Getting Started on Integrating Your Information

Understand information integration business issues

Learn by example how to implement integrated solutions

Enable data federation, Web Services, and XML

Corinne Baragoin
Joachim Dirker
Carolyn Elkins
Ian Harvey
Fred Lo

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

First Edition (February 2003)

The examples in this edition were performed using IBM DB2 Universal Database Version 8.1, IBM WebSphere MQ Version 5.3. They also apply to IBM DB2 Information Integrator Version 8.1.

© Copyright International Business Machines Corporation 2003. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
# Contents

Figures ................................................................. ix  
Tables ................................................................. xiii  
Examples .............................................................. xv  
Notices ................................................................. xix  
Trademarks ............................................................ xx  
Preface ................................................................ xxii  
The team that wrote this redbook ................................ xxii  
Become a published author ........................................ xxiii  
Comments welcome ...................................................... xxiv  

Chapter 1. Introduction .............................................. 1  
1.1 The business climate ............................................. 2  
1.2 Business challenges and information integration ............ 2  
1.3 Overview of information integration ............................ 4  
1.3.1 What is information integration? ................................. 5  
1.4 WebSphere business and information integration .......... 7  
1.4.1 WebSphere business integration overview .................. 7  
1.4.2 Traditional business integration ............................... 9  
1.4.3 How information integration fits with business integration .... 13  

Chapter 2. Information integration infrastructure ............... 17  
2.1 Information integration design objectives and features .... 18  
2.2 Information integration layers .................................... 25  
2.2.1 Why DB2 UDB? .................................................. 26  
2.2.2 Why message queueing? ......................................... 29  
2.2.3 Why XML? ....................................................... 34  
2.2.4 Why Web Services? ............................................. 44  
2.2.5 Why data warehousing? ......................................... 63  
2.3 Information integration at HLED Investments ................ 65  
2.4 Target environment ............................................... 68  
2.4.1 Technical infrastructure: Business Case I .................. 70  
2.4.2 Technical infrastructure: Business Case II .................. 71  
2.4.3 Technical infrastructure: Business Case III ................. 72  
2.4.4 Technical Infrastructure: Business Case IV ................. 73
# Chapter 3. Business case I: Federating data access using SQL

3.1 Business problem statement ............................................. 78
3.2 Solution overview ............................................................ 80
3.3 Infrastructure and technology solution ................................. 81
3.4 Step-by-step implementation ............................................. 82
  3.4.1 Implement the DB2 federated server .............................. 83
  3.4.2 Create federated database .......................................... 84
  3.4.3 Set up DB2 UDB data source ...................................... 84
  3.4.4 Set up Informix data source ...................................... 86
  3.4.5 Create the federated view ......................................... 89
  3.4.6 Install and configure WebSphere MQ ............................ 90
  3.4.7 Install and configure MQSeries AMI ............................ 92
  3.4.8 Add MQSeries Integration to the database ...................... 99
  3.4.9 Test MQSeries Integration ........................................ 100
  3.4.10 Install and configure z/OS WebSphere MQ and CICS resources .................................................. 100
  3.4.11 Develop SQL statements to access the information ........ 101
  3.4.12 Develop the GETPORTFOLIO stored procedure .............. 104
  3.4.13 Verify the test results ........................................... 106
  3.4.14 Alternatives ....................................................... 106
3.5 Further steps .................................................................. 110
  3.5.1 DB2 Information Integrator ........................................ 110
  3.5.2 Federated access to Oracle ........................................ 110
  3.5.3 Set up Oracle data source ........................................ 111
3.6 Integration considerations .............................................. 115
  3.6.1 Staffing requirements ............................................... 115
  3.6.2 Availability ......................................................... 116
  3.6.3 Maintenance ....................................................... 116
  3.6.4 Performance and capacity planning ............................. 116
  3.6.5 Security ............................................................. 117
3.7 Conclusion ................................................................ 118

# Chapter 4. Business case II: Accessing your information through Web Services

4.1 Business problem statement ............................................. 121
4.2 Assumptions ................................................................ 122
4.3 Solution overview ............................................................ 122
4.4 Infrastructure and technology solution ................................. 125
4.5 Step-by-step: Building DB2 Web Services ......................... 129
  4.5.1 Build Web Services using WORF ................................. 130
  4.5.2 Build Web Services with WebSphere Studio Application Developer .................................................. 171
4.6 Further steps ................................................................ 187
4.7 Integration considerations .............................................. 188
  4.7.1 Does Web Services mean interoperability? ..................... 188
System requirements for downloading the Web material ............... 352
How to use the Web material ............................................ 352

Abbreviations and acronyms ............................................ 355

Related publications ..................................................... 357
IBM Redbooks ............................................................. 357
Other resources ........................................................... 357
Referenced Web sites ..................................................... 358
How to get IBM Redbooks .............................................. 359
IBM Redbooks collections ............................................ 359

Index ............................................................................. 361
Getting Started on Integrating Your Information
Figures

1-1  WebSphere business integration ................................. 8
1-2  Enterprise integration framework ............................. 9
1-3  Middleware product implementation categories .............. 10
1-4  Integration communication styles ............................. 12
1-5  Application to data integration ............................... 15
1-6  Application to data integration with IBM information integration 16
2-1  Integrating information design stack ........................ 19
2-2  IBM information integration platform ......................... 25
2-3  Changing scope of data management .......................... 27
2-4  MQSeries Integration components ............................. 32
2-5  DB2 UDB, WebSphere MQ, and WMQI integrated environment 34
2-6  Sample XML document ........................................... 35
2-7  XML Extender components ...................................... 37
2-8  XML data source integration with DB2 UDB .................. 38
2-9  XML Storage options in DB2 UDB .............................. 39
2-10 Service Oriented Architecture ................................. 46
2-11 Web Services layered architecture ............................ 49
2-12 Web Services integration ....................................... 61
2-13 DB2 UDB as Web Services provider: implementation options 62
2-14 IBM Data Warehouse Center ...................................... 64
2-15 HLED technical environment - Business Case I ............. 70
2-16 HLED technical environment - Business Case II ........... 71
2-17 HLED technical environment - Business Case III ........... 72
2-18 System environment for the CDS scenario .................... 73
2-19 Runtime environment ........................................... 74
3-1  Current datastore connections .................................. 79
3-2  Solution Datastore connections ................................. 81
3-3  Setnet32 entries .................................................. 87
3-4  Registry Editor example ......................................... 88
3-5  AMI Administration Tool initial panel ........................ 94
3-6  IIR.BC1A.Sender .................................................. 95
3-7  IIR.BC1A.RECEIVER ............................................... 96
3-8  Initialization ....................................................... 97
3-9  Send Tab ........................................................... 98
3-10 Receive Tab ......................................................... 99
3-11 Process_Portfolio request flow ................................ 108
3-12 Process_Portfolio reply flow .................................. 109
3-13 SQL PLus .......................................................... 113

© Copyright IBM Corp. 2003. All rights reserved.
3-14 Read the Oracle table through the federated server ................. 114
4-1 Physical location of data ......................................... 123
4-2 Data schemas with their respectively data sources .................. 124
4-3 WORF architecture ................................................. 126
4-4 Installing the sample services Web application ...................... 134
4-5 Web Services sample page ........................................ 135
4-6 XML Extender Administration logon screen .......................... 139
4-7 XML Extender Administration: Select a Task ........................ 140
4-8 XML Extender Administration: Import a DTD ........................ 142
4-9 XML Extender Administration: Import a DTD confirmation .......... 142
4-10 XML Extender Administration: Edit a DAD ........................ 143
4-11 XML Extender Administration: editing DAD ......................... 147
4-12 Mapping your dtdid between your namespace and DAD file ......... 153
4-13 The processing flow of store stock/mutual fund information request . 155
4-14 Test client for stockInfo.dadx ................................... 156
4-15 Providing input to the storeStockMutualFundInfo screen ............ 157
4-16 Result from invoking the storeStockMutualFundInfo method ........ 158
4-17 Web Service method makeMQRequest sample results ............... 163
4-18 Web Service method viewMyPortfolio sample result ................ 166
4-19 Web Service method viewRecentTransactions sample result ......... 168
4-20 Web Service method makeAnOrder sample results .................. 170
4-21 Create an SQL statement using Application Developer .............. 175
4-22 Specifying a database connection .................................. 176
4-23 DADX group properties configuration ............................. 177
4-24 DADX generation ................................................ 178
4-25 Generated DADX file, StockInfo.dadx ................................ 179
4-26 Sample client proxy code (extract) .................................. 181
4-27 Testing the Web service using the sample ...................... 182
4-28 Deploying StockInfoApp in WebSphere Application Server ......... 184
4-29 StockInfo application running under WebSphere Application Serve 185
4-30 StockInfo Web Services running as its own application .......... 187
5-1 Alternative 1 flow .............................................. 197
5-2 Alternative 2 flow ............................................... 198
5-3 Alternative 3 flow ............................................... 199
5-4 Enable_db command .............................................. 201
5-5 Database stored procedures after enable_db ...................... 201
5-6 ClientTransaction.xml ........................................... 202
5-7 Stock_MutualFund_info.dtd ..................................... 202
5-8 Stock_MutualFund_info.dad ..................................... 203
5-9 Shred command ................................................ 204
5-10 Updated Stock_MutualFund_Info table ............................ 205
5-11 Source database for SQL/XML generated XML document ............ 206
5-12 SQL/XML generated XML document without XML header ........... 207
Tables

1-1 Business requirements and information integration technical challenges 3
2-1 MQSeries XML functions ................................................. 42
2-2 The SQL/XML functions. .................................................. 43
2-3 MQSeries text functions .................................................. 44
2-4 SOAP namespaces ........................................................ 51
2-5 WSDL namespaces ........................................................ 57
2-6 UDDI namespaces .......................................................... 59
3-1 Basic information integration implementation steps .................. 83
3-2 Federated access to DB2 UDB implementation steps ............... 84
3-3 Informix server network information .................................. 84
3-4 Informix implementation steps ......................................... 86
3-5 Informix server information ............................................. 86
3-6 Environment variables .................................................... 87
3-7 WebSphere MQ implementation steps ................................ 90
3-8 QM_COPPER definitions ................................................ 91
3-9 MQV1 definitions ........................................................ 91
3-10 Test MQSeries Integration .............................................. 100
3-11 CICS implementation steps ............................................. 100
3-12 Informix implementation steps ....................................... 111
3-13 Oracle server information .............................................. 111
4-1 DB2 Web Services step-by-step implementation summary .......... 130
4-2 Tasks summary for installing WORF .................................. 132
4-3 Steps to create and populate the data sources ....................... 136
4-4 Stock and mutual fund portfolio data and their data stores ....... 137
4-5 Orders table and its data store location ............................... 137
4-6 Tasks summaries for build XML collection and a DAD file ....... 137
4-7 Elements properties for stock_mutualFund_info XML document .. 146
4-8 Steps to create DADX group summary ................................ 150
4-9 Properties in group.properties file ................................... 151
4-10 Creating the DADX file outline ...................................... 153
4-11 Input values to test Web Service method makeAnOrder .......... 170
4-12 DB2 Web Services using WebSphere Studio implementation ...... 172
5-1 Alternatives diagram ..................................................... 195
5-2 Alternative one: implementation steps ............................... 200
5-3 Mapping of XML elements to the STOCK_MUTUALFUND_INFO table 204
5-4 Alternative two: implementation steps ............................... 205
5-5 Alternative three: implementation steps ............................. 209
6-1 Common Data Store setup: basic steps ............................... 232

© Copyright IBM Corp. 2003. All rights reserved.
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-2</td>
<td>DB2 UDB V8.1 server information</td>
<td>236</td>
</tr>
<tr>
<td>6-3</td>
<td>Informix server information</td>
<td>238</td>
</tr>
<tr>
<td>6-4</td>
<td>DB2MQ-CV_MQM definitions</td>
<td>242</td>
</tr>
<tr>
<td>6-5</td>
<td>MQV1 definitions</td>
<td>242</td>
</tr>
<tr>
<td>B-1</td>
<td>WebSphere MQ implementation steps</td>
<td>293</td>
</tr>
<tr>
<td>B-2</td>
<td>Queue Manager values</td>
<td>295</td>
</tr>
<tr>
<td>B-3</td>
<td>Remaining local queues</td>
<td>301</td>
</tr>
<tr>
<td>B-4</td>
<td>Remote queue information</td>
<td>302</td>
</tr>
<tr>
<td>B-5</td>
<td>WebSphere MQ z/OS definitions</td>
<td>307</td>
</tr>
<tr>
<td>C-1</td>
<td>CICS Resources List</td>
<td>328</td>
</tr>
<tr>
<td>E-1</td>
<td>List of stock/mutual fund information XML files</td>
<td>344</td>
</tr>
</tbody>
</table>
Examples

2-1 Specifying an XPath ................................. 38
2-2 Enable_MQXML command ............................... 42
2-3 Enable_MQFunctions command ......................... 44
2-4 SOAP message structure .............................. 50
2-5 SOAP message example: Sample request ............... 51
2-6 WSDL part I: namespace and type element .......... 53
2-7 WSDL part II: message element ....................... 54
2-8 WSDL part III: port type element ................... 55
2-9 WSDL part IV: binding element ....................... 56
2-10 WSDL part V: service and port elements .......... 57
2-11 Create Stock portfolio .............................. 65
2-12 Stock portfolio sample data ......................... 65
2-13 Create mutual fund portfolio ......................... 65
2-14 Mutual fund sample data ............................. 66
2-15 Create orders table ................................. 66
2-16 MQSeries Integration example ....................... 75
3-1 Current data access code flow ....................... 78
3-2 New data access code flow - alternative 1 .......... 82
3-3 New data access code flow - alternative 2 .......... 82
3-4 Create the federated database ...................... 84
3-5 Catalog remote DB2 UDB source node ................. 84
3-6 Catalog remote DB2 UDB database ................... 85
3-7 Create federated object for remote DB2 UDB source 85
3-8 Services file extract ................................ 88
3-9 Create federated object to Informix remote source 89
3-10 Create investments view ............................. 90
3-11 Customer information request copybook .......... 102
3-12 Customer Information reply copybook ............... 103
3-13 SQL statements example ............................ 103
3-14 GETPORTFOLIO stored procedure .................. 104
3-15 GETPORTFOLIO request ............................ 105
3-16 Test the request ................................... 106
3-17 Code flow to invoke WMQI ............................ 107
3-18 Complete portfolio request DTD ................. 107
3-19 MQSENDXML sample ............................... 107
3-20 Oracle client tnsnames.ora file .................... 112
3-21 Using SQLPlus interactive tool .................... 112
3-22 Create federated object to Oracle data source .... 114
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-2</td>
<td>Sample date bytes 1-71</td>
<td>313</td>
</tr>
<tr>
<td>C-3</td>
<td>Sample data bytes 72-143</td>
<td>313</td>
</tr>
<tr>
<td>C-4</td>
<td>Sample data bytes 144-231</td>
<td>313</td>
</tr>
<tr>
<td>C-5</td>
<td>Sample data bytes 232-261</td>
<td>313</td>
</tr>
<tr>
<td>C-6</td>
<td>CUSTIREQ copybook</td>
<td>314</td>
</tr>
<tr>
<td>C-7</td>
<td>CUSTADRR copybook</td>
<td>314</td>
</tr>
<tr>
<td>C-8</td>
<td>CUSTINFO copybook</td>
<td>315</td>
</tr>
<tr>
<td>C-9</td>
<td>READCINF</td>
<td>316</td>
</tr>
<tr>
<td>C-10</td>
<td>Sample CICS/MQ/COBOL compile proc</td>
<td>325</td>
</tr>
<tr>
<td>C-11</td>
<td>Compile JCL sample</td>
<td>327</td>
</tr>
<tr>
<td>C-12</td>
<td>IIREDBOK installation</td>
<td>333</td>
</tr>
<tr>
<td>E-1</td>
<td>Content of the scfgd_mutualfund.xml file</td>
<td>344</td>
</tr>
<tr>
<td>E-2</td>
<td>Content of the ibm.xml file</td>
<td>345</td>
</tr>
<tr>
<td>E-3</td>
<td>File stockInfo.dadx in its entirety</td>
<td>345</td>
</tr>
</tbody>
</table>
Getting Started on Integrating Your Information
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 illustrates 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. 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 IBM's application programming interfaces.
Trademarks

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

IBM eServer™
Redbooks (logo)™

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

AIX®
CICS/ESA®
CICS®
DataJoiner®
DB2 OLAP Server™
DB2 Universal Database™
DB2®
DRDA®
IBM®
IMS™

Informix®
Intelligent Miner™
iSeries™
Lotus®
MORE™
MQSeries®
OS/390®
Perform™
QMF™
Redbooks™

S/390®
SP™
SupportPac™
TME®
WebSphere®
Word Pro®
z/OS™
zSeries™

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

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

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

The explosion of the Internet and e-business in recent years has caused a secondary explosion in the amounts and types of information available to enterprise applications. The challenge facing organizations today is information integration. Enterprise applications must interact with databases, application servers, workflow systems, message queues, content management systems, search engines, data warehouses, mining, analytics, Web crawlers. They must use a variety of programming interfaces (ODBC, JDBC, Web Services, J2EE). They must extract and combine data in multiple formats (SQL, XML, SOAP, images) generated by multiple delivery mechanisms.

Integration solutions incorporate one or more elements of process, application, user interface, and information integration. Coping with the information explosion requires information integration, a class of middleware which lets applications access data as though it were in a single database, whether or not it is.

This IBM Redbook shows, through some basic examples, the various ways in which information integration can be implemented as a “pure play” or in conjunction with the other approaches for integration.

IBM DB2 Information Integrator is the core product for the IBM information integration solution. IBM DB2 Universal Database (DB2 UDB throughout the book) is the foundation for the IBM information integration technologies. IBM DB2 Information Integrator (DB2 Information Integrator throughout the book) extends the integration capabilities of DB2 UDB. However, IBM DB2 Information Integrator Version 8.1 was not yet available when our book was developed. Therefore, IBM DB2 Universal Database Versions 7.2 and 8.1 were used in the examples. All of the information in this book is directly applicable to IBM DB2 Information Integrator Version 8.1.

This book describes how to start implementing several information integration scenarios:

- To federate data access from multiple sources as message queues, with different relational databases using SQL
- To access information using a Web Services solution
- To store, compose and manipulate XML data
- To populate a common data store

This book will help architects and implementers to understand the integration technologies in DB2 Universal Database and DB2 Information Integrator and to start implementing information integration solutions.
The team that wrote this redbook

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

Corinne Baragoin is a Business Intelligence Project Leader at the International Technical Support Organization, San Jose Center. She has over 16 years of experience as an IT specialist in DB2 UDB and related solutions. Before joining the ITSO in 2000, she worked as an IT Specialist for IBM France, supporting Business Intelligence technical presales activities.

Joachim Dirker is a Data Management Solution Architect with IBM Germany. He has 15 years of experience in the Data Management field. His areas of expertise include data modeling, data base design, and performance tuning for high-end OLTP and Data Warehouse solutions on DB2 UDB for OS/390 and UNIX/Windows platforms.

Carolyn Elkins is an I/T Specialist in the USA. She has more years than she will admit (20+) of experience in information systems and application programming fields. Carolyn has worked at IBM for four years. Her areas of expertise include WebSphere MQ, WebSphere MQ Integrator and CICS.

Ian Harvey is an Integration Architect at HMSG Converged Services, Inc. in Toronto, Canada. He has 16 years of experience in business systems architecture design, integration, development, and implementation, as well as extensive experience with a variety of vendor integration technology implementations, including the IBM WebSphere MQ family of products. He holds a Business Administration diploma in Computer Programming/Analysis and Accounting and Financial Management from Centennial College of Applied Arts and Technology. He is also an IBM Certified WebSphere MQ specialist.

Fred Lo is an Application Architect for the IBM Pacific Development Center in Vancouver, British Columbia, Canada. He has over 13 years of experience in system and application design and development. He has architected, designed, and built software in areas such as database engine, database applications, and e-business solutions. He claims he knows WebSphere Application Server, database, and Web technologies. At the same time, he is also the first to admit that he has to learn more on the same topics. He has taught e-business architectures and designs to many students all over the world, including India and South Africa. He holds a BS degree in Computer Science and an MBA emphasizing Information Technology and International Business from Brigham Young University.
We would like to specifically thank the following people for their technical input and contributions to this project:

Yannick Barel
Jacques Labrie
Gil Lee
Susan Malaika
Micks Purnell
IBM Silicon Valley Lab

Thanks to the following people for their contributions and reviews:

Andrea Ames
Seeling Cheung
Eileen Lin
Mary Roth
Louis Thomason
IBM Silicon Valley Lab

Barry Devlin
Information Integration in Financial Services Solution Centre

Grant Hutchison
Patrick Zeng
IBM Toronto Lab

Yvonne Lyon, for technical editing
IBM International Technical Support Organization, San Jose

**Become a published author**

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll 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](http://ibm.com/redbooks/residencies.html)
Comments welcome

Your comments are important to us!

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

► Use the online Contact us review redbook form found at:
  ibm.com/redbooks

► Send your comments in an Internet note to:
  redbook@us.ibm.com

► Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. QXXE  Building 80-E2
  650 Harry Road
  San Jose, California 95120-6099
Introduction

The rapidly changing economic climate is driving the need for improved access to information, flexible analytical capabilities, and monitoring of business events for altered and emerging trends. An enterprise’s capacity to express its information value and externalize it through the Web will improve its attractiveness to customers and partners.

In this chapter we present the business climate and introduce the concept of information integration. We highlight the design objectives, features, and business requirements for implementing information integration in addressing the current business challenges which many enterprises face. Then we determine the position of information integration in relation to other integration approaches in a total business integration model.
1.1 The business climate

In many ways, the fundamental principles and sound practices of the old business model still remain. Yet it is no longer adequate to simply survive, given the prevailing business climate. Although the current business environment is heavily reliant on an Internet e-business model, it is no longer being driven by a technology domain. Business units are now shaping and driving this model, however, as technology has been relegated to an enabling role. Unfortunately, technology is merely a tool that is leveraged in the process of conducting business.

The current business climate can be described as open, demanding, cost conscious, event based, and technology enabled. It is driven by the need to have personalized interactions with customers, thereby maintaining loyalty and exploiting cross selling opportunities. In addition, a priority of businesses today is the need for improved relationships and tight integration with business partners. Invariably, this close relationship leads to efficiencies and opportunities that are of benefit to both parties. Underlying this need for better interaction with e-business constituents is the reliable and timely integration of information. Indeed, strategic information integration is a critical factor for successful participation, by any business, in the new economy.

