20

B
Basic Mapping Support (BMS) 56
Business Application Services 16
Business Innovation and Optimization Services 18
Business Process Execution Language (BPEL) 12

C
CICS Interdependency Analyzer (IA) 77
CICS Service Flow Feature (SFF) 308
   Service Flow Modeler (SFM) 308
   Service Flow Runtime (SFR) 308
CICS Web Services 59
CICS Web Services Assistant 31
COBOL, on the mainframe 9
Component Business Modeling (CBM) 6, 13
Container-Managed Persistence Entity JavaBeans 43

D
Data Web Services (DWS) 41, 160–161
Database Resource Adapter (DRA) 180–181
DataSource facility 191
DB2
   User Defined Functions (UDFs) 456
      XML parsing 456
DB2 on z/OS
   location name, finding 162
DB2, outbound Web Services
   User Defined Functions (UDFs)
      batch program 470
      parameters 456
      size of return message 456
      SOAP message, creating 458
      SOAP message, example 458
      SOAP response message, parsing 462
      SOAP return message, retrieving multiple parts 465
SOAPHTTPNC 456
SOAPHTTPNV 456
SQL, creating 459
wrapper UDF, creating 464
wrapper UDF, invoking 465
Development Services 18
DFSCILIB 183
Discovery tools 19
Distributed Remote Data Access (DRDA) 39
DLIModel utility 51, 201
DML statements 161
DRA startup table 181

E
Enterprise Service Bus (ESB) 6, 12
Enterprise Service Tools (EST) 290, 308
Enterprise transformation 19
Enterprise transformation strategy 20
   adapt 22
   improve 22
   innovate 22
Entity EJB 160

H
Hibernate 43

I
IBM Data Studio Developer 160
   connection, creating 162
   SQL statement, creating 163
   SQL statement, testing 163
   Web service, creating 164
   Web service, testing 169
IBM Data Studio Developer*Web service, deploying 168
IBM Rational Asset Analyzer (RAA) 77
IBM SOA reference architecture 14
IMS
   asynchronous callout 419
      Database Resource Adapter (DRA) 180–181
      Open Database Access (ODBA) 180
IMS asynchronous callout 115
IMS Connect
  hold queue name 421
  XCF member name 421
  XML Adapter name 421
IMS DB Resource Adapter
  configuring, in WAS 184
  installing 183
  RAR file, installing in WAS 183
  STEPLIB in WAS 183
IMS hierarchical database interface 180
IMS IVP for WAS Java environment 190
IMS ODBA
  custom service in WAS 186
IMS SOAP Gateway 115, 290
  Deployment utility 300
IMS SOAP Gateway resources
  correlator file 298
IMS SOAP Gateway, RDz version required 290
IMS TM Resource Adapter 84
IMS, Java class libraries 183
imsDBJCA.rar 184
imsjavaBase.jar 184
imsJDBC.jar 184
imsXQuery.jar 184
Information as a service 27
Information Services 15
initial state 25
Interaction Services 15
iogdeploy.sh 300
IT Service Management 18

J
Java DataBase Connectivity (JDBC) 160
Java Native Interface (JNI) 471
JZOS 37
  com.ibm.jzos.fields package 150
  JZOS launcher 130
  JZOS library 130
  ZFile class 130
JZOS library 130
JZOS record generator 151

K
Key Performance Indicator (KPI) 6

L
libJavTDLI.so 189

M
monolithic 2

O
Open DataBase Aaccess (ODBA) 180

P
Partner Services 16
Point-to-point connectivity 12
process choreography 6
Process Services 15
pureQuery 42–43, 160

Q
Quality of Service 14

R
Rational Asset Analyzer (RAA) 20
Rational Host Access Transformation Services
  (HATS) 100
Redbooks Web site 490
  Contact us xvii

S
SDFSJLIB data set 183
SDFSRESL data set 183
server.policy file, in WAS for z/OS 187
service granularity 5
service, coarse-grained 5
service, fine-grained 5
Service-Oriented Modeling and Architecture (SO-
  MA) 6
SOA approach 13
  Bottom-up 13
  Meet-in-the-middle 13
  Top-down 13
SOA lifecycle 11
  Assemble 11
  Deploy 11
  Manage 11
  Model 11
SOA on System z, advantages 14
SOAP Gateway Administrative Console 305

T
TMEMBER 421
Total Cost of Ownership  14

**U**
User Defined Functions (UDFs)  456

**V**
VSAM KSDS  130

**W**
Web Services Description Language (WSDL)  290
WebSphere Asset Analyzer (WSAA)  20
WebSphere Message Broker (WMB)  34
  Compute node  45
  Java Compute node  45
  message flow  34
  VSAM nodes  34
    VSAMDelete  34
    VSAMInput  34
    VSAMRead  34
    VSAMUpdate  34
    VSAMWrite  34
WebSphere Message Broker Toolkit  36

**X**
XML parsing, in COBOL  462
XML parsing, using DB2  462
XQuery  161

**Z**
ZOS Record Generator  129
Enabling z/OS Applications for SOA
Enabling z/OS Applications for SOA

The objective of this IBM Redbooks publication is to demonstrate ways of enabling existing applications and data on z/OS for integration into a service-oriented architecture (SOA). The focus is on solutions using Web Services as the underlying technology for this integration.

This book describes a variety of patterns that can be applied to enabling z/OS applications for service-oriented architecture. Many of these patterns are explained by means of a sample. Each sample is included in the additional material of this book.

This book is intended for readers who are interested in extending the function of their existing applications in order to benefit from SOA.

Learn how to use patterns to help you modernize your applications

Use our samples of a selection of patterns to get started

Understand integration and modernization

For more information: ibm.com/redbooks

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.