1.2 Business challenges and information integration

Given today’s competitive and cost-conscious business climate, successful business concerns require a technology solution that facilitates improved operational efficiency, strategic interaction with customers and business partners, quick response to emerging opportunities, the creation of a competitive advantage, and improved customer loyalty and satisfaction.

Information integration technology is a viable solution that can be used to address a broad range of integration needs. Examples of business problems from a number of customers are listed in Table 1-1.
Table 1-1  Business requirements and information integration technical challenges

<table>
<thead>
<tr>
<th>Business requirement</th>
<th>Information integration technical challenges</th>
</tr>
</thead>
</table>
| 1. The need to integrate islands of information to improve business effectiveness. These applications are legacy (home grown), packaged, or acquired through mergers, acquisitions, or restructuring | ▶ Leverage XML to integrate and exchange information across the enterprise  
▶ Build business objects in XML for use by new and existing applications  
▶ Provide an information service layer to support portals  
▶ Integrate structured and semi-structured data  
▶ Implement an information access infrastructure as a foundation for application and business processing integration  
▶ Store XML in a relational database  
▶ Implement text data and mining capabilities |
| 2. Redesign Web site to facilitate more efficient creation and delivery of content. Includes dynamic and static content to all business channels (as wireless devices and desktop computers). This will attract advertisers and a broader customer base by the enhanced usability of the Web portal with relevant and up-to-date content. | ▶ Integrate structured (as RDBMS), unstructured (as images) and semi-structured (as XML) data from existing data stores  
▶ Exploit XML technology to deliver content |
| 3. Become the partner of choice for order fulfillment resulting from B2B and B2C transactions. Speed product delivery by delivering order processing with internal manufacturing and shipping applications and external supplier and trading partner applications. This will streamline information flow between people, processes and applications to eliminate unnecessary work and delay. | ▶ Extract shipping data from customer orders  
▶ Ensure prompt return of delivery status information to customers  
▶ Use XML  
▶ Federation of structured and unstructured data |
| 4. Integrate tightly business processes and underlying data and applications to facilitate faster response times, improved business effectiveness and new business models. This allows a business to adhere to current industry and governmental regulations. | ▶ Use XML and the Web Services for process integration and message notification  
▶ Leverage J2EE application development  
▶ Integrate (federate) structured and semi-structured data  
▶ Provide an infrastructure for enterprise application integration (including message queuing, publish/subscribe, workflow) |
### 1.3 Overview of information integration

The term information integration, at face value, appears to be a simple concept. How difficult can it really be to integrate information? Though this question is meant to be rhetorical, one’s inability to resist answering this question with a resounding “What?” is understandable. In practice, information integration is much more than a simple concept and needs to be positioned within the broader context of traditional business integration.

With the current business and economic climate, the existing IT environment is a complex, non-integrated mix of old and new systems and enterprises, which need to:

- Improve access to information, flexible analytical capabilities, and formal information inventories
- Integrate analytical data stores (or heterogeneous data access), else enterprises will have difficulty altering strategies without incurring unacceptably long delays in implementation
- Analyze indeterminate business events, which requires analytic horsepower and functionality beyond most so-called “business intelligence” tools
- Monitor business events for altered/emerging trends, as well as manage agility to exploit them will improve the enterprises’ attractiveness to partners and suitors

In summary, enterprises must improve their ability to assemble, analyze, account for, and make available their information assets. They must also build a model-based, integrated infrastructure that can seamlessly interconnect systems and support long-term needs.
**1.3.1 What is information integration?**

**Definition:** Information integration is a category of middleware which lets applications access data as though it were in a single database, whether or not it is. It enables the integration of data and content sources to provide real time read and write access, to transform data for business analysis and data interchange, and to manage information for performance, currency, and availability.

Information integration is not a new concept for IBM; rather, it leverages IBM leadership in data management and builds on the solid foundation of existing data management solutions. IBM has over 25 years of engineering invested in DB2 UDB, the foundation for the IBM information integration products.

The DB2 Information Integrator product set consists of two products: DB2 Information Integrator and DB2 Information Integrator for Content. DB2 UDB is the foundation for DB2 Information Integrator. DB2 UDB integrates information from DB2 UDB and Informix IDS databases. DB2 Information Integrator extends the integration functions of DB2 UDB to support additional relational sources such as Oracle and Microsoft SQL Server as well as non-relational sources. DB2 Information Integrator provides an SQL API for integrating information.

DB2 Information Integrator for Content delivers an information integration framework that integrates diverse, distributed data primarily for IBM Content Manager customers. The interface for DB2 Information Integrator for Content uses the IBM Content Manager interfaces and object-oriented APIs.

Bringing focus to information integration will accelerate development and innovation to deliver solutions that maximize the value of information assets of an enterprise.

There are five aspects of information integration which we will now describe.

- **Federating and optimizing data access:**
  
  Federation is the ability to transparently access diverse business information, from a variety of sources and platforms, as though it were a single resource. A federated server may access data directly (by accessing a relational database) or access an application that creates and returns data dynamically (as a Web Service).

- **Incorporating heterogeneous data via XML:**
  
  XML has gained wide acceptance to be the standard for data interchange. The number of applications that are going to utilize XML as the data interchange format are growing in a rapid pace.
Although XML solves many problems by providing a standard format for data interchange, some challenges remain. When building an enterprise data application, you must answer questions such as:

– What kind of information must be shared between applications?
– How can I quickly search for the information I need?
– How can I have a particular action, such as new entry being added, trigger an automatic data interchange between all my applications?

These kinds of issues can be addressed by a database management system. By incorporating the XML information and meta-information directly in the database, XML data becomes more accessible to be integrated and federated as a database object.

Sharing information across extended enterprise and providing a robust infrastructure for business and application integration through Web services:

A Web Service is a set of one or more business functions that can be invoked programmatically over the Internet by applications or by other Web Services. As such, Web Services facilitate distributed computing and are designed to promote interoperability. Several underlying technologies play a key role in supporting Web Services, including HTTP, Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Definition Language (WSDL), and Universal Description, Discovery and Integration (UDDI).

Web Services providers publish their Web Services so that clients can access them using SOAP over HTTP or other protocols.

Web Services process requests from Web clients, invoking the appropriate business functions and typically returning a response. The Web service itself is described by a WSDL document stored in a repository (such as a UDDI registry) or on the Web Services provider’s server. Storing the Web Service description in an appropriate repository offers the potential for interested parties to discover its existence, potentially generating new business for the Web Services provider.

Replicating data and managing copies of data:

Replication is the process used to manage and propagate data changes between multiple copies of data.

Replication involves the automated and reliable movement of business information from one environment to another for on demand access at the target locations. This may involve replicating subsets of data from one system to many (as synchronizing pricing information in retail stores with corporate headquarters), or replicating selected data from many sources to several distributed organizational units (as consolidating information for reference or analysis).
Data mining, content aggregation, federated search and analytics:
This aspect of information integration provides a comprehensive enterprise content management infrastructure. Large collections of digital content as text, graphics, images, video, Web contents, and more, can be managed and integrated with leading applications such as customer service, ERP, assets management.

Information integration provides intelligent mining so you can gain new business insights and to harvest valuable business intelligence from your enterprise data as well as contents.

This book is providing scenarios to get starting on integrating your information and covering the following aspects of information integration:

- Federating data access
- Sharing information using Web Services
- Incorporating data using XML
- Populating a common data store by enabling federation

1.4 WebSphere business and information integration

This section discusses the different styles of integration specified in IBM WebSphere business integration solutions and examines how information integration, one of the five integration approaches in WebSphere, complements and works with the other WebSphere integration approaches.

1.4.1 WebSphere business integration overview

IBM uses the term business integration to describe five ways an enterprise can use integration software to address the company’s business needs.

- User interaction:
  - Delivers information drawn from multiple sources in a personalized fashion through multiple channels
  - Improves access and speed to personalized content, through flexible access channels

- Process integration:
  - Coordinates and controls activities that span multiple systems and people
  - Automates and improves business processes

- Application connectivity:
  - Creates enterprise-wide access to information, making it available where and when the information is needed
– Ensures timely delivery of updated information that every part of the business can work on

> **Build to integrate:**
> – Builds composite applications
> – Harnesses existing resources to address new business opportunities and reuse existing assets

> **Information integration:**
> – Enables the integration of diverse forms of business information across and beyond the enterprise
> – Enables coherent search, access, replication, transformation and analysis over a unified view of information assets to meet business needs

These five ways of using integration software are shown in Figure 1-1.
1.4.2 Traditional business integration

At the core, business integration is all about internal and/or external integration of applications, processes, and information, in support of business events, as shown in the enterprise integration framework in Figure 1-2.

Regardless of the terminology — for example, business-to-business (B2B), business-to-consumer (B2C), or mobile commerce — which is used to describe an application integration effort, business integration incorporates one or more of the various integration types, such as database integration, application integration, process integration, and portal integration.
In addition, traditional business integration is further categorized by product implementations of middleware technologies, as listed in Figure 1-3, which facilitate the sharing of data and business processes.

<table>
<thead>
<tr>
<th></th>
<th>Middleware product implementation categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Remote Procedure Call (RPC) Middleware</td>
</tr>
<tr>
<td>2.</td>
<td>Message Oriented Middleware (MOM)</td>
</tr>
<tr>
<td>3.</td>
<td>Distributed Objects Middleware</td>
</tr>
<tr>
<td>4.</td>
<td>Database Oriented Middleware</td>
</tr>
<tr>
<td>5.</td>
<td>Transaction Oriented Middleware</td>
</tr>
<tr>
<td>6.</td>
<td>Message Broker Middleware</td>
</tr>
<tr>
<td>7.</td>
<td>Web Services Middleware</td>
</tr>
</tbody>
</table>

1. **Remote Procedure Call (RPC) Middleware**: This provides the ability to invoke a function within one application and have that function execute within another application on a remote machine. For example, an RFC to an SAP BAPI component or a database using SQL.

2. **Message Oriented Middleware (MOM)**: This is traditional queuing software that use messages as a mechanism to move information from point to point. IBM’s MQSeries product is an example of a MOM.

3. **Distributed Objects Middleware**: These are small application programs that use standard interfaces and protocols to communicate with one another — for example, Java application programs using the RMI or CORBA IIOP protocol to communicate.

4. **Database Oriented Middleware**: This is any middleware that facilitates communication with a database, either from an application or between databases — for example, an ODBC/JDBC call within an application to access services provided by a DB2 UDB database.

5. **Transaction Oriented Middleware**: This includes Transaction Processing (TP) monitors and application servers. TP monitors provide a location for application logic and a mechanism to facilitate the communication between two or more applications. Two examples are BEA Tuxedo and IBM CICS. TP monitors are based on the premise of a transaction, which is a unit of work with a beginning and an end. This unit of work can be a bank account transfer transaction that must be reversed if not completed successfully as a whole. Application servers also implement the unit-of-work (UOW) concept and enable sharing and processing of application logic that may facilitate connection to back-end resources.
6. **Message Broker Middleware:** These are servers that facilitate information movement (such as brokering messages) between two or more resources (such as source or target applications) and can account for the differences in application semantics and platforms. In addition, message brokers can integrate many applications using common rules and routing engines. Message brokers also apply data mapping/transformation to messages flowing between applications.

An example of a message broker is the IBM WebSphere MQ Integrator product. Message brokers generally include adapters that act as a gateway between the message broker and an application's native environment. In particular, adapters are implemented as a layer between the message broker interface and the source or target application, via a set of libraries that map the differences between two distinct interfaces, that hides interface complexities. These are three possible message broker topologies:

a. **Hub-and-spoke:** The message broker rests between source and target applications being integrated in a configuration that resembles a star.

b. **Bus:** The message broker sits on the network bus and provides message broker services to other systems on the bus.

c. **Multihub:** A number of message brokers are linked or federated, with the source and target applications linked to any local/remote brokers available in the configuration. The multihub topology provides the ability to scale, making it possible to integrate a virtually unlimited number of source and target applications. As the need grows for more applications in a single broker environment, the excess is addressed by adding more brokers to the network.

7. **Web Services Middleware:** this type of middleware supports the sending of XML messages, that are wrapped in a SOAP envelop (also using XML notation), to a server for processing. Depending on the communications transport (HTTP, MOM, SMTP, FTP) selected Web Services can be implemented using different middleware technologies and communication styles. Web Services imposes additional overhead by including an abstraction layer above the traditional object/method approach to business integration. The benefit of this abstraction layer is that business logic code is not reliant or limited to a specific language implementation (as COBOL or Java)

Furthermore, middleware may be classified by using the following models:

1. **Point-to-point:** In this model, a direct link is allowed from one application to one other application. This link is limited to the binding of only two applications. The point-to-point configuration also lacks a facility for middle-tier processing, such as the ability to house application logic or the ability to change messages as they flow between the link.
2. **Many-to-many:** In this model, many applications are linked to many other applications. This approach is best suited for internal Enterprise Application Integration (EAI) and is employed by message brokers.

Finally, middleware typically employs one or more communication styles, as listed in Figure 1-4.

<table>
<thead>
<tr>
<th>Communication Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Synchronous</td>
</tr>
<tr>
<td>2. Asynchronous</td>
</tr>
<tr>
<td>3. Connection-oriented</td>
</tr>
<tr>
<td>4. Connectionless</td>
</tr>
<tr>
<td>5. Direct communications</td>
</tr>
<tr>
<td>6. Queued communications</td>
</tr>
<tr>
<td>7. Publish/subscribe</td>
</tr>
<tr>
<td>8. Request response (reply)</td>
</tr>
<tr>
<td>9. Fire and forget</td>
</tr>
<tr>
<td>10. Conversational mode</td>
</tr>
</tbody>
</table>

*Figure 1-4 Integration communication styles*

1. **Synchronous:** In this method, the middleware software is tightly coupled to applications. That is, the application is dependent on the middleware software to process one or more function calls at a remote application. As a result, the calling application must halt processing in order to wait (blocking) for the remote application to respond.

2. **Asynchronous:** This method enables the movement of information between one or many applications in an asynchronous mode. That is, the middleware software is able to decouple itself from the source or target applications. And, an application’s completion is not dependent (time-wise) on the other connected applications.

3. **Connection-oriented:** This enables two applications to connect, exchange messages and then disconnect. This is typically a synchronous process, however, it can also be asynchronous.

4. **Connectionless:** The calling program does not enter into a connection with the target process. And, the receiving application simply acts on the request, responding when required.

5. **Direct communications:** The middleware layer accepts a message from the calling program and passes it directly to the remote program. Direct communications is usually synchronous in nature and mostly used by RPC-enabled middleware.
6. **Queued communications**: This generally requires a queue manager to place a message in a queue. A remote application then retrieves the message, regardless of time considerations. If the calling application requires a response, the information flows back through the queuing mechanism. MOM products are mostly used for queued communications.

7. **Publish/subscribe**: This approach frees an application from the need to understand anything about the target application. The application simply sends (publishes) the information it desires to a destination contained within the publish/subscribe broker. And, the broker redistributes the information to any interested (subscribed) applications.

8. **Request response (reply)**: A request is made to an application using request response middleware, and it responds to the request.

9. **Fire and forget**: This allows the middleware user to send a message and forget about it, without worrying about who receives it, or even if the message is ever received. This is an asynchronous approach that allows a source or target application to broadcast specific types of messages to multiple recipients.

10. **Conversational mode**: This supports the bi-directional exchange of information between one or more applications in a way that resembles carrying on a conversation.

Though this depiction of business integration does not include all possible groupings, it does highlight the overlap of some categories. For example, a stored procedure database object can also use RPC and MOM technologies.

### 1.4.3 How information integration fits with business integration

In terms of the traditional business integration model, information integration can be classified as a database-oriented, RPC-oriented, transaction-oriented, message-oriented, and Web Services middleware implementation (refer to Figure 1-4 on page 12 for further details). The IBM information integration implementation is used to create business solutions that may include all of the integration types and all of the communication styles. The focus of this book, however, is on business solutions that will use any of these communication styles with a subset of the integration types, as data, application, and business process integration types are used for information integration.

In the heart of WebSphere, business integration is a common programming model for a Service Oriented Architecture (SOA) based on standards:

- J2EE for implementation
- Web Services for access and integration
- Effective interoperability with other standards based integration platforms
WebSphere business interaction employs J2EE for the implementation and runtime environments, with Web Services being the key technology for access, integration, and inter-connectivity. We will further review SOA and Web Services in Chapter 4, “Business case II: Accessing your information through Web Services” on page 119.

After reviewing the five approaches for integrations, we may want to differentiate the roles of these approaches. The need is particularly strong between the information integration and application connectivity. Customers and analysts have found that a quick way to understanding the relative roles is that information integration focuses on integrating data, while application connectivity focuses on integrating applications programs. Historically, applications were designed with the data schema tightly integrated with the application code; hence blurring the boundary between data and applications.

New technologies, specialized organizations for managing the two types of assets, and integration enablement tools have focused on separating these two closely related assets to maximize their respective and unique values to the business. Therefore, it makes sense to integrate at the data level in some cases, while at the application level in others.

**Note:** Application connectivity is used if the problem is to communicate business events or single events. Information integration is used when the problem is to access the state of the business as represented by the information recorded in its data stores.

Furthermore, most modern integration needs are a complex mix of both application and information integration. Solution providers have multiple ways to integrate, share, and distribute information.

The concepts of state and event provide an architecture level guideline on when to use application connectivity and when to use information integration. In addition, the capabilities of each approach have strengths which are best suited for different integration requirements.

The issue is to match the benefits of the different approaches to the different business integration requirements.

When we employ information integration, the approach has these benefits:

- You can integrate data and content without moving the data or changing the platform.
- You can access diverse and distributed data as though it were in a single database, whether or not it is.
You can make more progress, more quickly, and at lower cost. By using the built-in services, you can reduce coding requirements tenfold or more.

You can exploit features available inside the data server(s) to improve performance during any data retrieval operation with the flexibility to either centralize information where availability or performance needs require it, or manage distributed access to data that must remain in place.

Figure 1-5 illustrates the most common way to accomplish application to data integration without information integration at the data level. Of course, there can be many more interfaces than we have shown in the diagram. Developing applications for the different data sources requires knowledge in many different interfaces. Some of the interfaces are so complex that they make application development a monstrous task. Figure 1-6 shows how the application to data integration interfaces are reduced to the standard database access and through Web Services when information is integrated using information integration.

In comparing the two diagrams, applications are tightly coupled with their respective data sources in the first diagram in Figure 1-5. Applications are developed to the corresponding interfaces dictated by their data sources.
In Figure 1-6, we allow the database layer or information integration layer to handle all the tedious work of integrating information. One of the many benefits is that we standardize the way applications gain access to data that is stored in all the different data sources. This greatly reduces the complexity of application development.
Information integration infrastructure

In this chapter we introduce some key technologies enabled in DB2 UDB to realize information integration at the data integration level. Since information integration can involve many technologies, models, and processes, we attempt to limit the scope of what we are to introduce in this book.

At the end of this chapter, we introduce you to the business cases and scenarios used to demonstrate the functions in DB2 UDB for integration information.
2.1 Information integration design objectives and features

The integration job is never finished. IT environments are in a constant state of flux. New applications are coming online. Release-level changes to packaged applications can cause a ripple effect throughout the infrastructure. There is always the next new tool or technology to try. Thus, organizations are emerging within businesses to focus on an integration architecture. Whether they labeled it information management, integration services, or data architecture, companies are tackling the issue of integrating the business, and defining an integration architecture and infrastructure that will provide the foundation for their future applications.

IBM envisions an information integration infrastructure that will give the application layer a unified view of the data it must access, regardless of differences in data format, data location, and access interfaces.

The information integration framework’s objective (see Figure 2-1) is to promote simple integration at the data and information layer to be used in conjunction with the application and business process layers of the traditional business integration stack — thereby avoiding the complexities typically associated with having to learn and implement various product APIs in the process of business integration.
The IBM information integration infrastructure will enable integration of diverse, distributed, and real-time data as if it were a single source, no matter where it all resides. The key features of the infrastructure include the ability to federate, search, cache, transform, and replicate disparate data:

- **Federation:**
  
  Federation is the concept involving a collection of data sources that can be viewed and manipulated as if they were a single source, while retaining their autonomy and integrity. The resources may be uniform or diverse, or collocated or distributed, depending on the implementation.
Federation provides the facility to create a single-site image of disparate data by combining information from multiple vendor databases located elsewhere using database client wrapper libraries. The vendor databases may be located locally or remotely and access is optimized using a middleware query processor. A federated server is an abstraction layer that hides complexity and idiosyncrasies associated with the implementation of various vendor databases. The federated RDBMS works behind the scene to make access to this disparate data transparently and efficiently. Such work includes automatic data transformations, API conversions, functional compensation, and optimization of data access operations. Federation also allows the presentation of customer data through an unified user interface.

DB2 Information Integrator, DB2 Information Integrator for Content and DB2 UDB are the product offerings that provide heterogeneous federation capabilities. DB2 UDB is the foundation for DB2 Information Integrator and federates data from DB2 UDB on any platform and data from Informix. DB2 Information Integrator federates data from DB2 UDB, Informix, and non-DB2 sources such as Oracle, as well as non-relational sources such as XML. While DB2 Information Integrator and DB2 UDB has an SQL interface, DB2 Information Integrator for Content uses IBM Content Manager interfaces and object APIs as its federation interface.

Being able to get a unified data view of the data through the enterprise can be accomplished using SQL and XML client access to federated databases. Information integration enables federated access to unstructured data (such as e-mail, scanned images, and audio/video files) that is stored in repositories, file systems, and applications. This facilitates efficient storage, versioning, check-in/check-out, and searching of content data. An additional benefit of information integration content management is the reduction in operational costs.

- **Replication:**

  Replication is required as a fundamental characteristic of an information integration infrastructure. It complements the distributed access features, enabling management of centralized data stores, and provides the necessary infrastructure for efficiently managing data caches. Data caches (Materialized Query Tables) support placing and managing data at multiple points in the data hierarchy to improve performance and may be defined over the federated data sources.

  This functionality focuses on efficiently capturing database changes, altering the data to match the target schema, applying the change to the target system, and the end-to-end management of the transfer. This capability allows for either incremental or full replacement in a variety of combinations and with a range of data currency options. It assures that only committed information is captured and replicated. This replication is designed to minimize performance impact to production applications.
Data transformation:
The infrastructure must provide rich transformation features to facilitate analysis, interchange, or presentation. Standard SQL includes the ability to apply statistical functions and perform online analytical processing functions. Type-specific features further enhance these transformations, such as applying scoring algorithms, spatial analysis, or chemical similarity searches.

Extensible stylesheet language (XSL) translations facilitate document interchange and dynamic style matching to diverse display characteristics. User-defined functions enable customers to standardize any function for any data type. In addition, the ability to access Web Services as a built-in function means that any Web service can become an embedded transformation function. By leveraging built-in transformation features, data transformation occurs as close to the data store as possible, thereby helping to minimize data movement and speed results.

- Web Services capabilities implemented in information integration include:
  - The ability to expose stored procedure functionality as a Web Service. Using the Web Services Object Runtime Framework (WORF) and Document Access Definition Extension (DADX), these Web Services can be served to clients by an application server (for example, WebSphere Application Server).
  - The ability for stored procedures to become SOAP clients and request Web Services from SOAP servers.

- Advanced search and information mining answer customers need of sophisticated analysis of the textual information in their applications, content repositories, e-mail repositories, databases and file systems. Text analytics gathers and summarizes information about individual documents as well as groups of documents:
  - Language identification describes the language of each document, important for international businesses.
  - Feature extraction identifies information about the document, such as recognizing vocabulary items, names referring to a single entity, locating proper nouns, and abbreviations.
  - Categorization assigns documents into pre-existing categories based on a taxonomy defined ahead of time by the firm (product lines, competitors, and so on).
  - Clustering of information into groups of related documents automatically based on content. This is different from categorization, as it does not require pre-defined classes.
  - Summarization extracts sentences from each document to create a document synopsis.
These capabilities are delivered in IBM DB2 Information Integrator for Content.

- **Part of a complete integration solution:**

  The IBM information integration infrastructure provides a comprehensive range of functions:

  - **XML publishing, consumption, interchange:**

    IBM provides the ability to store and retrieve structured relational data, but also semi-structured XML documents and unstructured content, such as byte streams and scanned images.

    Publishing and interchange through XML documents is a key technology that enables an enterprise to share data with internal and external applications in a common format. In addition, information integration provides a facility for interchange of XML data and to manipulate XML documents. XML documents can be loaded in DB2 UDB (and use DB2 XML Extender to manipulate them) or be manipulated using new DB2 SQL/XML functions. They can also be accessed in place using the XML wrapper provided by DB2 Information Integrator. In this redbook, we only provide scenarios based on DB2 UDB, using DB2 XML Extender and DB2 XML functions.

    With the content of XML documents in a DB2 UDB database, you can combine XML information with traditional relational data. Based on the application, you can choose whether to store entire XML documents in DB2 UDB as in user-defined types provided for XML data, or you can map the XML content as base data types in relational tables.

    DB2 UDB provides the following functions to manipulate XML documents:

    - The capability to store entire XML documents as column data or externally as a file, while extracting the required XML element or attribute values and storing it in side tables, indexed subtables for high-speed searching. Storing XML documents as column data is known as the XML column method of storage and access. By storing the documents as column data, you can:
      - Perform fast search on XML elements or attributes that have been extracted and stored in side tables as SQL basic data types and indexed.
      - Update the content of an XML element or the value of an XML attribute.
      - Extract XML elements or attributes dynamically using SQL queries.
      - Validate XML documents during insertion and update.
      - Perform structural-text search with the text extender.
• The capability to compose or decompose contents of XML documents with one or more relational tables, using the XML collection method of storage and access. XML documents can be composed from relational storage and also WebSphere MQ and file system data stores.

- **WebSphere business integration:**

  Key to developing a corporate integration infrastructure is the ability to easily leverage appropriate integration technologies, together or separately. The information integration infrastructure provides client APIs that leverage messaging and workflow facilities, such as those provided by the WebSphere software platform from IBM. Thus, a database event — for example, the arrival of a new piece of information — can transparently create a notification, for example, by putting a new message on a queue.

  This allows information integration to facilitate the full participation of internal and external applications in the business processes of a given enterprise. This can be accomplished using the MQSeries Workflow and WebSphere MQ products.

  Message queuing is a data transport mechanism that is often described as “e-mail between applications”. A sending or requesting application formats a message, a structure that can contain any type of data, and writes the message to a data store called a queue. The serving or receiving application then retrieves the message from the queue and processes the data — a simple concept that provides effective decoupling of applications, allowing an application to communicate information to another without being directly connected. The sender and receiver do not have to reside on the same platform; in fact, neither the sender nor receiver has to know where the other application is hosted. Nor do they have to be running at the same time. Messages will remain queued until they are retrieved by an application or expire.

  WebSphere MQ supports two messaging styles used by information integration: datagrams, also known as “fire and forget” and request/reply messaging. Datagrams, the ability to send or publish information without expecting a response from the receiving or subscribing application, is often used in data replication. Request/reply messages provide communication between requesting applications and the servers that fulfill the request. This style is often used in providing business services to both internal and external clients.

  While WebSphere MQ is basically an asynchronous communication method, many request/reply applications actually run in what we call “near real time”. That is, the initiating application expects a reply from the service request. Please note that this does not imply that the application waits on a reply, although it frequently does.
In addition, WebSphere MQ runs on 35+ platforms and uses a common API on all these platforms. More importantly, WebSphere MQ guarantees persistent message delivery.

For example, DB2 Information Integrator enhances the WebSphere MQ integration in DB2 UDB with access to federated heterogeneous data.

With MQSeries Integration, DB2 UDB stored procedures and user-defined functions and triggers can send and receive text or XML messages to/from WebSphere MQ queues. In addition, WebSphere MQ queues can be used as a data source for SQL calls. Message queuing facilitates data interchange with traditional business integration implementations. A DB2 UDB developer or administrator can use SQL statements and does not have to learn the WebSphere MQ Integrator to access WebSphere MQ queue data.

**Note:** DB2 Information Integrator new family of products consists of:

- IBM DB2 Information Integrator V8.1
- IBM DB2 Information Integrator for Content V8.2 (formerly IBM Enterprise Information Portal)

DB2 Information Integrator V8.1 core capabilities include:

- A federated data server:
  
  This provides an integrated and single view across diverse and distributed data. Data sources may be:
  
  – Relational as DB2 UDB, Informix, Microsoft SQL Server, Oracle, Sybase, Teradata, ODBC sources
  
  – Non-relational, as WebSphere MQ message queues, XML documents, Web Services, spreadsheets, flat files, Microsoft Access, and so on.

- A replication server for mixed relational databases:
  
  Data can be replicated between mixed relational data sources. DB2 UDB, Informix, Microsoft, Oracle and Sybase are supported as replication sources and targets. Teradata is supported as a replication target.

Applications can query or search across the aggregated data sources as if they were in a single database using standard SQL, text search semantics within the query. The query may produce standard SQL answer sets or XML documents. Data caches (Materialized Query Tables) may be defined over the federated data. (For a complete description of the sources accessed and capabilities, please refer to the announcement letter).

DB2 Information Integrator teams with WebSphere Business Integration to form a complete business integration solution.
2.2 Information integration layers

There are a number of key functions that must be addressed by an information integration platform:

- Read/write access to all forms of information:
  - Structured, semi-structured, and unstructured
  - Heterogeneous: legacy, packaged and Web
  - Real time versus point-of-time
- Unify query access over local and federated data: SQL/XML and SQL
- Metadata management: Access and manage data and its description with common tool, data that are stored in DB2 UDB
- Choice of storage modes: Relational or XML
- Choice of data delivery: Application interface as CLI, ODBC, JDBC (relational), Web Services (XML)
- Data placement management: Replication, ETL, caching over heterogeneous data

In this section, we examine the features, extensions, and integration functions in DB2 UDB.

DB2 UDB, with these functions, forms the foundation for DB2 Information Integrator (shown in Figure 2-2). Other technologies and tools include message queuing, XML, and Web Services. Next, we discuss the benefits of these tools.

Figure 2-2  IBM information integration platform
2.2.1 Why DB2 UDB?

The database management system is at the core of enterprise computing. Companies need to store and have access to a wide range of information such as XML, documents, streaming video, and other rich media types. They need fast, near real time, and efficient access to this wide range of information. At the same time, they need to provide access to this information to their business partners and external organizations.

The e-business evolution makes continuous availability a necessity and is driving the convergence of transaction, business intelligence, and content management applications as companies integrate their business operations. The evolution of e-business requires the support of database technology to achieve both the management and integration at the data and application levels.

Due to organizational or operational constraints, information from structured, semi-structured, and unstructured data sources does not generally lend itself to being fully replicated or consolidated under a single database view; hence the increased demand for federated access to distributed sources.

The ever changing business climate and requirements are shaping a new breed of database management system. The scope of data management is expanding significantly beyond the traditional OLTP, Business Intelligence, and content market, instead, it has to create value for the users by integrating information (see Figure 2-3).

- Data, applications, and business processes have to be integrated more tightly to provide:
  - Faster response times
  - Improved business effectiveness
  - New business models

- Data is not purely relational, but rather, it has:
  - XML and Web content
  - Integration with both structured and unstructured data
  - Support for analytics (such as sophisticated search or mining)
  - An infrastructure for enterprise application integration (such as messaging or workflow)
In order to integrate information, we need the best middleware platform that partners and customers can exploit for integrating information:

- Best of breed integration of unstructured and structured data
- Advanced integration services, search, analytics
- Robust infrastructure to complement application and process integration
- Best data store for XML data
- Access real time data on the Internet using Web Services

A full family of leading technologies and tools for integrating information is made available through DB2 UDB:

- DB2 UDB provides the foundation for storing, managing, and federating all forms of content. DB2 UDB includes a rich set of statistical and analytical functions and offers the best optimization technology in the industry, including distributed optimization and automated caching functions. DB2 UDB provides traditional client access and access from Web Service clients.

- DB2 UDB XML capabilities, Net.Search provides data type-specific extensions to query, access, update, and manage various data objects. For example, with DB2 UDB Extenders, you can traverse XML documents, query by image shape or color, or query based on proximity to a given location. With all the DB2 UDB Extenders, particular with XML Extender, DB2 UDB is well suited for integrated B2B applications.
DB2 Information Integrator is a new product that enables applications that use SQL or tools that generate SQL to access, integrate, and manipulate distributed and diverse data through a federated system server. It provides source-specific wrappers to access and optimize access to heterogeneous relational databases, including DB2 UDB, Informix, Oracle, Microsoft SQL Server, and Sybase. Also available is a replication server to replicate data between mixed relational data sources supporting point-in-time and near-real time replication with embedded transformation for populating data warehouses and datamarts. DB2 Information Integrator was announced in February 2003 with general availability to be announced at a later date.

DB2 UDB is the first to demonstrate the power of combining SQL and its extensions, federated data, Web Services and text search to solve common, yet complex, e-business problems:

- DB2 UDB leads the industry in integrating the exploiting Web Services for real time data access: The Web Services provider support gives access to database resources from non-traditional clients, for example, enabling pervasive devices to access XML documents stored in DB2 UDB.
- DB2 UDB provides a more complete XML implementation. See the Bloor report at: http://www-3.ibm.com/software/data/pubs/pdfs/bloor.pdf
  The strategy here is to integrate and manage XML data and to bridge the need to integrate XML data with traditional data while at the same time providing a native XML store.
- DB2 UDB and DB2 Information Integrator enable the realization of data federation a real-world option. Federation refers to the ability to access data as if it were a single data sources yet at the same time retain the autonomy and integrity of the separate data sources. Industry adoption has been limited by perceived performance limitations and lack of standards.
  DB2 UDB has been delivering federation technology for several years in DB2 DataJoiner, DB2 Relation Connect, DB2 Life Sciences Connect, IBM DiscoveryLink, and IBM Enterprise Information Portal and has the ability to federate relational database, flat files, XML documents, spreadsheets, content repositories, and more. Based on patented optimization technology, IBM can demonstrate the performance required to make federation a viable option. It is the only technology to date that viably opens the door to delivering coherent access to petabytes of information on intranets and extranets.
- DB2 UDB led the way with data federation. Now DB2 Information Integrator extends the transformation and accelerating power of the relational database engine to federated data. Built-in transformations such as statistical analysis, online analytical processing, XML transformations, spatial transformation, or user-defined functions can be applied by the engine to federated data to facilitate analysis, interchange, or presentation.
Query rewrite capability automatically rewrites queries to provide the best performance from the distributed and remote federated sources of information. Additionally, the database optimizer can improve access performance by automatically detecting the executing actions in parallel over federated sources (as Web Services) or by supporting locally cached copies of federated data via automatic summary tables, all transparent to applications.

In building and developing the new e-business applications, enterprises will benefit tremendously from the set of functions and capabilities provided by DB2 UDB and DB2 Information Integrator.

### 2.2.2 Why message queueing?

WebSphere MQ provides some unique services useful to both information and application integration. These services include platform independence, timing independence, and guaranteed delivery of messages.

From an information integration standpoint, though there is a growing emphasis on tight information integration, that does not imply a common or single data source for an enterprise. There are a number of practical considerations, including performance, failover and disaster recovery, in addition to application considerations that drive the need for multiple, redundant data sources. These data sources may not even reside on the same platform type, even though they contain much of the same data.

Using WebSphere MQ for data source synchronization allows a data store (via triggers or an equivalent function) to publish data changes to any number of endpoints, and those requests may be processed up immediately (near real time) or be held on a queue until the receiving application is ready to process them (asynchronous).

In addition, because WebSphere MQ guarantees the message delivery, the synchronization process does not normally have to be concerned with information loss, simplifying the code. That combined with WebSphere MQ availability on a wide variety of platforms make it a valuable tool in data replication.

Another reason WebSphere MQ has become the middleware market leader is that many commercially available applications, financial management and human resource systems for example, have added messaging interfaces. This interface usually provides a way for external systems to request services from the application without directly accessing proprietary data. In addition, it allows the application to request services from other applications.
Allowing WebSphere MQ to carry the business service requests and responses is particularly useful where the requestor and provider reside on unknown, or a potentially changing platforms. This is often the case when an enterprise has a continuous processing requirement (24x7). An enterprise will often implement redundant instances of a service application residing on multiple servers to prevent loss of service during both planned and unplanned outages. Platform independence and high availability is also extremely important to enterprises that provide services to external customers, where there is little control over the platform or application type issuing the requests.

A request/reply scenario usually works as follows:

- The requesting application formats the service request message and writes it to a queue.
- WebSphere MQ routes the message to an available queue based on the way the target queue is defined. This queue could be local (on the same server), it could be a specific remote queue, or it could vary if the target queue and application are defined a number of times within the network.
- The target application:
  - Retrieves the message
  - Fills the request
  - Formats and sends the reply.
- The requesting application retrieves the reply and continues processing.

In addition, while the message queueing model is asynchronous, the reliability and speed that WebSphere MQ has demonstrated has led to its use in real time, or near real time, applications.

WebSphere MQ and the WebSphere MQ family provide not only the transport and communication layer to implement data and application integration, they also provide tools that may be used to format data for different data sources and applications, route information to a specified endpoint, publish information to any number of subscribers, and allow an application process to subscribe to published information. These tools include two that are extremely important to information integration:

- WebSphere MQ publish/subscribe for distributed platform: this tool provides the ability to publish messages to multiple subscribers based on a defined topic, removing the normal one-to-one tie between the sender and consumer of WebSphere MQ messages. It is delivered in MQSeries SupportPac MA0C available from the MQSeries SupportPac Web site at:
  
WebSphere MQ Integrator (WMQI), which provides a number of information and application integration services, including:

- The ability to transform, enrich, route, and apply business logic to messages based on the message content.
- A more sophisticated publish/subscribe process, that allows consumers to subscribe based on the message topic and on message content.
- A graphical development environment.
- A message definition palette that allows users to import the structure of the message from COBOL copybooks, C headers, XML DTDs, and XML schemas.

For more information about WebSphere MQ and the complete family, please see the following sources:

- An Introduction to Messaging and Queueing, GC33-0805
- WebSphere MQ Integrator Broker Introduction and Planning, GC34-6167
- The WebSphere MQ family Web site:  
- MQSeries Publish/Subscribe SupportPac MA0C:  
- Getting started with MQSeries Publish/Subscribe SupportPac MA0D:  

MQSeries Integration
The widespread use of WebSphere MQ as a communication and data transportation tool has been integrated into DB2 UDB. The DB2 Information Integrator extends this capability with SQL-based access to a variety of non-DB2 data sources.

An SQL application or DBA can now easily access WebSphere MQ resources, providing a very powerful tool for alternate platform communication and ongoing data source synchronization.

The MQSeries Integration is a set of DB2 User Defined Functions (UDFs) that provide seamless access to WebSphere MQ messaging via standard SQL statements. These UDFs can be used to send and receive text or XML messages to/from WebSphere MQ queues.
MQSeries AMI

The MQSeries Application Messaging Interface (AMI) provides an MQ interface wrapper designed to allow access to WebSphere MQ resources, without needing to understand and code all available functions and options available via the MQI. Through the AMI concept of Service Points and Policies applications that want access a queue just need to know by which services and policies this certain resources is controlled. The application then uses AMI function, passing the defined service point, policy, message area, and a limited number of other optional parameters as defined for that function.

An AMI service point is a logical representation of a WebSphere MQ destination, a physical queue manager, and queues. The queue manager must be local to the instance of the DB2 UDB server. The queues may reside locally, in which case they can be used for both message send and retrieval functions; or they may be remote, in which case they may only be used for send functions.

An AMI policy sets the operational controls for the functions: It specifies how a WebSphere MQ message is to be created or retrieved.

The MQSeries Integration uses the AMI interface to access WebSphere MQ resources, relying on the MQseries AMI administrator to define the appropriate service points and policies necessary. The MQSeries Integration implementation architecture for the business cases explored in this document is shown in Figure 2-4.

![Figure 2-4 MQSeries Integration components](image-url)
Information integration and WebSphere MQ Integrator

The extremely reliable and platform rich support provided by WebSphere MQ is often the first step in integration. Other services are often needed to integrate data and applications, including (but certainly not limited to) data transformation, data enrichment, and message content based routing and subscriptions. WebSphere MQ does not examine the message body when making routing or transformation decisions, only at the message descriptor.

A common problem when integrating data and applications is when the format of the data is not homogenous in the enterprise. Again, there are many reasons for these differences, including a mixture of application types and platforms.

As an example, a simple request to validate a general ledger account may come from several different applications that run on several platforms. While simple ASCII to EBCDIC can be done by most transportation tools, it is more difficult to provide a way to convert an account number that is viewed as all numeric values (9876543210) into one that contains dashes (98-7654-32-10) for verification by the general ledger application. Integration must be added for a second application that zero fills the account number on the right (987654321000000) and another that zero fills on the left (000009876543210). The problem is compounded if the requests are coming from vendor purchased applications, as the source is not always available for tailoring so the request can be remapped before being sent.

Differing data formats often leads to building numerous interface programs. These programs often become a maintenance nightmare, taking on a life of their own, as each change to data, the applications accessing the data, or the application fulfilling the service request requires the examination of all interface programs.

WQMI helps resolve that issue by providing data transformation within an WebSphere MQ message. A key component of transformation, using existing data formats, such as C headers, COBOL copybooks, DTDs and XML schemas is supported by WMQI. Each of these format types can be imported into the tool and used in the message flows to make remapping the data simpler, allowing users to use a ‘drag and drop’ technique between the formats instead of hand coding the data manipulation.

Message enrichment is often used to provide missing information required by the server application but not available to the requesting application. Message enrichment can take many forms including:

- Supplying default values for elements not included in the request
- Additional data retrieved from a database
- Aggregation of more than one message to build the server request
WMQI is aware of the message content and can be used to determine a message’s destination based on an element within the message. This is usually called Intelligent routing and is often used in geographically distributed environments, or where data may be partitioned based on an element, like account number.

WMQI also provides more sophisticated Publish/Subscribe capabilities. The most frequently mentioned is a level of content based subscriptions. In the base MQSeries Publish/Subscribe feature, all subscriptions are based on a message topic. WMQI allows subscribers to use content filters to further restrict subscription information.

An integrated environment that includes DB2 UDB, WebSphere MQ, and WMQI looks as shown in Figure 2-5.

![Figure 2-5 DB2 UDB, WebSphere MQ, and WMQI integrated environment](image)

### 2.2.3 Why XML?

XML is simple, open, universal, extensible, flexible, and controlled. These terms are only a subset of the possible reasoning or justification in answer to the question: Why XML? In fact, XML (eXtensible Markup Language), a W3C standard, is an accepted and viable interface option for many business integration scenarios. XML is a simple text based notation and tagged message format that enables loose coupling of sending and receiving applications.
A pair of tags that surround a piece of data within an XML document is referred to as an element. For example, the “symbol” tags in Figure 2-6 are considered to be one XML element. An attribute can also be associated with an element. In general, attributes have a one-to-one relationship with an associated element, but elements have a many-to-one relationship with an XML document. For example, the “key” variable in Figure 2-6 is considered to be an XML attribute.

XML imposes a measure of control by requiring the presence and proper nesting of user specified tags, allowing for well-formed XML documents. Control is also manifested by optional use of DTD (Document Type Definition) or XML Schema validation mechanisms for checking XML documents with an XML parser.

XML is also a notation for defining tagged languages. This capability allows XML to be widely used and extended by many businesses to create a customized vocabulary for communication with customers and business partners. In essence, one is able to create a unique language for communication. This is due, in part, to the device-independent and platform-independent nature of XML in facilitating communication and information sharing between disparate systems. The content of an XML document is shown in Figure 2-6.

![Sample XML document](image)

**Figure 2-6** Sample XML document

**Note:** Additionally, XML documents can be accessed via the XML wrapper in DB2 Information Integrator.

Though XML, by itself, is an interesting and useful technology, the combination of XML and database technologies provides a more powerful and persuasive argument for using XML. Indeed, there are some computing problems better suited to the use of a database management system. Enabling quick information searching and providing a central repository for storing XML documents are only two of the possible capabilities made available by using a relational database management system.
To this end, IBM has combined the virtues of XML, a database, and a set of integration technologies into a single product offering. The IBM implementation of a database, XML and integration offering is DB2 UDB V8.1 enabled with XML Extender, standards based (ANSI, ISO) SQL/XML extensions (or SQL/XML) and WebSphere MQ for message queuing.

XML support in DB2 UDB is provided by a set of SQL functions, data types, and stored procedures that manipulate XML documents. In general, the stored procedures are used for composition and decomposition, however, a large number of user-defined functions (UDFs) are used for storing intact XML and integration with the file system and WebSphere MQ. In addition, XML Extender provides a set of new data types that are also used for direct storage of intact XML. User-written DB2 UDB client programs can use the SQL, SQLJ, ODBC and/or JDBC application program interfaces (APIs) to access the IBM supplied or user-defined functions and stored procedures to exploit XML. DB2 XML applications can be Java servlets, stored procedures, user-defined functions, PC clients, CICS and IMS, amongst others. And, the application can be written in Java, C, C++, COBOL, REXX, Perl, SQL stored procedures, and many other languages capable of using the DB2 UDB database APIs. An XML Parser, installed separately and used by DB2 UDB, is also shipped with XML Extender.

We will now begin a more detailed discussion of the XML Extender and SQL/XML capabilities available in DB2 UDB V8.1 and DB2 XML Extender.

DB2 Information Integrator contains both DB2 UDB and the DB2 XML Extender. While the term DB2 is used in the following discussion, the XML capabilities are also in DB2 Information Integrator.

**DB2 XML Extender**

DB2 XML Extender provides the stored procedures, user-defined functions and user-defined data types to store, compose, shred, validate and search XML documents in a DB2 UDB database or on a local file system. See Figure 2-7 for a high-level depiction of the DB2 XML Extender components.

**Decomposition (shredding):** This is the process of storing an XML document in DB2 UDB. The XML document is broken apart (or shredded) and the elements are stored as fields in one or more DB2 UDB tables.

**Composition:** This is the process of creating (or composing) an XML document from data in DB2 UDB tables. Elements in the generated XML document are created from fields in one or more DB2 UDB tables. And, the XML document can be stored in DB2 UDB or outside of DB2 UDB, in the file system and WebSphere MQ message queues.
XML Extender is capable of importing or exporting XML documents that are in memory, in the file system or in WebSphere MQ queues. In addition, XML documents can be transformed by using XML style sheets (XSL). The overview diagram in Figure 2-8 is used to show how XML data sources are integrated with DB2 XML Extender.

Figure 2-7   XML Extender components
Document Access Definition (DADs are XML documents used for mapping XML and relational data); and DTD files are stored in the XML Extender tables, XML_USAGE and DTD_REF. The contents of DAD files determine whether an XML column is used to store an XML document in DB2 UDB or an XML collection is used to store or generate an XML document to/from DB2 UDB tables. DTDs are used to validate the structure of XML documents.

XML Extender also uses a subset of the XML Path Language (XPath) to navigate the structure of an XML document. XPath is used by XML Extender functions to identify elements and attributes when extracting or updating in XML columns. In regard to XML collections, XML Extender stored procedures use XPath to override values in the DAD file. Example 2-1 shows two examples of specifying an XPath.

**Example 2-1** Specifying an XPath

/PersonnelRec/Person/Name/Family
/PersonnelRec/Person/@Salary
XML Extender provides the following three storage options for storing XML documents in DB:

- **XML columns**: An XML document is stored, as-is, in an XML enabled column of a database table. This XML column can be updated, retrieved and searched. In addition, the element and attribute data of an XML document can be mapped to DB2 UDB side tables and indexed to exploit the RDBMS search capabilities available in DB2 UDB. Following are some of the reasons that you might choose to use an XML column instead of an XML collection:
  - XML documents already exist.
  - You need to store intact XML documents for archiving or audit purposes.
  - Frequently searched XML elements and attributes are known in advance.
  - Frequently read and rarely updated XML documents are known in advance.
  - You need to keep XML documents external to DB2 UDB on a file system.
XML Extender provides the XMLVarchar, XMLCLOB and XMLFile data types for direct storage of XML data in an XML column. Of the three new data types, XMLCLOB is most frequently used, due to the maximum size limit of two gigabytes. However, the XMLVarchar data type is used less, because the three kilobyte maximum size creates more uncertainty, as application designers cannot guarantee that the size of XML documents will be within the three kilobyte maximum. The XMLFile data type is used to store and manage XML documents externally (outside of the DB2 UDB system) on the file system.

Although the DAD file maps XML to relational data, it is primarily used to configure indexing (optional) of data in an XML column. In addition, the DAD file specifies a DTD file for validating XML documents stored in an XML column.

After an XML document has been stored in an XML column, the user has the ability to extract, update and search XML elements and attributes. The <Xcolumn> tag is used to specify that Xcolumn access and storage is required.

XML collection: XML collections enable the mapping of XML document structures to DB2 UDB tables and facilitates the storing of XML documents in DB2 UDB or the creation of XML documents, outside of DB2 UDB, from content residing in DB2 UDB tables. Basically, an XML document is shredded (or decomposed) and the untagged pieces of data are stored in the columns of one or more DB2 UDB tables. Conversely, an XML document can also be generated (or composed) from data existing in DB2 UDB tables. Following are some of the reasons that you might choose to use an XML collection instead of an XML column:

– Performance of updates is critical.
– You may need to:
  • Generate XML documents from DB2 UDB tables.
  • Generate XML documents from specific columns.
  • Store untagged portions of XML documents for other application access.
  • Drive existing business systems using the document content.
  • Frequently update small parts of an XML document.
  • Process document content with analytical software.

We recommend that XML documents be validated by a DTD file, stored in the DTD repository, prior to shredding in DB2 UDB tables. It should be noted that, for composition, only one DTD can be used to validate generated XML documents. Multiple DTDs can be specified in a DAD file, however, when decomposing XML documents.

The <Xcollection> tag is used to specify that Xcollection access and storage is required.
When using XML collections, a mapping scheme for XML and relational data must be selected and configured in a DAD file. XML Extender provides the SQL statement and RDB node mapping schemes for defining an XML collection in a DAD file.

- **SQL mapping:** This scheme uses an SQL SELECT statement to define the DB2 UDB tables and conditions used for composing an XML document. The SQL mapping scheme enables a direct mapping of relational data to an XML document using a single SQL statement and the XPath data model. The SQL mapping notation is broken into two parts. The first part consists of an SQL statement that provides the scope (e.g. rows and columns) of the relational data to be included in the document. The second part, beginning with the root node, describes the shape of the XML document. It should be noted that the SQL mapping scheme is a one-way notation and cannot be used for decomposition.

- **RDB node mapping:** This scheme uses a two-way mapping notation that employs an XPath-based relational database node (RDB) to compose or decompose an XML document. The DAD file RDB nodes contain definitions for tables, optional columns and conditions that assist XML Extender in determining where to store or retrieve XML data. The RDB node mapping notation is also broken into two parts. The first part is used for scoping, however, there is no SQL statement as is the case for the SQL mapping scheme. Scoping is accomplished by listing the required relational tables, along with their relationships, that you want to appear in the XML document. The second part is used for shaping and describes the mapping of relational data to the element and attribute names in the order and nesting level as needed for appearance in the XML document.

- **Regular CLOB (Character Large Object) column:** XML Extender provides the XMLCLOB UDT, created from the existing CLOB data type, to support XML data in DB2 UDB. XML documents can be stored, however, as a regular CLOB data type.

**DB2 XML Extender and WebSphere MQ**

The WebSphere MQ user-defined functions available in XML Extender are intended to provide database programmers and administrators with a gentle introduction to MQSeries XML message integration with the WebSphere MQ family (including WebSphere MQ, MQSeries Publish/Subscribe and WebSphere MQ Integrator).

These functions are used to pass only XML messages between DB2 UDB and the various WebSphere MQ implementations. Use of DB2 Information Integrator extends the passing of XML messages to non-DB2 sources and the various WebSphere MQ implementations.
In addition, the MQSeries XML Extender functions must be enabled, as in Example 2-2, for access in DB2 UDB.

**Example 2-2  Enable_MQXML command.**

```
enable_MQXML -n DATABASE -u USER -p PASSWORD -
```

All MQSeries XML functions have a DB2XML database schema and are listed in Table 2-1. XML Extender uses XPath and SQL to manipulate XML documents.

**Table 2-1  MQSeries XML functions**

<table>
<thead>
<tr>
<th>MQSeries XML Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2MQ.MQSendXML</td>
<td>Send an XML message to the queue.</td>
</tr>
<tr>
<td>DB2MQ.MQReadXML</td>
<td>A nondestructive read of matching XML message(s) from the queue.</td>
</tr>
<tr>
<td>DB2MQ.MQReadXMLAll</td>
<td>A nondestructive read of all XML messages from the queue.</td>
</tr>
<tr>
<td>DB2MQ.MQReadXMLCLOB</td>
<td>A nondestructive read of matching XML CLOB message(s) from the queue.</td>
</tr>
<tr>
<td>DB2MQ.MQReadXMLAllCLOB</td>
<td>A nondestructive read of all XML CLOB messages from the queue.</td>
</tr>
<tr>
<td>DB2MQ.MQReceiveXML</td>
<td>A destructive read of matching XML message(s) from the queue.</td>
</tr>
<tr>
<td>DB2MQ.MQReceiveXMLAll</td>
<td>A destructive read of all XML messages from the queue.</td>
</tr>
<tr>
<td>DB2MQ.MQReceiveXMLCLOB</td>
<td>A destructive read of matching XML CLOB message(s) from the queue.</td>
</tr>
<tr>
<td>DB2MQ.MQReceiveXMLAllCLOB</td>
<td>A destructive read of all XML CLOB messages from the queue.</td>
</tr>
</tbody>
</table>
**DB2 SQL/XML**

DB2 SQL/XML is a set of SQL extensions implemented as functions in DB2 UDB V8.1 and DB2 Information Integrator. As part of the ANSI SQL standard, DB2 SQL/XML provides a vendor neutral implementation for working with XML and databases. As a result, DB2 SQL/XML is an alternative option for composing an XML document, and is not considered to be part of XML Extender. The generated XML document may be stored in DB2 UDB or outside of DB2 UDB in a file. It should be noted that the DB2 UDB V8.1 SQL/XML set of functions can only be used to compose XML documents. Table 2-2 is a listing of the four SQL/XML functions available in DB2 UDB V8.1.

**Table 2-2 The SQL/XML functions.**

<table>
<thead>
<tr>
<th>SQL/XML Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. XMLATTRIBUTES()</td>
<td>Creates an XML attribute.</td>
</tr>
<tr>
<td>2. XMLELEMENT()</td>
<td>Creates an XML element.</td>
</tr>
<tr>
<td>3. XMLAGG()</td>
<td>Produces a forest of elements from a collection of elements.</td>
</tr>
<tr>
<td>4. XML2CLOB()</td>
<td>Returns the XML document result as a CLOB variable.</td>
</tr>
</tbody>
</table>

**WebSphere MQ integration**

DB2 UDB and DB2 Information Integrator provide access to WebSphere MQ from database applications. The objectives are to combine the set-oriented nature of SQL with messaging, introduce a new data source for federation and facilitate the publishing of database changes to WebSphere MQ. The WebSphere MQ messaging capabilities available in DB2 UDB are implemented as a set of user-defined functions. The entire set of WebSphere MQ functions available in DB2 UDB can be separated into two categories.

First, DB2 UDB and DB2 Information Integrator have MQSeries XML functions that are included with XML Extender. These functions send and receive MQSeries XML messages, and were previously listed in Table 2-1.

Second, DB2 UDB and DB2 Information Integrator have MQSeries text functions that are the basic WebSphere MQ messaging capabilities implemented in DB2 UDB. These functions send and receive traditional MQSeries text messages, and are listed in Table 2-3.
Table 2-3  MQSeries text functions

<table>
<thead>
<tr>
<th>MQSeries Text Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2MQ.MQSend</td>
<td>Send a text message to the queue.</td>
</tr>
<tr>
<td>DB2MQ.MQRead</td>
<td>A nondestructive read of matching text message(s)</td>
</tr>
<tr>
<td>DB2MQ.MQReadAll</td>
<td>A nondestructive read of all text messages from</td>
</tr>
<tr>
<td>DB2MQ.MQReceive</td>
<td>A destructive read of matching text message(s)</td>
</tr>
<tr>
<td>DB2MQ.MQReceiveAll</td>
<td>A destructive read of all text messages from the</td>
</tr>
<tr>
<td>DB2MQ.MQPublish</td>
<td>Publish a text message for a given topic.</td>
</tr>
<tr>
<td>DB2MQ.MQSubscribe</td>
<td>Send a text subscription message to register</td>
</tr>
<tr>
<td></td>
<td>interest in a given topic.</td>
</tr>
</tbody>
</table>

In addition, the MQSeries text functions must be enabled, as in Example 2-3, for access in DB2 UDB and DB2 Information Integrator. Furthermore, the MQSeries text functions have a DB2MQ database schema and are not considered to be part of the XML Extender DB2XML database schema. Refer to 2.2.2, “Why message queueing?” on page 29 for a more detailed explanation of WebSphere MQ.

Example 2-3  Enable_MQFunctions command

```
enable_MQFunctions - DATABASE -u USER -p PASSWORD
```

### 2.2.4 Why Web Services?

Given today’s business climate and requirements, business success (and at times, even survival) requires broad interoperability:

- Intra-enterprise
- Inter-enterprise
- Across heterogeneous sets of applications, platforms, and languages

Enterprises are looking for a single enabling technology to solve the problem of interoperability, and they are finding that solution in Web Services.
What are Web Services?

There are many definitions of Web Services:

- A Web Service is a collection of functions that are packaged as a single entity and published to the network for use by other applications. Web Services are building blocks utilizing the component object model for creating open distributed systems, and allow companies and individuals to quickly and cheaply make their digital assets available worldwide.

  This definition is somewhat loose but illustrates that Web Services is a means to expose one’s enterprise assets to a wide range of applications.

- Web Services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web Services perform functions, which can be anything from simple requests to complicated business processes. Once a Web Service is deployed, other applications (and other Web Services) can discover and invoke the deployed service.

  This is a more formal definition and is defined by IBM.

- A Web Service is a software application identified by an URI, whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts, and supports direct interactions with other software applications using XML-based messages via internet-based protocols.

  This is the working definition of a Web Service that the W3C Web Services Architecture group has agreed upon.

Upon examining the definitions, a Web Service has to support the following:

- Open XML description of the service’ form and function to enable the service to be published, located and invoked/binded

- XML-based messages between different software applications

- Open Transport Protocols as HTTP, SMTP, TCP/IP, and so on

Behind Web Services technologies lies the backdrop framework of Service-Oriented Architecture (SOA).

Service Oriented Architecture (SOA)

In their article, “Cut Loose from Old Business Processes,” Hagel and Brown propose that, to reach the key business objectives of flexibility, collaboration, and business leverage, companies need loosely coupled business processes. These business processes can address current inefficiencies in applications and processes that tie your company to your customers, trading partners or suppliers. They can also help you better compete for customers through integrated, trading-partners collaboration.
The loosely coupled business processes that Hagel and Brown alluded to can be easily and quickly built by utilizing the SOA framework as illustrated in Figure 2-10.

Three major roles can be identified in SOA:

- The service provider creates a Web Service and publishes its interface and access information to the service registry.
- The server broker (also known as service registry) is responsible for making the Web Service interface and implementation access information available to any potential service requestor.
- The service requestor locates entries in the broker registry using various find operations and then binds to the service provider in order to invoke one of its Web Services.
Through SOA, business applications can be easily published and discovered by businesses because they are described in a uniform way on the Internet. The entire “Publish-Find-Bind” process allows e-business applications to be connected more easily both inside and outside the enterprise.

Business processes are best implemented with services, particularly with a service-oriented architecture that supports their development as well as deployment. With a service-oriented architecture, functions within your applications are defined as services. Your business processes are a collection of services — inside and outside your company — connected as needed through a formal interface. Applications that use services need not be aware of the service details at the time the application is developed.

Services and service-oriented architecture offer a giant step in reducing the complexity as well as the costs and risks of new application development and deployment because of the following reasons:

- Reduce intra-enterprise and inter-enterprise development and deployment dependencies.
- Reduce the amount of training developers have to undertake, merely to understand the interface to a particular service, instead of an entire system.
- Reduce size of projects; developers are doing components or services one at a time, instead of working on an entire system.

Web Services is an instance of the service-oriented architecture that enables the provisioning and the consuming of services (Web Services), providing tools and infrastructure to the different organizations’ operators for specifying, publishing, finding and binding services.

Web Services are founded on the service-oriented architecture and possess all the attractive attributes from its architecture, that make Web Services stand as the next generation of service technology that supports build-to-integrate and other integration approaches. Web Services use open, industry standards to define, discover, and deliver new services. If you build your application as a Web Service, it can be more easily connected in the future to other applications and processes.

Web Services are a key innovation for integration:

- **Web Services promote interoperability**: The interaction between a service provider and a service requester is designed to be completely platform and language independent. This interaction requires a WSDL document (see “WSDL overview” on page 52) to define the interface and describe the service, along with a network protocol (usually HTTP). Because the service provider and the service requester have no idea what platforms or languages each other are using, interoperability is a given.
Web Services enable just-in-time integration: As service requesters use service broker to find service providers, the discovery takes place dynamically. Once the requester and provider have found each other, the provider's WSDL document is used to bind the requester and the service together. This means that requesters, providers, and brokers work together to create systems that are self-configuring, adaptable, and robust.

Web Services reduce complexity through encapsulation: Service requesters and providers concern themselves with the interfaces necessary to interact with each other. As a result, a service requester has no idea how a service provider implements its service, and a service provider has no idea how a service requester uses it service. Those details are encapsulated inside the requesters and providers.

Web Services give new life to legacy applications: It is relatively straightforward to take an application, generate a SOAP (see “SOAP overview” on page 49) wrapper, then generate a WSDL document to cast the application as a Web Service. This means that legacy applications can be used in interesting new ways. In addition, the infrastructure associated with legacy applications (security, directory services, transactions, and so on) can be “wrapped” as a set of services as well.

In fact, SOA and Web Services play an important role in the IBM WebSphere business integration directions shown in Figure 1-1 on page 8:

- SOA is established as the base for all five types of business integration.
- Web Services are the base for access and integration.

Web Services overview
Web Services technology is essentially a programming model to aid development and deployment of loosely coupled applications within a company or across industries. The success of the World Wide Web is rapidly expanding the use of Web Services in response to the need for application-to-application communication and interoperability. These services provide a standard means of communication among different software applications involved in presenting dynamic context-driven information to users.

A well architected application, in today’s terms, must be scalable, extensible, and reliable. Today’s applications must be able to execute in a distributed environment. Distributed applications provide benefits such as scalability, extensibility, and reliability. Defining an interface for distributed applications has been a challenge over the years. Language-independent technologies such as CORBA (Common Object Request Broker Architecture) provide a complicated and powerful programming model. Other distributed technologies work well in single language environment, such as ‘C’ RPC (Remote Procedure Call), Java RMI (Remote Method Invocation) and Microsoft’s DCOM (Distributed Common Object Model).
In contrast, Web Services provide an easy-to-understand interface between the provider and consumer of the application resources using a Web Service Description Language (WSDL). Web Services adopt the layered architecture to simplify the implementation of distributed applications, as illustrated in Figure 2-11.

![Web Services layered architecture](image)

The Web Services layered architecture provides the following features:

- Application interface discovery using Universal Description, Discovery, and Integration (UDDI)
- Application interface description using WSDL and again UDDI
- A standard message format using Simple Object Access Protocol (SOAP), which is being developed as a specification in W3C as XML Protocol.
- A standard network protocol using TCP/IP.

**SOAP overview**

The current industry standard for XML messaging is Simple Object Access Protocol (SOAP). SOAP is the basis for the W3C XML Protocol Working Group.

SOAP has the following characteristics:

- SOAP is designed to be simple and extensible.
- All SOAP messages are encoded using XML.
SOAP is transport protocol independent. HTTP is one of the supported transport. Hence, SOAP runs on top the existing Internet infrastructure.

SOAP does not support distributed garbage collection. Therefore, call by reference is not supported by SOAP; a SOAP client does not hold any stateful references to remote objects.

SOAP is operating system independent and not tied to any programming language or component technology. It is object model neutral.

SOAP clients can be implemented independent of technology as long as the clients can issue service request through XML messages. Similarly, the service can be implemented in any language, platform as long as it can process XML messages and package XML messages as responses.

Originally, SOAP was created to be a network and transport neutral protocol to carry XML messages around. SOAP over HTTP became the premier way of implementing this protocol, to the point that the latest SOAP specification mandates HTTP support. Conceptually, there is no limitation for the network protocol that can be utilized. For example, because HTTP is a transport that does not guarantee delivery and is non-transactional, SOAP messages can also be transferred by a message software such as WebSphere MQ.

SOAP Remote Procedure Call (RPC) is the latest stage in the evolution of SOAP; the body of a SOAP message contains a call to a remote procedure and the parameters to pass in. Both, the call and the parameters are expressed in XML.

**Overall message format**

SOAP messages (see structure in Example 2-4) have these characteristics:

- A SOAP message is an envelope containing zero or more headers and exactly one body. The envelop is the top element of the XML document, providing a container for control information, the addressee of a message, and the message itself.

- Headers transport any control information such as quality of service attributes. The body contains the message identification and its parameters.

- Both the headers and the body are child elements of the envelope.

**Example 2-4  SOAP message structure**

```xml
<SOAP-ENV:Envelope.... >
    <SOAP-ENV:Header name="nmtokens">
        <SOAP-ENV:HeaderEntry.... />
    </SOAP-ENV:Header>
    <SOAP-ENV:Body name="nmtoken">
        [message payload]
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope.... >
```
Example 2-5 is an example of a SOAP request.

**Example 2-5   SOAP message example: Sample request**

```xml
<SOAP-ENV:Envelope
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >

 <SOAP-ENV:Body>
   <ns1:echoString xmlns:ns1="http://soapinterop.org/">
     <string-arg xsi:type="xsd:string">Welcome to Web Services!</string-arg>
   </ns1:echoString>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Encoding rules**

Encoding rules have these characteristics:

- Encoding rules define a serialization mechanism that can be used to exchange instances of application-defined data types.
- SOAP defines a programming language independent data type scheme based on XSD, plus encoding rules for all data types defined according to this model.

**RPC representation**

Here are some characteristics of the RPC representation:

- The RPC representation is a convention suited to represent remote procedure calls and the related response messages.
- The usage of this convention is optional. If the RPC convention is not used, the communication style is purely message oriented. When working with the message oriented style, also called document oriented communication, almost any XML document can be exchanged.

**Namespaces**

SOAP defines the XML namespaces shown in Table 2-4.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace URI</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP-ENV</td>
<td><a href="http://schemas.xmlsoap.org/soap/Envelope/">http://schemas.xmlsoap.org/soap/Envelope/</a></td>
<td>SOAP envelope</td>
</tr>
</tbody>
</table>
More information
You can discover details on what is new in SOAP and download SOAP specifications from:

http://www.w3.org/2000/xp/Group/

For information on Apache SOAP, check out the user and API documentation as well as the FAQ list at:

http://www.apache.org/soap

WSDL overview
If we want to find services automatically, we require a way to formally describe both their invocation interface and their location. The Web Services Description Language (WSDL) V 1.1. provides a notation serving these purposes.

WSDL allows a service provider to specify the following characteristics of a Web Service:

- Name of the Web Service and addressing information
- Protocol and encoding style to be used when accessing the public operations of the Web Service
- Type information: operations, parameters, and data types comprising the interface of the Web Service, plus a name for this interface

A WSDL specification uses XML syntax, therefore, there is an XML schema that defines the WSDL document. A valid WSDL document consists of two parts:

1. The interface part describing the abstract type interface and its protocol binding. This part of the document contains the following top-level elements:
   - **Port type** An abstract set of one or more operations supported by one or more ports. Each operation defines an in and an out message as well as an optional fault message.
   - **Message** An abstract, typed definition of the data being communicated. A message can have one or more typed parts.
   - **Type** A container for data type definitions using some type system such as XSD.
   - **Binding** A concrete protocol and data format specification for a particular port type. The binding information contains the protocol name, the invocation style, a service ID, and the encoding for each operation.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace URI</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP-ENV</td>
<td><a href="http://schemas.xmlsoap.org/soap/encoding/">http://schemas.xmlsoap.org/soap/encoding/</a></td>
<td>SOAP serialization</td>
</tr>
</tbody>
</table>
2. The implementation part describing the service access (location) information. The service implementation document contains the following elements:

- **Service**: A collection of related ports
- **Port**: A single endpoint, which is defined as an aggregation of a binding and a network address

Let us examine an example of a complete and valid WSDL specification. We break up one WSDL specification into five parts. Each part deals with a particular element that is found in either the service implement document or the service interface document. With one exception, we lump service and port elements into a single example.

The five parts are as follows:

- Namespaces and type element, as shown in Example 2-6.

### Example 2-6  WSDL part I: namespace and type element

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions

targetNamespace="http://tempuri.org/toy/ITSOGroup/ITSOSelectEmployeesDadx.dadx/WSDL"

xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://tempuri.org/toy/ITSOGroup/ITSOSelectEmployeesDadx.dadx/WSDL"
xmlns:xsd1="http://tempuri.org/toy/ITSOGroup/ITSOSelectEmployeesDadx.dadx/XSD">

<types>
   <schema
targetNamespace="http://tempuri.org/toy/ITSOGroup/ITSOSelectEmployeesDadx.dadx/XSD"

 xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://tempuri.org/toy/ITSOGroup/ITSOSelectEmployeesDadx.dadx/XSD">
   <element name="selectEmployeesResult">  
      <complexType>
         <sequence>
            <element maxOccurs="unbounded" minOccurs="0"
name="selectEmployeesRow">
               <complexType>
                  <sequence>
                     <element name="EMPNO" type="string"/>
                     <element name="FIRSTNME" type="string"/>
```
<element name="LASTNAME" type="string"/>
<element name="PHONENO" nillable="true" type="string"/>
</complexType>
</element>
</complexType>
</element>
<element name="selectEmployeesResponse">
<complexType>
<sequence>
<element name="return">
<complexType>
<sequence>
<element ref="tns:selectEmployeesResult"/>
</sequence>
</complexType>
</element>
</sequence>
</complexType>
</element>
</schema>
</types>

The type element encloses data type definitions used by the exchanged messages. WSDL uses XML schemas definitions (XSDs), as it is a canonical and built in type system.

- Message element, as shown in Example 2-7.

**Example 2-7  WSDL part II: message element**

```
<message name="selectEmployeesInput"/>
<message name="selectEmployeesSoapOutput">
<part element="xsd1:selectEmployeesResult" name="return"/>
</message>
<message name="selectEmployeesGetPostOutput">
<part element="xsd1:selectEmployeesResponse" name="response"/>
</message>
```

A message represents one interaction between service requestor and service provider. If an operation is bidirectional (a RPC returning a result, for example), two messages are used in order to specify the transmitted on the way to and from the service provider.
Port type element, as shown in Example 2-8.

Example 2-8  WSDL part III: port type element

```xml
<portType name="theSoapPortType">
  <operation name="selectEmployees">
    <wsdl:documentation
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns="http://www.w3.org/1999/xhtml">
    </wsdl:documentation>
    <input message="tns:selectEmployeesInput"/>
    <output message="tns:selectEmployeesSoapOutput"/>
  </operation>
</portType>

<portType name="theGetPostPortType">
  <operation name="selectEmployees">
    <wsdl:documentation
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns="http://www.w3.org/1999/xhtml">
    </wsdl:documentation>
    <input message="tns:selectEmployeesInput"/>
    <output message="tns:selectEmployeesGetPostOutput"/>
  </operation>
</portType>
```

A port type is a named set of abstract operations and the abstract messages involved. WSDL defines four types of operations that a port can support:

**One-way** The port receives a message. Only an input message is defined.

**Request-response** The port receives a message, and sends a correlated message. An input message followed by an output message are defined.

**Solicit-response** The port sends a message, and receives a correlated message. An output message followed by an input message are defined.

**Notification** The port sends a message. Only an output message is defined. This type of operation is well fitted in a publish/subscribe scenario.
- Binding element, as shown in Example 2-9.

**Example 2-9  WSDL part IV: binding element**

```xml
(binding name="theSoapBinding" type="tns:theSoapPortType">
    <soap:binding style="rpc">
        transport="http://schemas.xmlsoap.org/soap/http">
            <operation name="selectEmployees">
                <soap:operation soapAction="http://tempuri.org/ITSOGroup/ITSOSelectEmployeesDadx.dadx"/>
                    <input>
                        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
                        namespace="http://tempuri.org/ITSOGroup/ITSOSelectEmployeesDadx.dadx"/>
                        parts=""/>
                        use="encoded"/>
                    </input>
                    <output>
                        <soap:body namespace="http://tempuri.org/ITSOGroup/ITSOSelectEmployeesDadx.dadx"/>
                        parts="return"/>
                        use="literal"/>
                    </output>
                </operation>
            </binding>
            <binding name="theGetBinding" type="tns:theGetPostPortType">
                <http:binding verb="GET"/>
                <operation name="selectEmployees">
                    <http:operation location="selectEmployees"/>
                        <input>
                            <http:urlEncoded/>
                        </input>
                        <output>
                            <mime:mimeXml/>
                        </output>
                </operation>
            </binding>
            <binding name="thePostBinding" type="tns:theGetPostPortType">
                <http:binding verb="POST"/>
                <operation name="selectEmployees">
                    <http:operation location="selectEmployees"/>
                        <input>
                            <mime:content type="application/x-www-form-urlencoded"/>
                        </input>
                        <output>
                            <mime:mimeXml/>
                        </output>
                </operation>
            </binding>
        </soap:operation>
    </binding>
</binding>
```
A binding contains:
- Protocol specific general binding data such as underlying transport protocol and the communication style for SOAP
- Protocol extensions for operations and their messages, including the URN and encoding information for SOAP

Each binding references one port type; one port type can be used in multiple bindings. All operations defined within the port type must be bound in the binding.

- Service and port elements, as shown in Example 2-10.

**Example 2-10  WSDL part V: service and port elements**

```xml
<service name="theService">
  <port binding="tns:theSoapBinding" name="theSoapPort">
    <soap:address location="http://a78rp1nh:8080/toy/ITSOGroup/ITSOSelectEmployeesDadx.dadx/SOAP"/>
  </port>
  <port binding="tns:theGetBinding" name="theGetPort">
    <http:address location="http://a78rp1nh:8080/toy/ITSOGroup/ITSOSelectEmployeesDadx.dadx/"/>
  </port>
  <port binding="tns:thePostBinding" name="thePostPort">
    <http:address location="http://a78rp1nh:8080/toy/ITSOGroup/ITSOSelectEmployeesDadx.dadx/"/>
  </port>
</service>
</definitions>
```

A service definition merely bundles a set of ports under a name. A port definition describes an individual endpoint by specifying a single address for a binding. Any port in the implementation part must reference exactly one binding previously defined in the same document.

**Namespaces**

WSDL uses the XML namespaces listed in Table 2-5.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace URI</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsdl</td>
<td><a href="http://schemas.xmlsoap.org/wsdl/">http://schemas.xmlsoap.org/wsdl/</a></td>
<td>Namespace for WSDL framework</td>
</tr>
<tr>
<td>soap</td>
<td><a href="http://schemas.xmlsoap.org/wsdl/soap/">http://schemas.xmlsoap.org/wsdl/soap/</a></td>
<td>SOAP binding</td>
</tr>
<tr>
<td>http</td>
<td><a href="http://schemas.xmlsoap.org/wsdl/http/">http://schemas.xmlsoap.org/wsdl/http/</a></td>
<td>HTTP binding</td>
</tr>
</tbody>
</table>
The first four namespaces are defined by the WSDL specification itself; the next four definitions reference namespaces defined in the SOAP and XSD standards. The last one is local to each specification. Note that in our example we do not use real namespace; the URLs contain localhost.

### More information

The best sources for more information on WSDL are the IBM Web Services Toolkit (WSTK) and the WSDL V1.1 specification. WSDL V1.1 is contained in the IBM WSTK or it can be found at:

http://www.w3.org/TR/wsdl

### UDDI overview

UDDI stands for universal description, discovery, and integration. UDDI is a technical discovery layer. It defines:

- The structure for a registry of service providers and services
- The API that can be used to access registries with this structure
- The organization and project defining this registry structure and its API

UDDI is a search engine for application clients, and there is a browser interface to view and search for entries stored in the registry.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace URI</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mime</td>
<td><a href="http://schemas.xmlsoap.org/wsdl/mime/">http://schemas.xmlsoap.org/wsdl/mime/</a></td>
<td>MIME binding</td>
</tr>
<tr>
<td>soapenc</td>
<td><a href="http://schemas.xmlsoap.org/soap/encoding/">http://schemas.xmlsoap.org/soap/encoding/</a></td>
<td>Encoding namespace as defined by SOAP 1.1</td>
</tr>
<tr>
<td>soapenv</td>
<td><a href="http://schemas.xmlsoap.org/soap/envelope/">http://schemas.xmlsoap.org/soap/envelope/</a></td>
<td>Encoding namespace as defined by SOAP 1.1</td>
</tr>
<tr>
<td>xsi</td>
<td><a href="http://www.w3.org/2000/10/XMLSchema-instance">http://www.w3.org/2000/10/XMLSchema-instance</a></td>
<td>Instance namespace as defined by XSD</td>
</tr>
<tr>
<td>xsd</td>
<td><a href="http://www.w3.org/2000/10/XMLSchema">http://www.w3.org/2000/10/XMLSchema</a></td>
<td>Schema namespace as defined by XSD</td>
</tr>
<tr>
<td>tns</td>
<td>(URL to WSDL file)</td>
<td>The this namespace (tns) prefix is used as a convention to refer to the current document. Do not confuse it with the XSD target namespace, which is a different concept!</td>
</tr>
</tbody>
</table>
Registry structure
The following entities comprise the core of the UDDI data model:

Business entity
Business entities are the white and yellow pages of the registry. Business entities provide business information about a service provider such as business name, contacts, description, identifiers, and categories.

Business service
The green pages, part 1 provide nontechnical service information. A business service is a descriptive container used to group a set of Web Services related to a business process or group of services. Categorization is available on this level. A business service maps to a WSDL service.

Binding template
The green pages, part 2, containing service access information. These are the technical Web Service descriptions relevant for application developers who want to find and invoke a Web Service. A binding template points to a service implementation description (for example, a URL). This entity is sometimes also called access locator. A binding template maps to a WSDL port.

tModel
A tModel is a technical fingerprint, holding metadata about type specifications as well as categorization information. Its attributes are key, name, optional description, and URL. The simplest tModel contains some text characterizing a service. A tModel points to a service interface description (for example, through a URL). The type specification itself, which can be a WSDL document or any other formal specification, is not part of the UDDI model. This entity is sometimes also called service type.

Namespaces
UDDI defines the XML namespaces listed in Table 2-6:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace URI</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(none)</td>
<td>urn:uddi-org:api</td>
<td>Target namespace for the UDDI data model</td>
</tr>
<tr>
<td>xml</td>
<td><a href="http://www.w3.org/1999/XMLSchema">http://www.w3.org/1999/XMLSchema</a></td>
<td>Schema namespaces as defined by XSD</td>
</tr>
</tbody>
</table>
More information
The core information model is defined in an XML schema that can be found at:

http://www.uddi.org/schema/uddi_1.xsd

To learn more about what IBM is doing with Web Services and UDDI, you can visit:

http://www-3.ibm.com/services/uddi/

The Web site of UDDI organization is rich in content; you can find all UDDI specifications there, for example, the data model and API for Version 1 and 2. Moreover, there are technical white papers, a best practices section, and an informative list of frequently asked questions:

http://www.uddi.org

Web Services integration
Figure 2-12 illustrates how DB2 UDB and DB2 Information Integrator are going to utilize Web Services to:

► Provide access to data that is stored inside DB2 UDB to additional clients; in this case, they are the SOAP clients (see the right hand side of Figure 2-12).

► Act as a consumer of Web Services to bring external data and serve the data as part of the data stored in DB2 UDB (see the left hand side of Figure 2-12).

In this book, we will only concentrate on examining DB2 UDB acting as a Web Services provider.
DB2 UDB acting as a Web Service provider simply means exposing DB2 UDB data through the Web Services interface. With DB2 Information Integrator, non-DB2 data can also be exposed through the Web Services interface. Traditionally, database application developers build custom applications and wrap the applications with the Web Services interface. DB2 UDB provides tools that are integrated into the WebSphere product suite to ease the effort of creating Web Services using DB2 UDB data (see Figure 2-13).
Figure 2-13  DB2 UDB as Web Services provider: implementation options

Whether you build custom Web Services applications or use the Web Services Object Runtime Framework (WORF) to generate your Web Services through some definition files, in this case, we use the Document Access Definition Extension (DADX) files, your SOAP clients are using the same interface and accessing the same data that is stored in the database.

- For more information on WORF, see “Web Services Object Runtime Framework” on page 125.
- For more information on DADX, see “DADX overview and structure” on page 127.
Summary
The industry is moving towards the service-oriented architecture. IBM WebSphere business integration leverages SOA to provide a common programming model and utilizes Web Services as the enabling technology to access information and to integrate applications.

The many values of Web Services include, but are not limited to, the following:

- Promoting new levels of reuse
- Improving time to market
- Facilitating new levels of integration
- Unifying disparate computing paradigms
- Linking diverse underlying technologies
- Providing consistent business/technical vocabulary
- Spanning the entire software development cycle
- Crossing enterprise boundaries

Through DB2 UDB and Web Services integration, we can truly provide universal access to the DB2 UDB data.

2.2.5 Why data warehousing?

DB2 UDB Data Warehouse Center provides a distributed, heterogeneous infrastructure for designing, building, and maintaining data warehouses. It is included at no additional cost with DB2 UDB on Windows, UNIX, and Linux platforms, and offers the following functions:

- Register and access data sources.
- Define data extraction and transformation steps.
- Populate target databases.
- Automate and monitor warehouse management processes.
- Manage and interchange metadata.

As depicted in Figure 2-14, Data Warehouse Center supports data extraction from a wide variety of data sources like relational databases, OLAP Servers, and flat files. It supports full refresh and incremental update data movement options, including leveraging the power of IBM's integrated data replication functions.
Integrated SQL Assist capabilities let customers define transformations using the rich SQL. Users can also build custom transformation routines using the DB2 Development Center. The Warehouse Launchpad simplifies the job of populating a warehouse by leading you through the related tasks.

The Data Warehouse Center components Process Builder, System Monitor and Scheduler let you define, test, deploy, and monitor the entire process of data extraction, transfer/transformation, and final load into a target database.

A single metadata store is used by the Data Warehouse Center for the management and interchange of technical metadata. One of the objectives of this metadata store is to act as a central control point for managing warehouse construction operations done both by IBM products and by products from business partners like Hyperion Solutions, Evolutionary Technologies International, and Vality Technology. The metadata store also supports the OMG CWMI specification with technology and tools that handle metadata interchange based on XML, and a common set of UML-based data warehouse metamodel.
2.3 Information integration at HLED Investments

Our sample company is HLED Investments, whose primary business is individual stock and mutual fund investment accounts. It is our supposition that HLED has been in business for some time, and has a good track record of managing their information and applications. However, they have identified several areas where they can improve on their current model, to provide more consistency and easier access to information for current and new in-house applications and provide additional value and for-fee services to their external customer base.

Currently they have several disparate data sources they use when processing customer portfolio requests:

- The **Stock Portfolio** table that is stored on an Informix RDBMS. The definition of this source is shown in Example 2-11.

  **Example 2-11  Create Stock portfolio**

  ```sql
  CREATE TABLE INFX_STOCKS
  (CUST_ID CHARACTER (10)  NOT NULL,
   STOCK_SYMBOL CHARACTER (5)  NOT NULL,
   TOTAL_SHARES INTEGER NOT NULL,
   CONSTRAINT CC1034013402107
   PRIMARY KEY (CUST_ID, STOCK_SYMBOL))
  OPTIONS (REMOTE_SERVER 'INFMX_SRC', REMOTE_SCHEMA 'db2drs4',
   REMOTE_TABNAME 'Stock_Portfolio');
  ```

  The sample data used is shown in Example 2-12.

  **Example 2-12  Stock portfolio sample data**

  1001  
  'IBM  ', 00005231
  '1001  
  'QRS1 ', 25
  '1001  
  'RV60 ', 00000922
  '1003  
  'IBM  ', 00004356
  '1003  
  'CNB7 ', 00092567
  '1002  
  'IBM  ', 263
  '1002  
  'ALEX ', 5380

- The **Mutual Fund Portfolio** that is stored on a DB2 UDB RDBMS. The definition of this source is shown in Example 2-13.

  **Example 2-13  Create mutual fund portfolio**

  ```sql
  CREATE TABLE MUTUALFUNDPORTFOLIO
  (**CUSTOMERID** CHARACTER (10)  NOT NULL,
   **MUTUAL_FUND_SYMBOL** CHARACTER (5)  NOT NULL,
   **QUANTITY** DECIMAL (11, 3)  NOT NULL  WITH DEFAULT 0.000,
   CONSTRAINT CC1034013402107
   PRIMARY KEY (**CUSTOMERID**, **MUTUAL_FUND_SYMBOL**))
  OPTIONS (REMOTE_SERVER 'INFMX_SRC', REMOTE_SCHEMA 'db2drs4',
   REMOTE_TABNAME 'Mutual_Fund_Portfolio');
  ```
"FIRSTNAME" CHARACTER (20),
"MI" CHARACTER (1),
"LASTNAME" CHARACTER (20),
"STREETADR1" CHARACTER (30),
"STREETADR2" CHARACTER (30),
"STREETADR3" CHARACTER (30),
"CITY" CHARACTER (30),
"STATE" CHARACTER (20),
"ZIP" CHARACTER (10),
"CILASTUPDATE" CHARACTER (10),
PRIMARY KEY (CUSTOMERID, MUTUAL_FUND_SYMBOL))
DATA CAPTURE NONE IN USERSPACE1

The sample data is shown in Example 2-14.

Example 2-14  Mutual fund sample data

"1003      ","FGFND", 00000100.225,"Ian                 ", "Harvey
","111 Toronto Way             ","Penthouse Suite              ",
", "Whitby ", "ON                  ", "LIN5K1
", "20020924        
"1001      ","FGFND", 00000050.150,"Fred                ", "Lo
","123 Bay St                    ",
", "Vancouver ", "BC                  ", "V7E2R9
", "20011011        
"1002      ","FGFND", 00000025.250,"Carolyn             ", "Elkins
","301 Old Mill Cove             ","Dungeon                 ", "Cell 255-D
", "Woodstock                   ", "GA                  ", "30018
", "19880208           

► Customer Information file that is stored on a VSAM file accessed via CICS transactions: The definition of this source and the sample data are shown in Appendix “C” on page 311.

► Current Stock and Mutual Fund Valuations stored in an XML file: The definition of this source and the sample data are shown in Appendix E, “Web Services XML and DADX files” on page 343.

► Orders transactions stored on a DB2 UDB RDBMS: The definition of this source and the sample data are shown Example 2-15.

Example 2-15  Create orders table

CREATE TABLE ORDERS
  ("ORDER_NUMBER" CHARACTER (10) NOT NULL ,
   "SYMBOL" CHARACTER (5) NOT NULL ,
   "ACTION_INDICATOR" CHARACTER (1) NOT NULL ,
   "TRANSACTION_DATE" DATE ,
   ...
HLED has a wide variety of applications consuming this data and has found that with a few limited exceptions the data from the first three is always combined into a customer portfolio before it is used. There are also a number of applications that combine the customer portfolio information with the current valuations to create a valued portfolio image. For HLED information integration contains elements of both data integration, federating data from various sources, and application integration, to retrieve data from application bound stores.

HLED has a number of technical issues for information integration:

- In “Business case I: Federating data access using SQL” on page 77:
  - To make accessing related information, like the complete investments portfolio, simpler for their applications
  - To isolate changes made to the data sources from the applications as much as possible, so that as they do more fully integrate their data the application impact can be minimized
  - To reduce their ever-growing Web of connections to multiple data sources

- In “Business case II: Accessing your information through Web Services” on page 119:
  - To provide portfolio information to both current and new external customers
  - To provide platform independence; integrating systems regardless of their implementations
  - To use open standards, protocol and network transport
  - To reduce tight coupling between business functions and integration

- In “Business case III: Incorporating XML data” on page 193:
  - To provide clients with the option to send and receive investment transactions as XML documents
  - To shred (parse) store XML documents into a DB2 UDB table
  - To optionally use WebSphere MQ as the transport mechanism for the XML transactions

- In “Business Case IV: Populating a Common Data Store” on page 215:
  - To provide portfolio information to customer through new service channel (Investment Portal, Web Services)
– To have the provided portfolio information consistent for all Customer Service Applications (Call-Center, Investment Portal, Web Services)
– To make accessing related information, like the complete investments portfolio, simpler for their applications

We have selected four scenarios, each demonstrating the ways information integration can be used to begin HLED’s journey towards full information integration. The information integration functions included in DB2 UDB are used in these scenarios:

1. **Implementing federated access and MQSeries Integration**: This scenario sets up the initial information integration infrastructure.
2. **Accessing your information through Web Services**: This scenario builds on the infrastructure developed in scenario 1), and demonstrates using DB2 UDB as a provider of Web Services.
3. **Database Access Using XML Extender, SQL/XML and WebSphere MQ**: This scenario builds further, demonstrating DB2 UDB’s XML capabilities adding that valuable tool to the information integration architecture.
4. **Populating a common data store**: This scenario uses the information integration capabilities of federated access and MQSeries Integration to build a common data store.

### 2.4 Target environment

In most case, we do not have the luxury to choose a set of products and start the implementation from scratch. Typically, enterprises have their IT infrastructure and current working environment. We can almost guarantee that the infrastructure and working environment contain many different standards, network transports and protocols, operating systems, and diverse data sources.

There is no difference in our scenario. HLED is using Informix as their enterprise data store with the customer profile information stored in a host environment. HLED uses:

- CICS TS Version 2.2 running on Z/OS and data stores in VSAM

HLED acquired a mutual fund company which stores their data in:

- DB2 UDB Version 7.2 running on Windows.
The first challenge for HLED Investments is to consolidate all the different data sources. HLED has decided to use a couple of technologies found in DB2 UDB V8.1 to help consolidate all the different data:

- Federated access to Informix and DB2 UDB data
- Access to WebSphere MQ to access VSAM data.

Since federated access to Informix and access to WebSphere MQ comes as a standard install for DB2 UDB v8.1, we do not need any additional package or extension to support access to those two data sources. However, if we are to include federated access to Oracle, Sybase, Microsoft, and other heterogeneous sources, we need other information integration products. DB2 Information Integrator would be used once it has become generally available. Prior to that time DB2 Relational Connect would be used.

After the HLED Investments, clients were able to see their portfolio information, but they wanted to see the market values on their portfolios. Stock and mutual fund quotes are services provided by some external organization. The best way to get this information is through the use of Web Services. The external system provides HLED with an XML representation when HLED requests a stock quote. HLED will decompose the XML document and store the XML as relational data in the database. We need DB2 XML Extender for the XML decomposition. DB2 XML Extender is part of the standard distribution of DB2 UDB V8.1. Now, we can join the portfolio and the stock quote by using regular SQL.

Lastly, HLED will have to expose the data in a federated server as Web Services to Almaden Credit Union. DB2 UDB V8.1 is also shipped with Web Services support. However, the federated server has to be installed with the following software:

- DB2 UDB V8.1
- Informix Client SDK V2.50 and above
- IBM WebSphere MQ V5.3
- IBM MQSeries AMI version 1.2.0 and above
- IBM WebSphere Application Server version 4.03 and above
- IBM WebSphere Studio Application Developer for development purpose

On the host machine, we have to install:

- IBM WebSphere MQ V5.2

The sections following show the software and hardware configurations of HLED Investments for the different business cases.
2.4.1 Technical infrastructure: Business Case I

The technical environment for Business Case I at HLED Investments is illustrated in Figure 2-15.

![Figure 2-15  HLED technical environment - Business Case I](image)

**Note:** For these scenarios we used Windows servers for our distributed environments. This was for convenience, not due to product necessity.

**Development and runtime software components**

For the runtime environment we used the following software infrastructure components:

- Microsoft Windows 2000, service pack 3 on all distributed servers
- Microsoft Internet Explorer, version 6.0 on all distributed servers
- IBM DB2 Universal Database Enterprise server, version 8.1 on COPPER
- IBM DB2 Universal Database Enterprise server, version 7.2 fixpack 5 on A78RPLPW
- IBM WebSphere MQ for Windows, version 5.3
- IBM MQSeries AMI, version 1.2.0 for MQSeries Integration
- Informix V 9.21.TC5-1
- Informix Client-SDK 2.50
- IBM z/OS operating system
- IBM CICS Transaction Server, version 2.2
- IBM WebSphere MQ for z/OS, version 5.3
2.4.2 Technical infrastructure: Business Case II

The technical environment for Business Case II at HLED Investments is illustrated in Figure 2-16.

As shown in Figure 2-16, Business Case II uses the same basic development and runtime software components, with the following additions:

- IBM WebSphere Application Server, version 4.03
- IBM WebSphere Studio Application Developer, version 4.03
2.4.3 Technical infrastructure: Business Case III

The technical environment for Business Case III at HLED Investments is illustrated in Figure 2-17.

The Almaden Credit Union will use a combination of the following products to implement each of the solution alternatives.

**Development and runtime software components**

For the runtime environment we used the following software infrastructure components:

- IBM WebSphere Studio Application Developer, version 4.0.3
- IBM DB2 UDB version 8.1 Development Center
- IBM DB2 UDB version 8.1 with XML Extender, federation and WebSphere MQ functions enabled; DB2 UDB employs the AMI application program interface to send and receive messages with WebSphere MQ queues
- IBM WebSphere Application Server, version 4.03
- IBM WebSphere MQ, version 5.3
- Microsoft Windows 2000, service pack 3
Microsoft Internet Explorer, version 5.0

**Hardware configuration**
This is the hardware configuration we used:
- **Server**: An Intel based machine with 576 MB of RAM and a 20 GB disk drive
- **Client**: A client workstation equipped with Internet access and running a Microsoft Internet browser (version 5.0)

### 2.4.4 Technical Infrastructure: Business Case IV

The system environment that was used for the CDS scenario included four servers with different OS platforms. Figure 2-18 shows a high level description of the servers.

![Figure 2-18 System environment for the CDS scenario](image)

**Development and runtime software components**
For the runtime environment we used the following software infrastructure components:
- DB2 UDB on Windows 2000, hosting the Common Data Store
- Data Warehouse Center as the core component of the data populating subsystem
- MQSeries Integration or UDFs to access WebSphere MQ queues
- WebSphere MQ for asynchronous data transfer between the mainframe server and the CDS
- WebSphere MQ CICS Adapter for accessing the Customer VSAM data set via CICS
- DB2 UDB on AIX, hosting the Mutual Fund tables
- Informix XPS 9.3 on Linux, hosting the Stock Portfolio tables

Figure 2-19 depicts the implemented software component on each server.

**Figure 2-19  Runtime environment**

Here is a brief description of the software components that formed the base of the CDS infrastructure. Furthermore, we explain the context in which the components were used.

**DB2 UDB Version 8.1 on AIX**
The DB2 UDB server on the C55VK69R machine was used as the home for CDS instance and database.

**Data Warehouse Center**
It allows you to move data directly from source-to-target and offers a wide variety built-in transformers for data cleansing and statistical transformations. You can also model/define, schedule, maintain, and monitor your data populating processes with Data Warehouse Center.

In our CDS scenario the Data Warehouse Center is the core component for the data population infrastructure.
**WebSphere MQ**
WebSphere MQ offers message transport mechanism that enables guaranteed delivery to applications that are internal and external to an enterprise. DB2 UDB stored procedures and user-defined functions and triggers can be used to send and receive text or XML messages to/from MQ queues. In addition, MQ queues can be used as a data source for SQL calls.

In our scenario message queuing provides a high level transport mechanism for the data transfer between our mainframe (CICS/VSAM) system and the Common Data Store.

**MQSeries Integration**
The MQSeries Integration is basically a set of DB2 User Defined Functions (UDFs) that provide seamless access to WebSphere MQ messaging via standard SQL statements. This UDF can be used to send and receive text or XML messages to/from WebSphere MQ queues.

Example 2-16 is a brief example how to send a message containing customer information for each CUSTOMER record that got updated today.

```sql
Example 2-16  MQSeries Integration example
SELECT MQSEND(LASTNAME || ' ' || FIRSTNAME || ' ' ||
CUSTOMERID) FROM CUSTOMER WHERE INSERTDATE = CURRENT DATE
```

In our data populating subsystem for CDS, WebSphere MQ queuing is one of the transport mechanisms we used for source-to-target data transfers. On the receiver site of these transfers, DB2 UDFs are used to receive data from a queue.

**MQSeries AMI**
The MQSeries Application Messaging Interface (AMI) provides an MQ interface wrapper designed to allow application access to WebSphere MQ resources, without needing to understand and code all available functions and options available via the MQI. Through the AMI concept of *Service Points and Policies*, applications that want access to a queue just need to know by which services and policies these particular resources are controlled.
Chapter 3. Business case I: Federating data access using SQL

The basic component of information integration is data level integration. While the ultimate goal is a fully integrated data store containing all of an enterprise’s data in a relational model, accessible from any application, this utopia does not usually exist. Databases may be distributed to allow a higher level of performance, purchased vendor applications may require using different database managers, in-house developed systems may be using different database management systems, and business mergers and acquisitions may occur. These are all very valid reasons for the multiple data sources, data store types, and data locations that confront enterprises today.

In this chapter we discuss how to federate data access to multiple disparate data stores with a single SQL statement. Using this technique, applications get a completely integrated view of the enterprise data.

Our model will federate two fully relational sources, DB2 UDB and Informix, and will use the MQSeries Integration to retrieve data from a VSAM (non-relational) source using a CICS transaction.

Because the sources to be federated are DB2 UDB and Informix only, the information integration functions included in DB2 UDB are used. If access to non-DB2 sources such as Oracle or Microsoft SQL Server was required, then DB2 Information Integrator would be needed.
3.1 Business problem statement

HLED Investments corporation sells two types of investments to their customers, both individual stocks and mutual funds. The data stores for the two investment types are both relational, but use different RDBMS because one was developed in-house and the other was a purchased application. In addition, their customer information is primarily kept on z/OS VSAM files, accessed and maintained via CICS transactions.

One of the most common functions performed at HLED is to access a customer’s complete stock and mutual fund portfolio. This is done for a number of business reasons, including customer inquiries, both Web-enabled and call center applications; and more batch oriented functions such as statement preparation.

Currently, to present a complete portfolio “image” their applications must access all three data sources. This requires connections to each RDBMS and a connection to WebSphere MQ to initiate the CICS transaction to retrieve their legacy data (see the current data access code flow example, Example 3-1).

Example 3-1 Current data access code flow

Connect to DB2 database
Select * from MutualFundPortfolio for this customer ID
For each investment returned from the MutualFundPortfolio Select:
   Set Display.CustInvSymbol = Mutual Fund symbol
   Set Display.CustInvQuantity = Mutual Fund Quantity
Disconnect from DB2 database

Connect to Informix database
Select * from Stock Portfolio for this customer ID
For each investment returned from the Stock Portfolio Select:
   Set Display.CustInvSymbol = Stock symbol
   Set Display.CustInvQuantity = Stock Quantity
Disconnect from Informix datastore

Connect to MQSeries queue manager (MQCONN)
Open Request_Queue
PutCustomer Info Request
Close Request_Queue
Open Reply_Queue
Retrieve Customer Info Reply from Reply_Queue
Parse the reply into the customer information fields
CloseReply_Queue
Disconnect from queue manager
Set Display.CustomerName = Message.Customer_Name
Set Display.Customer Address = Message.Customer_Address
Even though HLED has been successful with this technique and with keeping their data in synchronization across the multiple stores, they are experiencing some growing pains. Each application instance is creating its own connections to the data stores and a WebSphere MQ queue manager, creating a Web of connections that grows more complex with each application and client instance (see Figure 3-1). In addition, each time a database server has been moved or the data source used to hold some elements has changed, all the applications accessing and combining the disparate data had to change.

Each client has a connection to both datastores and an MQSeries client

Figure 3-1  Current datastore connections
HLED has several goals for information integration:

- To make accessing related information, like the complete investments portfolio, simpler for their applications.
- To isolate changes made to the data sources from the applications as much as possible, so that as they do more fully integrate their data the application impact can be minimized.
- To reduce their ever growing Web of connections to multiple data sources.

HLED wants to formulate a solution that will help them address their multiple data sources and multiple connection problems while helping them to evolve towards a fully integrated information model. They know that trying to replace their current information model is impractical in the near term.

### 3.2 Solution overview

DB2's federated server offers a highly optimized access method allowing the federation (combination) of data from a growing number of both relational and non-relational data sources. In our scenario, we will demonstrate how the DB2 federated server and the DB2 integration with WebSphere MQ addresses several of HLED’s major concerns:

- DB2 federated server provides a gateway to a variety of data sources, including both relational and non-relational stores. Using the federated server to access the different source of data will reduce the “connection Web” problem HLED is currently experiencing.
- DB2 federated server will own the data source locations and definitions, isolating the applications from changes.
- MQSeries Integration provides a gateway to WebSphere MQ functions, reducing the need for an additional WebSphere MQ connection from the applications — further reducing the “connection Web”.
- Using the DB2 federated server also allows us to create views, functions, and stored procedures, giving the applications access to a consolidated image of the business information instead of raw application data. This can reduce the application effort to just one call to the federated server. It also, and perhaps more importantly, changes the psychological perspective of the portfolio information. It is no longer three separate data sources that must be manipulated by the programs, it is now integrated information.
3.3 Infrastructure and technology solution

The new data store connections architecture will look as shown in Figure 3-2.

As was discussed, this reduces the number of external connections an application that needs a complete portfolio must make. As HLED moves forward in their information integration processes, the separate data sources may become a common source. Because the information integration foundation has been put into place by the federated server and MQSeries Integration, the applications are already retrieving information, not combining data sources, and should not have to change.

We will demonstrate two alternatives for implementation:

1. Have the applications continue to make the data store calls, using the federated server as the single connection and data retrieval source. The calls are limited to an DB2 **MQSend** to initiate the CICS/VSAM retrieval and a select to retrieve the investment information from both stores and the CICS/VSAM reply as a consolidated table. The application code flow is simplified as shown in Example 3-2.
Example 3-2  New data access code flow - alternative 1

Connect to DB2 Federated Server
Send customer information request message via the DB2 MQSEND request
SELECT combined investments and customer information reply from the DB2 federated server.
For each investment returned from the Federated Select:
   Set Display.CustInvSymbol = INVSYM
   Set Display.CustInvQuantity = INVQTY

Set Display.CustomerName = retrieved customer name
Set Display.CustomerAddress = retrieved customer address
Disconnect from the server

2. Create a DB2 UDB stored procedure on the federated server that will allow applications to make one SQL call to return the customer portfolio information, essentially encapsulating the requests above into one. The application code flow is simplified as shown in Example 3-3.

Example 3-3  New data access code flow - alternative 2

Connect to DB2 Federated Server
Call GETPORTFOLIO using CustomerID
For each investment returned from the call:
   Set Display.CustInvSymbol = INVSYM
   Set Display.CustInvQuantity = INVQTY

Set Display.CustomerName = retrieved customer name
Set Display.CustomerAddress = retrieved customer address
Disconnect from the server

3.4  Step-by-step implementation

This section details the features and product components that were configured to implement the solution. The steps may look daunting, but once completed the basic infrastructure for information integration is in place. In many environments only a subset of these steps will be needed, as some of the infrastructure is already in place.

Our implementation steps are covered in detail below, and summarized in Table 3-1. Each step has the staff role responsible for the implementation in the “Who” column:

- DB2 UDB DBA for the DB2 UDB Database Administrator
- Informix DBA for the Informix Database Administrator
- MQ Admin for the WebSphere MQ Administrator
Table 3-1  Basic information integration implementation steps

<table>
<thead>
<tr>
<th>Step</th>
<th>Who</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DB2 DBA</td>
<td>Implement the DB2 federated server</td>
</tr>
<tr>
<td>2</td>
<td>DB2 DBA</td>
<td>Create federated database</td>
</tr>
<tr>
<td>3</td>
<td>DB2 DBA</td>
<td>Set up DB2 UDB data source</td>
</tr>
<tr>
<td>4</td>
<td>Informix DBA, DB2 DBA</td>
<td>Set up Informix data source</td>
</tr>
<tr>
<td>5</td>
<td>DB2 DBA</td>
<td>Create the federated view</td>
</tr>
<tr>
<td>6</td>
<td>MQ Admin</td>
<td>Install and configure WebSphere MQ on COPPER</td>
</tr>
<tr>
<td>7</td>
<td>MQ Admin</td>
<td>Install and configure MQSeries AMI</td>
</tr>
<tr>
<td>8</td>
<td>DB2 DBA</td>
<td>Add MQSeries Integration to the database</td>
</tr>
<tr>
<td>9</td>
<td>DB2 DBA</td>
<td>Test MQSeries Integration</td>
</tr>
<tr>
<td>10</td>
<td>MQ Admin, CICS System Programmer</td>
<td>Install and configure z/OS WebSphere MQ and CICS resources</td>
</tr>
<tr>
<td>11</td>
<td>DB2 DBA</td>
<td>Develop SQL statements to access the information</td>
</tr>
<tr>
<td>12</td>
<td>DB2 DBA</td>
<td>Develop the GETPORTFOLIO stored procedure</td>
</tr>
<tr>
<td>13</td>
<td>DB2 DBA, or Application Programmer</td>
<td>Verify the test results</td>
</tr>
</tbody>
</table>

**Note:** The steps, for the most part, should be performed in the order listed.

3.4.1  Implement the DB2 federated server

Our initial implementation step was to install DB2 UDB V8.1 on our Windows 2000 server, COPPER. The installation process is documented in the *IBM DB2 Universal Database Quick Beginnings for DB2 Servers*, GC09-4836.

Complete documentation on the planning and implementation of the DB2 federated server is found in the *DB2 Universal Database Federated Systems Guide*, GC27-1224. We only describe the steps used to set up our existing DB2 UDB server, COPPER, as a federated server for our scenarios.
3.4.2 Create federated database

Our first step is to create the database instance on our previously installed DB2 federated server. The command used to do this is shown in Example 3-4. Note that the command used to create the database will vary by environment.

Example 3-4 Create the federated database

```sql
CREATE DATABASE IIREDBOK AT NODE 0 ON C: ALIAS IIRedbok USING CODESET IBM-1252 TERRITORY US COLLATE USING SYSTEM
```

3.4.3 Set up DB2 UDB data source

The DB2 UDB remote data source to federate is located on a DB2 UDB V7.2 instance on another Windows 2000 platform. There are several steps in implementing the connection, as shown in Table 3-2.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gather DB2 UDB information</td>
</tr>
<tr>
<td>2</td>
<td>Catalog DB2 node and database</td>
</tr>
<tr>
<td>3</td>
<td>Create the federated objects in DB2 UDB</td>
</tr>
</tbody>
</table>

Step 1: Gather DB2 UDB instance information

The first step in creating the connection between our DB2 federated server and our DB2 UDB V7 server is to get the database specific information from that server, as shown in Table 3-3.

<table>
<thead>
<tr>
<th>Host Name</th>
<th>A78RPLPW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port Name</td>
<td>50000</td>
</tr>
</tbody>
</table>

Step 2: Catalog the remote node and database

To access a remote DB2 UDB data source, we have to catalog the remote source node into the federated database as in Example 3-5.

Example 3-5 Catalog remote DB2 UDB source node

```sql
CONNECT TO IIREDBOK;
CATALOG TCPIP NODE DB2V72 REMOTE A78RPLPW SERVER 50000;
```
We have also to catalog the remote DB2 UDB database source into the federated database as in Example 3-6.

Example 3-6  Catalog remote DB2 UDB database

CONNECT TO IIREDBOK
catalog database iirbodbv7 as iirbodb72 at node db2v72 authentication server

Note: You may need to restart your database instance before these changes take effect.

Step 3: Create the federated objects in DB2 UDB
To create the federated objects in our DB2 UDB V8 federated database to access the remote data source DB2 UDB V7, we launched four DB2 commands on the federated server as shown in Example 3-7:

1. The first creates the wrapper, or tells DB2 UDB the name of the program (DLL) to use when making calls to the remote data source.
2. The second defines the DB2 UDB V7 server to the federated database and associates the wrapper with any requests made to that data store.
3. The third creates the user mapping between the two servers.
4. Finally the fourth creates the nickname to tell DB2 UDB how the DB2 UDB V7 data store will be referenced in SQL calls.

Example 3-7  Create federated object for remote DB2 UDB source

CONNECT TO IIREDBOK

CREATE WRAPPER IIRDWMAPPER library 'db2drda.dll'

CREATE SERVER DB2V72SVR TYPE DB2/NT VERSION 7.2 WRAPPER IIRDWMAPPER AUTHORIZATION "db2admin" PASSWORD "db2admin" OPTIONS (NODE 'DB2V72',DBNAME'IIRBDB72')

CREATE USER MAPPING FOR db2admin SERVER DB2V72SVR OPTIONS (REMOTE_AUTHID'db2admin',REMOTE_PASSWORD'db2admin')

CREATE NICKNAME MUFUNDP FOR DB2V72SVR.CBARAR4.MUTUALFUNDPORTFOLIO

Note: The federated objects (wrapper, server, user mapping, nickname) can be created using DB2 Control Center by selecting Instances->Databases->IIREDBOK->Federated Database Objects.
3.4.4 Set up Informix data source

The steps to implement the Informix access are documented in the *DB2 Universal Database Federated Systems Guide*, GC27-1224, previously mentioned. For our implementation we performed the six steps outlined in Table 3-4.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gather Informix information</td>
</tr>
<tr>
<td>2</td>
<td>Install the Informix client</td>
</tr>
<tr>
<td>3</td>
<td>Update environment variables</td>
</tr>
<tr>
<td>4</td>
<td>Define the SQLHOSTS</td>
</tr>
<tr>
<td>5</td>
<td>Add the service name</td>
</tr>
<tr>
<td>6</td>
<td>Create the federated object in DB2 UDB</td>
</tr>
</tbody>
</table>

**Table 3-4  Informix implementation steps**

**Step 1: Gather Informix information**

Just as connecting to DB2 UDB required us to gather information about where the instance is running, we have to do the same thing for Informix. In our scenario we found the information shown in Table 3-5.

<table>
<thead>
<tr>
<th>Server Name</th>
<th>ifmx92t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host name</td>
<td>a23bk51y</td>
</tr>
<tr>
<td>Protocol</td>
<td>olsoc/tcp</td>
</tr>
<tr>
<td>Service</td>
<td>turbo</td>
</tr>
<tr>
<td>Port</td>
<td>1526</td>
</tr>
</tbody>
</table>

**Table 3-5  Informix server information**

**Step 2: Install the Informix client**

The Informix Client SDK must be installed on the DB2 federated server. For complete instructions, please see the Informix documentation: *Informix Client Products Installation Guide*. The Informix product documentation is online at:


When the Informix Client SDK installation is complete, you should have the information for the INFORMIXDIR environment variable.
Step 3: Update environment variables
The environment variables listed in Table 3-6 are required for this configuration. In our environment these were manually entered on the COPPER server:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFORMIXDIR</td>
<td>C:\Program Files\Informix\Client-SDK\</td>
</tr>
<tr>
<td>INFORMIXSERVER</td>
<td>ifmx92nt</td>
</tr>
</tbody>
</table>

Step 4: Define the SQLHOSTS
From Windows, click Start -> Programs -> Informix Client SDK -> Setnet32.

Click the “Server Information” tab, and enter the information from the Informix server as shown in Figure 3-3.

![Setnet32 entries](image)

At this point we found it a good idea to verify that you have the correct information in the Windows registry. To do this, from a command prompt, enter `regedit`. Navigate down the following path, HKEY_LOCAL_MACHINE -> SOFTWARE -> Informix -> SQLHOSTS.
Our entry looked as shown in Figure 3-4.

![Registry Editor example](image)

**Figure 3-4  Registry Editor example**

**Step 5: Add the service name**

The TCP/IP service name (turbo) must be defined to the local server. Using Notepad, open the services file, which can usually be found in: C:\WINNT\system32\drivers\etc.

The entry for turbo should be added in the appropriate placed based on the sequence of services and ports defined. In our example, the entry was placed as shown in Example 3-8.

**Example 3-8  Services file extract**

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wins</td>
<td>1512/tcp</td>
<td>tcp</td>
<td>#Microstate Windows Internet Name Service</td>
</tr>
<tr>
<td>wins</td>
<td>1512/udp</td>
<td>udp</td>
<td>#Microsoft Windows Internet Name Service</td>
</tr>
<tr>
<td>ingreslock</td>
<td>1524/tcp</td>
<td>tcp</td>
<td>ingres</td>
</tr>
<tr>
<td>turbo</td>
<td>1526/tcp</td>
<td>tcp</td>
<td></td>
</tr>
<tr>
<td>l2tp</td>
<td>1701/udp</td>
<td>udp</td>
<td>#Layer Two Tunneling Protocol</td>
</tr>
<tr>
<td>pptp</td>
<td>1723/tcp</td>
<td>tcp</td>
<td>#Point-to-point tunnelling protocol</td>
</tr>
</tbody>
</table>

**Note:** You may need to reboot the server before all the changes made in the previous steps take effect.
Step 6: Create the federated objects in DB2 UDB

To create the federated objects in the DB2 federated database to access the remote data source Informix we launched four DB2 UDB commands on the federated server as shown in Example 3-9:

1. The first creates the wrapper, or tells DB2 UDB the name of the program (DLL) to use when making calls to data source.
2. The second defines the Informix server to the DB2 UDB database and associates the wrapper with any requests made to that data source.
3. The third creates the user mapping between the userids of the two servers.
4. Finally, the fourth creates the nickname to tell DB2 how the Informix data source will be referenced in SQL calls.

Example 3-9   Create federated object to Informix remote source

create wrapper INFORMIX
create server INFMX_SRC type informix version 9.3 wrapper INFORMIX options (node 'ifmx92nt', dbname 'stores_demo', fold_id 'N', fold_pw 'N')
create user mapping for cbarar5 server INFMX_SRC options(remote_authid 'db2drs4', remote_password 'db2drs4')
create nickname INFMX_SRC.INFX_STOCKS for INFMX_SRC."DB2DRS4"."Stock_Portfolio"

Note: You may need to restart your federated database instance before these changes take effect. Also, note that the most common problem we have seen is a “Wrapper Not Found” message on the “create nickname” command; this is normally due to the Informix environment variables or the path not being correct.

Note: The federated objects (wrapper, server, user mapping, nickname) can be created using DB2 Control Center by selecting Instances->Databases->IIREDBOK->Federated Database Objects.

3.4.5 Create the federated view

To facilitate subsequent calls to the two relational data sources, we created a federated view of the data. This helped to keep our subsequent retrieval calls simpler. The view definition is shown in Example 3-10.
Example 3-10  Create investments view

```
CREATE VIEW INVESTMENTS AS SELECT CUST, INVSYM, INVQTY
    from (SELECT CUST_ID AS CUST, STOCK_SYMBOL AS INVSYM,
        TOTAL_SHARES AS INVQTY
        FROM INFMX_SRC.INFX_STOCKS
    UNION
    SELECT CUSTOMERID AS CUST, MUTUAL_FUND_SYMBOL AS INVSYM,
        QUANTITY AS INVQTY
        FROM MUFUNDP) AS
UNIONTBL
```

3.4.6 Install and configure WebSphere MQ

There were five steps involved in setting up WebSphere MQ on COPPER machine for this scenario. Note that this assumes the WebSphere MQ has already been put in place on z/OS.

If that is required in your environment, please see the Program Directory for WebSphere MQ for z/OS, GI10-2548 and WebSphere MQ for z/OS System Setup Guide, SC34-6052.

The steps to implement WebSphere MQ are outlined in Table 3-7.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Install WebSphere MQ on COPPER</td>
</tr>
<tr>
<td>2</td>
<td>Create Queue Manager QM_copper</td>
</tr>
<tr>
<td>3</td>
<td>Define WebSphere MQ objects on QM_copper</td>
</tr>
<tr>
<td>4</td>
<td>Define WebSphere MQ objects on MQV1</td>
</tr>
<tr>
<td>5</td>
<td>Test the communication channels</td>
</tr>
</tbody>
</table>

Note that steps 2-5 are covered in detail in Appendix B, “WebSphere MQ implementation and object definitions” on page 293.

**Step 1: Install WebSphere MQ on COPPER**

For this scenario we installed WebSphere MQ V 5.3 on our COPPER server. Complete documentation on installing WebSphere MQ on the Windows platform is found in the WebSphere MQ for Windows Quick Beginnings Version 5.3, GC34-6073.
Step 2: Create queue manager QM_copper

Once WebSphere MQ is installed on COPPER, we created the queue manager QM_copper. This step is described in B.1, “Create the queue manager QM_copper” on page 294.

Step 3: Define WebSphere MQ objects on QM_copper

The list of definitions required for the COPPER queue manager is given in Table 3-8.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Resource Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>QREMOTE</td>
<td>CUSTOMER.INFO.REQUEST</td>
</tr>
<tr>
<td>QLOCAL</td>
<td>CUSTOMER.INFO.REPLY</td>
</tr>
<tr>
<td>QLOCAL</td>
<td>BackOut</td>
</tr>
<tr>
<td>QLOCAL</td>
<td>BackOut</td>
</tr>
<tr>
<td>QLOCAL (XMITQ)</td>
<td>COPPER.TO.MQV1</td>
</tr>
<tr>
<td>CHANNEL SENDER</td>
<td>COPPER.TO.MQV1</td>
</tr>
<tr>
<td>CHANNEL RECEIVER</td>
<td>MQV1.TO.COPPER</td>
</tr>
</tbody>
</table>

The creation of the WebSphere MQ objects begins with B.2, “Define WebSphere MQ objects on QM_copper” on page 297.

Step 4: Define WebSphere MQ objects on MQV1

A list of the definitions required on MQV1, the z/OS queue manager, is given in Table 3-9.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Resource Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>QLOCAL</td>
<td>CUSTOMER.INFO.REQUEST</td>
</tr>
<tr>
<td>QREMOTE</td>
<td>CUSTOMER.INFO.REPLY</td>
</tr>
<tr>
<td>QREMOTE</td>
<td>COPPER.TEST</td>
</tr>
<tr>
<td>CHANNEL (SENDER)</td>
<td>MQV1.TO.COPPER</td>
</tr>
<tr>
<td>CHANNEL (RECEIVER)</td>
<td>COPPER.TO.MQV1</td>
</tr>
<tr>
<td>PROCESS</td>
<td>GCIN.PROCESS</td>
</tr>
</tbody>
</table>
A complete list of the objects, including the required parameters, is given in “Define WebSphere MQ objects on MQV1” on page 307.

**Note:** Please note that we did not create the channels so they would begin sending messages to the companion platform automatically (triggered channels) for testing purposes. It may be more convenient to have them set that way in your environment.

### Step 5: Test the communication or channels

Testing the channels, or communication, between the MQSeries queue manager on COPPER and the z/OS queue manager, MQV1, includes the following steps:

1. Start the sender channel on QM_copper, COPPER.TO.MQV1.
2. Start the sender channel on MQV1.
3. Use the MQSeries API Exerciser to put a test message on MQV1.TEST
4. Verify message on COPPER.TEST.

Complete instructions on doing this can be found beginning with “Test the communication channels” on page 308.

### 3.4.7 Install and configure MQSeries AMI

To use MQSeries Integration features the MQSeries Application Messaging Interface (AMI) feature must be implemented. AMI provides an MQ interface wrapper designed to allow application access to WebSphere MQ resources, without needing to understand and code all available functions and options available via the MQI. The application and application programmer, in this case DB2 UDB and the DB2 DBA, just need to know which services and policies controlling the resources are available to it, not specific WebSphere MQ details.

AMI implementation can be broken down into two steps:

- AMI installation
- AMI resource definitions.

For more information about both steps please see the manual, *WebSphere MQ Application Messaging Interface*, SC34-6065.
Step 1: AMI installation
Many existing WebSphere MQ customers do not have the AMI component fully installed or implemented in their environment. For AMI installation instructions, please refer to the following sources:

- AMI for distributed platforms is delivered as MQSeries SupportPac MA0F, and is available at the WebSphere MQ Web page:
- AMI for z/OS is available as part of WebSphere MQSeries V 5.2 and above.
- AMI is also delivered as part of the DB2 UDB install and may be found in the `<program files directory>\IBM\SQLLIB\CFG\mq directory`

Step 2: Define AMI resources
Once AMI is installed; two services, IIR.BC1A.RECEIVER and IIR.BC1A.SENDER, and one policy, IIR.TEST.POLICY, were defined by the WebSphere MQ administrator for this scenario. We will go through each AMI definition by panel and point out where changes have been made to the defaults provided.

To initiate AMI, go to START -> Programs -> MQSeries AMI -> IBM MQSeries AMI Administration Tool. You should see the panel shown in Figure 3-5.
Getting Started on Integrating Your Information

If you do not see a file name following the panel title, click File->Open and select the AMT.XML file. This is used as the source AMI file for the DB2 integration.

Note: It is possible that during the process of implementing MQSeries Integration support and installing AMI multiple copies of the AMI control file, AMT.XML have been created. Please check the AMT_DATA_PATH environment variable to make sure you are updating the copy of AMT.XML pointed to by that variable.

Define AMI sender service
To create the service points required, right-click Service Points folder and select Create ->Service Point. You will see the Create a New Service Point panel, as shown in Figure 3-6.
To create our sender service point, we entered the following:

- **IIR.BC1A.SENDER** in the name field
- **CUSTOMER.INFO.REQUEST** in the Queue Name field
- **QM_copper** in the Queue Manager field
- **MQSTR** in the Default Format field.

**Note:** The names are case sensitive. Also note that at the time this document was written the DB2 development tool rolls all MQ names to upper case.

Specifying the default format as MQSTR is very important for the sender service point, because the message data is being built in an ASCII character set but may be read by an EBCDIC process. We are taking advantage of the WebSphere MQ ability to convert messages that contain all character information from one code page to another. Specifying the format as MQSTR tells WebSphere MQ that the entire message is in character data (string) format and is eligible for conversion if required and requested by the application that retrieves the message.
If messages are being created that contain binary or packed decimal data, special conversion routines or data formatting tools must be used to convert messages.

We also created a second sender service point, called IIR.TEST.SEND with the following characteristics:

- Queue Name = COPPER.TEST
- Queue Manager Name = QM_copper
- Default format = MQSTR

**Define AML receiver service**

To create our receiver service point (see Figure 3-7) we again right-click the **Service Points** folder and select **Create ->Service Point**.

![Figure 3-7 IIR.BC1A.RECEIVER](image)

To create our receiver service point, we entered the following:

- IIR.BC1A.RECEIVER in the Name field
- CUSTOMER.INFO.REPLY in the Queue Name Field
- QM_copper in the Queue Manager Name field.
Note that for the MQRECEIVE service the format does not have to be set, the sending application will define the message format.

**Define AMI policy**

Finally, to create our policy we right-click the **Policies** folder and select **Create ->Policy**. The first panel should look as shown in Figure 3-8.

![Create a new Policy](image)

**Figure 3-8  Initialization**

On the **Initialization** tab (Figure 3-8), we entered the following fields:

- IIR.TEST.POLICY in Name
- QM_copper in Connection Name, which defines the queue manager we are connecting to.

The next information required is defined on the **Send** tab.
In Figure 3-9, the only field that was changed was the **Response Correl ID button**. This defaults to Message ID, when using AMI the only opportunity we have to relate the relational data stores’ information to the MQ message is by using the customer ID as an identifier in the correlation ID of both the request and reply messages.

After setting this Response Correl ID button, the final information required is defined on the Receive tab.
Figure 3-10   Receive Tab

On the **Receive Tab** (Figure 3-10) it is very important that the **Convert** selection is checked, this controls whether MQ will convert the message automatically. While this is the default, please verify that it is checked.

Once the policy definition is complete, click the **Finish** button to save it. This should return you to the AMI Administration Tool panel.

Now you may save the AMT.XML file, by clicking the diskette icon on the toolbar.

### 3.4.8 Add MQSeries Integration to the database

To add MQSeries Integration and MQXML support for the database:

- Open a command prompt screen
- Enter the enable MQ and MQXML commands:
  - `enable_MQFunctions -n IIREDBOK -u db2admin -p db2admin`
  - `enable_MQXMLFunctions -n IIREDBOK -u db2admin -p db2admin`

In this scenario we will not be using the MQXML functions, but we implemented them at the same time we did the basic MQ functions.
3.4.9 Test MQSeries Integration

A complete description of the testing used to verify the MQSeries Integration may be found in “Test MQSeries integration” on page 281. The summary of the steps is in Table 3-10.

Table 3-10  Test MQSeries Integration

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Connect to the database</td>
</tr>
<tr>
<td>2</td>
<td>Test MQSend function</td>
</tr>
<tr>
<td>3</td>
<td>Verify MQSend results</td>
</tr>
<tr>
<td>4</td>
<td>Test MQReceive function</td>
</tr>
<tr>
<td>5</td>
<td>Verify MQReceive function</td>
</tr>
</tbody>
</table>

3.4.10 Install and configure z/OS WebSphere MQ and CICS resources

We went through several steps to set up the z/OS CICS environment for this scenario. They are summarized in Table 3-11. The VSAM definitions, test data, CICS program source (COBOL), and the CICS definitions used are given in “z/OS VSAM and CICS definitions and samples” on page 311.

Table 3-11  CICS implementation steps

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Define and load the VSAM file</td>
</tr>
<tr>
<td>2</td>
<td>Add MQ support to the CICS region</td>
</tr>
<tr>
<td>3</td>
<td>Define CICS resources</td>
</tr>
</tbody>
</table>

Step 1: Define and load the VSAM file

As mentioned above sample IDCAMS jobs to define and load the customer information file, CUSTINFO is in “CUSTINFO Definition” on page 328.

Step 2: Add MQ support to the CICS region

The steps to implement MQ in an existing CICS region are documented in the WebSphere MQ for z/OS System Setup Guide, SC34-6052. They should be performed by the CICS or WebSphere MQ administrator.
They are quite simple and include:

1. Updating the CICS System Definitions with the MQSeries group, CSQCAT1
2. Define the MQSeries connection in the CICS System Initialization Table
3. Add some MQSeries libraries to the CICS STEPLIB
4. Add some MQSeries libraries to the CICS RPL
5. Recycle the CICS region

### Step 3: Defining the redbook CICS resources

Three resources were defined to the CICS region for this scenario, all as part of the group IIREDBOK. Complete definitions are in the “CICS definitions” on page 328.

- File - CUSTINFO
- Program - READCINF
- Transaction GCIN

### 3.4.11 Develop SQL statements to access the information

After the setup work is completed, you can now access the multiple data sources through the DB2 federated database.

To access the information two SQL statements are required, one to initiate the CICS transaction to return the VSAM customer information data (`MQSEND`), the second to federate the stock, mutual fund, and MQ reply message data. As you can see in Example 3-13 on page 103 we have used the second SQL statement to federate the relational data sources and to parse the MQ message into the columns needed to create the complete customer portfolio.

#### The MQSend function

The `MQSend` function is used to initiate an application request or send information to receiving applications. `Mqsend` functions that are used as a request usually include 4 parameters, as Example 3-13 on page 103 shows. The first two parameters are the sender service point, defining the queue and queue manager, and test policy we defined in “Install and configure MQSeries AMI” on page 92.

The third parameter is the message itself. For the `Mqsend` function this is defined as a VARCHAR(4000) area, limiting the size of an MQ message processed by DB2 to that length (MQSeries message can be as long as 100Mb). For this scenario our CICS COBOL application program is expecting a simple two field format, the request and the customer ID, as shown in Example 3-11.
The request structure is so simple we have concatenated the request along with the customer ID we are searching for directly into the MQsend function. As that is not usually the case, we have built the message outside the MQsend in the stored procedure example (see Example 3-14 on page 104).

The fourth parameter is the MQSeries correlation identifier, more commonly called the correlid. This field is held by WebSphere MQ itself, in an area known as the MQ message descriptor (MQMD) and not part of the application message body. It is used to provide the logical connection between the request for information and its associated reply. The MQSend supplies this value, and the MQReceive will only return a message whose correlid matches the parameter.

If we did not supply the correlid field on both the MQSend and the MQReceive, the MQReceive would bring back the first message on the queue. In a busy environment, with many requests/replies flowing to and from applications, the request at the top of the queue may not contain the information for the right customer for this application instance.

The correlid is a 24 byte area that is treated as binary information. It is NOT converted across platforms. It is the responsibility of the application processing the request to set the reply correlid field properly.

For more information on the correlid please see the WebSphere MQ Application Programming Guide, SC34-6064 and the WebSphere MQ Application Programming Reference, SC34-6062.

The MQReceive function

The MQReceive retrieves a message from an MQSeries queue, treating it as a table read of one column defined as VARCHAR(4000). Like the MQSend function, the first two parameters are the service point, in this case the receiver service point, and test policy we defined in “Install and configure MQSeries AMI” on page 92. The third parameter is the correlid, which we discussed in the MQSend function.

As expected, the VARCHAR column must be divided into useful fields. we used the SQL substring function to divide the message into the required fields. The COBOL copybook defining the reply message is shown in Example 3-12.
Example 3-12  Customer Information reply copybook

01  CUST-ADDR-REQ-REPLY.
    *                                  
    05  CUST-ADDR-ACTION               PIC X(20)  
        VALUE 'GET CUST INFO          '.
    05  CUST-ADDR-RESPONSE-CD          PIC 9(04) VALUE ZERO.
    05  CUST-ADDR-ERROR-MSG            PIC X(60) VALUE SPACES.
    05  CUST-KEY                        PIC X(10) VALUE SPACES.
    05  CUST-NAME.
        10  CUST-FIRST-NAME             PIC X(30) VALUE SPACES.
        10  CUST-MI                    PIC X(01) VALUE SPACES.
        10  CUST-LAST-NAME             PIC X(30) VALUE SPACES.
    05  CUST-ADDR.
        10  CUST-ADDR-LINE1            PIC X(40) VALUE SPACES.
        10  CUST-ADDR-LINE2            PIC X(40) VALUE SPACES.
        10  CUST-ADDR-LINE3            PIC X(40) VALUE SPACES.
        10  CUST-ADDR-CITY             PIC X(40) VALUE SPACES.
        10  CUST-ADDR-ST-PROV          PIC X(20) VALUE SPACES.
        10  CUST-ADDR-CODE             PIC X(10) VALUE SPACES.

Complete details of the MQsend and MQReceive functions can be found in the IBM DB2 Universal Database SQL Reference Volume 1, SC09-4844.

SQL statements example

The two SQL statements for MQsend are provided in Example 3-13.

The second SQL statement federates the relational data sources and parses the MQ message into the columns needed to create the complete customer portfolio.

Example 3-13  SQL statements example

VALUES db2mq.MQSEND('IIR.BCIA.SENDER','IIR.TEST.POLICY','GET CUST INFO    ' ||
  '1001      ','1001      ')
SELECT T.CUST, T.INVSYM, T.INVQTY,
    B.FIRST AS FIRSTNAME,
    B.MI AS MIINIT,
    B.LAST AS LASTNAME,
    B.ADRLN1,
    B.ADRLN2,
    B.ADRLN3,
    B.CITY AS ADRCITY,
    B.STATE AS ADRSTATE,
    B.ZIPCODE AS ADRCODE,
    B.ERR_MSG AS ERRMSG
FROM (SELECT * FROM INVESTMENTS A WHERE A.CUST = '1001      ') AS T,
      (SELECT substr(msg,1,20) as REQ,
               substr(msg,21,4) as RETCODE,
Getting Started on Integrating Your Information

3.4.12 Develop the GETPORTFOLIO stored procedure

As an alternative, to reduce the amount of code in the client applications, we created an SQL stored procedure that contains both the SQL requests needed to return the complete portfolio. The applications using this procedure will have to issue a single call to the GETPORTFOLIO procedure to return the integrated portfolio information.

For those of you unfamiliar with creating and using stored procedures, please refer to the DB2 UDB manual Application Development Guide: Building and Running Applications, SC09-4825. Please pay particular attention to the development environment set-up steps.

The stored procedure we used in this example is shown in Example 3-14.

Example 3-14 GETPORTFOLIO stored procedure

```sql
CREATE PROCEDURE GETPORTFOLIO (IN CUSTID CHAR(10)) RESULT SETS 1 LANGUAGE SQL BEGIN
    DECLARE SendService CHAR(25);
    DECLARE AMIPolicy CHAR(25);
    DECLARE SVCRequest CHAR(30);
    DECLARE REQ CHAR(20);
    DECLARE C1 CURSOR WITH RETURN FOR
    SELECT T.CUST, T.INVSYM, T.INVQTY,
        B.FIRST AS FIRSTNAME, B.MI AS MIINIT,
        B.LAST AS LASTNAME, B.ADRLN1, B.ADRLN2, B.ADRLN3,
        B.CITY AS ADRCITY, B.STATE AS ADRSTATE,
    from
        table(db2mq.mqRECEIVEall('IIR.BC1A.RECEIVER','IIR.TEST.POLICY','1001 ',1)) AS FROM_QUEUE) AS B
```
B.ZIPCODE AS ADRCODE, B.ERR_MSG AS ERRMSG
FROM (SELECT * FROM CBARAR5.INVESTMENTS WHERE CUST = CUSTID) AS T,
(SELECT substr(msg,1,20) as REQ,
 substr(msg,21,4) as RETCODE,
 substr(msg,25,60) as ERR_MSG,
 substr(msg,85,10)  as CUST,
 substr(msg,95,30) as FIRST,
 substr(msg,125,1) as MI,
 substr(msg,126,30) as LAST,
 substr(msg,156,40) as ADRLN1,
 substr(msg,196,40) as ADRLN2,
 substr(msg,236,40) as ADRLN3,
 substr(msg,276,40) as CITY,
 substr(msg,316,20) as STATE,
 substr(msg,336,10) as ZIPCODE
 from table(db2mq.mqRECEIVEall('IIR.BC1A.RECEIVER',
 'IIR.TEST.POLICY',CUSTID,1)) AS FROM_QUEUE) AS B;
DECLARE MQ1 CURSOR FOR
VALUES db2mq.MQSEND('IIR.BC1A.SENDER','IIR.TEST.POLICY',
 SVCRequest,CUSTID);
SET REQ = 'GET CUST INFO       ';
SET SVCRequest = REQ || CUSTID;
OPEN MQ1;
CLOSE MQ1;
OPEN C1;
END @

To load the stored procedure into our database we used the following command
from the DB2 UDB command line:

db2 -td@ -vf C:\GETPORTFOLIO.txt

Note: The ‘-td@’ gives the end of file delimiter and, if not entered, each line of
the procedure will be treated as separate input. The ‘-v’ indicates verbose
output. The ‘f’ denotes file input.

Testing the stored procedure was quite simple. From a DB2 UDB command line
processor window we entered the request shown in Example 3-15.

Example 3-15   GETPORTFOLIO request

CALL GETPORTFOLIO('1001      ')
3.4.13 Verify the test results

There are multiple ways to test the scenario, we used the DB2 UDB command line processor to test the individual SQL calls, as shown in Example 3-13 on page 103, and the stored procedure call, as shown in Example 3-15 on page 105. Either of the options may be used to embed the requests into application code.

The results were the same for entering the SQL requests directed into the command line and invoking the stored procedure as shown in Example 3-16.

Example 3-16 Test the request

<table>
<thead>
<tr>
<th>CUST</th>
<th>INVSYM</th>
<th>INVQTY</th>
<th>FIRSTNAME</th>
<th>MIINIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>QRS1</td>
<td>25.000</td>
<td>FRED</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>FGFND</td>
<td>50.150</td>
<td>FRED</td>
<td>M</td>
</tr>
<tr>
<td>1001</td>
<td>RVG@</td>
<td>922.000</td>
<td>FRED</td>
<td>M</td>
</tr>
<tr>
<td>1001</td>
<td>IBM</td>
<td>5231.000</td>
<td>FRED</td>
<td>M</td>
</tr>
<tr>
<td>LASTNAME</td>
<td>ADRLN1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LO</td>
<td>123 BAY STREET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LO</td>
<td>123 BAY STREET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LO</td>
<td>123 BAY STREET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LO</td>
<td>123 BAY STREET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADRLN2</td>
<td>ADRLN3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADRCITY</td>
<td>ADRSTATE</td>
<td>ADRCODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VANCOUVER</td>
<td>BC CANADA</td>
<td>V7E2R9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VANCOUVER</td>
<td>BC CANADA</td>
<td>V7E2R9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VANCOUVER</td>
<td>BC CANADA</td>
<td>V7E2R9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VANCOUVER</td>
<td>BC CANADA</td>
<td>V7E2R9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.4.14 Alternatives

In this somewhat artificial scenario the data was fairly homogenous and there was not any data redundancy. This is, of course, the exception rather than the rule, as this scenario demonstrated the federated server capability and DB2 UDB's MQSeries integration.

In many instances, the data may require some level of transformation before an integrated image can be presented, or certain types of information may be requested. In addition, there are many times when a customer would like to include a level of data synchronization, application processing, or automated process flow into the scenario, to assist with the long term goal of information integration.
For these operations the WebSphere MQ Integrator (WMQI) can be used to supplement the DB2 federated server’s ability to combine data from multiple stores into a single image. If, for example, our simple example were changed slightly to include the following:

- XML documents will be used for all service requests and replies.
- Data transformation, in particular the key used by the client applications does not map exactly to the CICS/VSAM data source. For the purposes of this scenario it could be as simple as right justifying and zero filling the client key to the match the format expected.
- Customer information is kept on both the CICS/VSAM file and on the DB2 UDB MutualFundPortfolio table. The service should determine which data store has the most current information, present that copy to the requestor, automatically update the older version of the data, and send a notification that the action has taken place.

The application code flow would be like the example shown in Example 3-17.

**Example 3-17  Code flow to invoke WMQI**

1. Connect to the DB2 federated database
2. Create XML request message ‘Get Complete Portfolio’ via the SQL function MQSENDXML
3. Retrieve the reply via the SQL function MQRECEIVEXML
4. Disconnect from the data base.

The DTD for the request might look as shown in Example 3-18.

**Example 3-18  Complete portfolio request DTD**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % CustomerRequestType "(ServiceRequest,CustomerNumber)">
<!ELEMENT ServiceRequest (#PCDATA)>
<!ELEMENT CustomerNumber (#PCDATA)>
<!ELEMENT CustomerRequest %CustomerRequestType;>
```

The SQL statements to build and send the request might look as shown in Example 3-19.

**Example 3-19  MQSENDXML sample**

```sql
SET MSG = '<CustomerRequestType>' || CustomerNumber || '</CustomerRequestType>'
```

Chapter 3. Business case I: Federating data access using SQL 107
values db2xml.mqsendxml('DB2.DEFAULT.SERVICE', 'DB2.DEFAULT.POLICY', MSG,CustomerNumber);

We would expect two WMQI flows to be developed, one to initiate the customer information request from the CICS/VSAM file, the second to process the reply and make the federated access SQL request to the investments view.

The first flow, which we illustrate in Figure 3-11, we call Process Portfolio Request Flow. We would anticipate it performing the following services:

- In the node labeled PortfolioRequestcopy, copy the client request to another queue to be used in aggregation.
- In the node labeled CreateCustInfoRequest:
  - Construct the VSAM key from the customer ID passed.
  - Format the Customer Information request message
- In the node labeled CustomerRequests, put the formatted message on the CUSTOMER.REQUESTS queue to initiate the CICS/VSAM transaction.

Figure 3-11 Process Portfolio request flow
The second flow, called Process_Portfolio_Reply_1, which we illustrate in Figure 3-12, performs the following services:

- In the node labeled CUSTOMER.REPLY the response from the CICS/VSAM transaction is retrieved.
- In the node labeled PORTFOLIO.REQUEST.CPY the copy of the original request is retrieved.
- In the node labeled PORTFOLIO.AGG.REPLY the two retrieved messages are combined into a single reply.
- In the node labeled GET FEDERATED DATA and CICS REPLY:
  - Issue the SQL SELECT request to the INVESTMENTS view to retrieve the investment and the customer information held on the MutualFundPortfolio data source.
  - Build the investment portion of the XML reply
  - Determine which copy of the customer information is ‘newer’ and build the customer information portion of the reply from the more recent update.
  - If the CICS/VSAM data is older:
    - Format and send “Update Customer Information” request message to CICS using the newer information and last update date.
    - In the node labeled CUSTOMER.INFO.UPDATE, put the update request.
    - Format the audit trail message.
    - In the node labeled Customer Info Audit, MQPUT an audit trail message.
  - If the MutualFundPortfolio data is older:
    - Update, via an SQL call to the federated server, the customer information.
    - Format the audit trail message.
    - In the node labeled Customer Info Audit, MQPUT an audit trail message.
Using the federated database simplifies the WMQI flows because each compute node is limited to connecting to one database. If a federated database is not available, each relational source needed is accessed from individual nodes, adding complexity to the flows. If there is a federated database that can be accessed, then all data stores known to that database are available to the node.

3.5 Further steps

This scenario can be enhanced using DB2 Information Integrator to expand its federated access to heterogeneous relational databases as Oracle but also access to unstructured data as XML files.

In this section we provide an example of how DB2 Information Integrator could be used to access the Stock Portfolio data on an Oracle source instead of INFORMIX source.

To get more information on information integration, please refer to the following Web site:


3.5.1 DB2 Information Integrator

The DB2 Information Integrator product provides additional wrappers enabling access to non-DB2 and non-relational sources. DB2 UDB provides wrappers only to DB2 UDB and Informix.

DB2 Information Integrator can be installed in two ways. It can be installed as a standalone product since it contains the technology to provide data federation, replication and transformation. Where a server already has DB2 UDB installed the DB2 Information Integrator install behaves like a feature of DB2 UDB. DB2 Information Integrator uses the installed server components of DB2 UDB and installs only the components which adds access to heterogeneous data sources. The discussion below assumes the second style of installation.

3.5.2 Federated access to Oracle

We assume that:

- The federated server is already set up as described in “Create federated database” on page 84.
- DB2 Information Integrator is set up on top of DB2 UDB V8.1.

Also, we only describe the steps that will be required to access Oracle server.
3.5.3 Set up Oracle data source

For Oracle implementation, the steps to perform will be the ones outlined in Table 3-4.

Table 3-12 Informix implementation steps

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gather Oracle information</td>
</tr>
<tr>
<td>2</td>
<td>Install the Oracle client</td>
</tr>
<tr>
<td>3</td>
<td>Update the client configuration file:tnsnames.ora</td>
</tr>
<tr>
<td>4</td>
<td>Update federated configuration variables (on AIX only)</td>
</tr>
<tr>
<td>5</td>
<td>Create the federated object in DB2 UDB</td>
</tr>
</tbody>
</table>

Step 1: Gather Oracle information

Just as connecting to Informix required us to gather information on the Informix server, we have to do the same thing for Oracle (see Table 3-5).

Table 3-13 Oracle server information

<table>
<thead>
<tr>
<th>Oracle Server Name (SID) or Oracle Server Global Name</th>
<th>IIDEMO2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>9.30.76.61</td>
</tr>
<tr>
<td>Protocol</td>
<td>TCP</td>
</tr>
<tr>
<td>Service name</td>
<td>iidemo2.stl.ibm.com</td>
</tr>
<tr>
<td>Port</td>
<td>1521</td>
</tr>
<tr>
<td>Oracle Server userid</td>
<td>demo</td>
</tr>
<tr>
<td>Oracle Server password</td>
<td>cdidemo</td>
</tr>
</tbody>
</table>

Some information on the Oracle Server system (except for userid and password) can be found out in the listener.ora file located on the Oracle server side for example in:

- Windows: C:\oracle\ora81\network\ADMIN\listener.ora
- AIX: $ORACLE_HOME/network/admin/listener.ora

Step 2: Install the Oracle client

To access Oracle V8 data sources, we set up Net8 client software. For the prerequisites on Oracle server and client versions required by DB2 federated server, please check the Federated Systems Guide, GC27-1224.
Oracle client setup can be done using the Oracle Universal Installer program.

**Step 3: Update the client configuration file: tnsnames.ora**

On the Oracle client side, tnsnames.ora should be updated to match the Oracle server parameters.

For example, this file is located on:

- Windows: C:\oracle\ora81\network\ADMIN\tnsnames.ora
- AIX: $ORACLE_HOME/network/admin/tnsnames.ora

**Example 3-20  Oracle client tnsnames.ora file**

```
# TNSNAMES.ORA Network Configuration File:
c:\oracle\ora81\network\admin\tnsnames.ora
# Generated by Oracle configuration tools.

INST1_HTTP =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 9.30.76.61)(PORT = 1521))
    )
  )
  (CONNECT_DATA =
    (SERVER = SHARE)
    (SERVICE_NAME = iidemo2.stl.ibm.com)
    (PRESENTATION = http://admin)
  )

IIDEMO2 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 9.30.76.61)(PORT = 1521))
    )
  )
  (CONNECT_DATA =
    (SERVICE_NAME = iidemo2.stl.ibm.com)
  )
```

Connectivity to Oracle server from the client side can be tested independently using the SQLPlus interactive program from Oracle on the client (Example 3-21).

**Example 3-21  Using SQLPlus interactive tool**

- Test connection using:
  ```
  c:\>SQLPLUS demo/cdidemo@iidemo2
  ```
For example, the table on Oracle was created using Oracle SQLPlus as shown in Figure 3-13.

![Figure 3-13 SQL Plus](image)

Note: You may need to reboot the server before all the changes made in the previous steps take effect.

**Step 4: Update environment variables**

On Windows, DB2 federated server does not use the db2dj.ini file located under sqllib\cfg to get environment variables and their values to work with the non-DB2 client software.

On Windows, DB2 federated server uses the System Environment Variables (Start-->Settings-->System) to read the variables/values to access the non-DB2 client software.

On UNIX, DB2 federated server does use the db2dj.ini file to set up ORACLE_HOME variable.

You may also have to check the PATH variables to be sure that ORACLE libraries (for example oracle\ora81\bin) are set up.

**Step 5: Create the federated objects in DB2 UDB**

To create the federated objects in our DB2 UDB V8 federated database to access the remote Oracle table, we launched four DB2 UDB commands on the federated server, as shown in Example 3-22:

1. The first creates the *wrapper* (NET8), or tells DB2 UDB the name of the program (DLL) to use when making calls to data source.
2. The second *defines the Oracle server* to the DB2 UDB database and associates the wrapper with any requests made to that data source.
3. The third creates the *user mapping* between the userids of the two servers.
4. Finally, the fourth creates the nickname to tell DB2 UDB how the Oracle data source will be referenced in SQL calls.

Example 3-22  Create federated object to Oracle data source

```sql
create wrapper net8;

create server ORASRVR type oracle version 8 wrapper net8 authorization "demo" password "cdidemo" options (node 'iidemo2');

create user mapping  for user server ORASRVR options (REMOTE_AUTHID 'demo', REMOTE_PASSWORD 'cdidemo');

create nickname ora_stocks for ORASRVR.demo.stocks;
```

**Note:** When testing the connection to Oracle, make sure using SQLPlus tool that you can access the data on the Oracle server. Problems encountered on the `create nickname` command are often linked to a communication problem; then check the parameters in tnsnames.ora and the path used.

The nickname previously defined can be used by the federated server to access data on the Oracle server as shown in Figure 3-14.

```sql
L:\redbook\files>db2 select * from ora_stocks

<table>
<thead>
<tr>
<th>CUS1_ID</th>
<th>STOCK_SYMBOL</th>
<th>TOTAL_SHARES</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000001</td>
<td>IBM</td>
<td>1,0000000000000000E+002</td>
</tr>
<tr>
<td>000000002</td>
<td>MSFT</td>
<td>5,0000000000000000E+001</td>
</tr>
<tr>
<td>000000003</td>
<td>ORA</td>
<td>2,5000000000000000E+001</td>
</tr>
</tbody>
</table>

3 record(s) selected.

L:\redbook\files>
```

**Figure 3-14  Read the Oracle table through the federated server**

**Note:** The federated objects (wrapper, server, user mapping, nickname) can be created using DB2 Control Center by selecting Instances->Databases->IIREDBOK->Federated Database Objects.
3.6 Integration considerations

Integrating your information may require beyond the implementation to take in account considerations such as:

- Skills required
- Availability
- Maintenance
- Performance and capacity planning
- Security

3.6.1 Staffing requirements

We recommend the following staffing model for this type of information integration project:

1. Project Manager: This person should be someone with a fairly “global” view of the enterprise data and the corporate direction for information integration. While the Project Manager may not need to know the details of each application, data store, and tool being used, they do need to know the capabilities of each.

2. Application/Information experts: These are the core of any integration project, they know the information and how it is used. Integration projects, even integration at very low data level, should include representatives of the applications or data creators and consumers.

3. DB2 DBA: This person will be responsible for defining the federated server, the views and resources used.

4. WebSphere MQ Administrator on both distributed and z/OS platforms: This person (or these people) will be responsible for defining the WebSphere MQ objects used in the information integration.

5. Other DBAs: In our example we used Informix as a federated remote data source, so the Informix DBA was included.

6. CICS System Programmer: This person will help to add both MQ support to the CICS region and the resources required for this scenario.
3.6.2 Availability

Service, data, and application availability have become increasingly important as more businesses have a global presence, even if that presence is only via the Internet. Availability planning should include the following topics:

- Planned outages for hardware and software upgrades.
- Failover - unplanned outages and disaster recovery, this should include both hardware and software failover.
- Data archiving and recovery.
- Scalability - how easy is it to add a new server or copy of data?
- Application and data redundancy - often having redundant copies of databases and applications is not even a decision, they have to be there to support a seamless failover solution. Data replication and synchronization processes must be included in planning.

3.6.3 Maintenance

Maintenance of distributed systems has traditionally been a challenge, especially during data store or application upgrades. Integrating data and applications, often makes that task less of an issue. Simply, efforts to integrate data such as those outlined in business case 1, can be used decouple the clients making requests from changes made to the data stores. These changes can include anything from server location or address, platform, to data formats. When changes are needed, they are made at the federated server level, not in client code or client connections.

3.6.4 Performance and capacity planning

Performance and capacity planning is a perpetual process. When planning an integration project it is important to accurately predict capacity and volume so that performance expectations can be set and met. Once implemented, monitoring and tuning is often of even more importance.

Capacity planning for integration contains multiple components that need to be considered. For scenario 1 each execution of the GETPORTFOLIO stored procedure generates several obvious actions:

- Call to DB2 UDB for Mutual Fund data
- Call Informix for Stock data
- Call to MQSeries to send the CICS/VSAM request
- CICS program to access VSAM file
- Call to WebSphere MQ to retrieve the CICS/VSAM reply
Note that these are not all the actions generated, just the most obvious ones. In our business case these same actions are currently taking place from each client and will be consolidated by using the federated server, which could in fact mean a lowering of overall resource utilization on HLEDs systems. An information integration project needs to include the potential capacity and performance impact, whether an increase or a decrease, on the complete environment.

Here is another point to keep in mind: using the DB2 UDB V8 federated server has a very highly optimized access engine. This can substantially reduce access costs to data stores and should be factored into the capacity planning.

Like capacity planning, ongoing monitoring and tuning for integrated applications often requires gathering information from multiple systems and combining it into a total picture. If one component of an integration solution is reaching it’s capacity, it impacts the whole solution. While this is nothing new, it does need to be emphasized as it is often overlooked.

### 3.6.5 Security

Security is always an implementation and integration challenge. Some of the questions that should be considered are:

- What kinds of application security are in place or planned?
- How do the various components of an integration secure their resources?
- How are user IDs passed between the components? Are there restrictions and user ID “translations” that must take place between the systems and platforms?

A good example of user ID usage is demonstrated in our testing of business case I. As is customary in a development and test environment, we have very little security turned on. The user ID making the request for the portfolio information is the user ID connected to the database, a Windows user ID. When the WebSphere MQ message initiates the CICS transaction, the user ID used is the user ID that initiated the CICS region, a user ID with a very high level of authorization.

Many projects find it very useful to map out user ID usage on each of the systems and applications involved.

- How “deep” should security go? There are rare times, especially when the data is not sensitive, when you get to the level of information integration, that little security checking is implemented, and users requesting services have been authenticated and authorized, well before getting to this level.
3.7 Conclusion

As we said earlier, implementing this environment can appear quite complex. However, once the infrastructure is in place, changes to the data sources and often to the environment impact fewer applications and normally require much less coordination. For example, as information integration projects complete, data sources may change location or type while the information being used by the majority of the applications remains the same.

In this scenario we demonstrated that by using the federated server, with the WebSphere MQ integration, we were able to create a “GETPORTFOLIO” procedure and SQL calls. This effectively consolidates the three connections and types of calls made by applications currently, simplifying both the information and the connection code that applications must manage.
Chapter 4. Business case II: Accessing your information through Web Services

In this chapter we show how you can expose information stored or federated within DB2 through the Web Services interface. We explain how DB2 UDB data/information can be dynamically transformed to XML and demonstrate the role played by DB2 in the Web Services world through an example. This chapter focuses on externalizing DB2 UDB data, but with the use of DB2 Information Integrator, data from non-DB2 sources can also be made available through a Web Services interface.

We utilize tools such as IBM WebSphere Studio to help build our sample Web Services, and introduce you to the Web Services Runtime Framework (WORF). Through WORF, Web Services of these operation types are supported:

- XML-based query or storage — in other words, an XML document is stored in DB2 UDB relational tables and composed again on retrieval. This method of operation requires the presence of DB2 XML Extender.
- SQL-based operation, such as calling stored procedures, or inserting, updating, and deleting DB2 UDB data.
Both XML-based and SQL-based operations are controlled by a file called Document Access Definition eXtension (DADX). The DADX defines the operations that can be performed by the Web Services. In simplified terms, access to DB2 information can be defined using an XML file (yes, DADX files uses XML format to describe the intended operations). We will examine the construct of a DADX file in this chapter.

Also, we introduce you to the scenario that will demonstrate DB2 Web Services. We build the scenario using tools found on the Java and the IBM WebSphere platform. Lastly, we build the same scenario using the IBM WebSphere Studio, in our case, we use IBM WebSphere Studio Application Developer.
4.1 Business problem statement

Almaden Credit Union is using HLED Investments to service its clients if the clients want to carry out any investment type of transaction; that is, buying or selling stocks and/or mutual funds. Almaden Credit Union currently has the ability for its clients to check their balances, view a list of their recent transactions, and do some self-service banking. Since the investment accounts and services are managed and stored in a system external to the Credit Union, the Credit Union does not provide any online service on the investment accounts.

Clients who have investment accounts with Almaden Credit Union demand to see their investment portfolios, their recent investment transactions, and they want to be able to invoke a buy or sell stocks/mutual funds order online. Almaden Credit Union needs to access the information managed and stored at HLED Investments. Almaden Credit Union is wondering how to achieve the integration between its system and the system at HLED Investments.

Essentially, HLED Investments has to make available their investment portfolio data to Almaden Credit Union and to other potential business partners. When contemplating a possible solution, HLED Investments has come up with the following requirements:

1. The system must provide portfolio information to Almaden Credit Union and potentially, any future customers who wish to obtain portfolio information. The system has to be extensible, scalable and reusable.

2. The system must support platform independence; integrate systems regardless of their implementations; thus, unifying disparate computing paradigms and linking diverse underlying technologies.

3. The system must be built upon open standards, protocol, and network transport.

4. The system must reduce the coupling between business functions and integration; hence, reduce the inter- and intra-enterprise development dependencies.

5. The system must allow customers to submit requests (see their portfolio, list their transactions, buy or sell stocks and mutual funds, and so on) to HLED Investments and get intermediate feedback from the system.

6. The system must provide a way for HLED to externalize their business processes and integrate with its business partners.
4.2 Assumptions

HLED Investments stores the account information in DB2 UDB, Informix, and VSAM files. See Chapter 4, “Business case II: Accessing your information through Web Services” on page 119 for a description on how the datasource as well as the data are being set up. You must complete the scenario outline in the same chapter before you can work on this scenario.

You have been introduced to the Web Services concepts and technologies (see “Why Web Services?” on page 44). To learn more on Web Services, refer to:


There are many ways to derive Web Services, for example, from Java Beans, Enterprise Java Beans (EJBs), as well as some Web Services toolkits. Here, we will concentrate on building Web Services through DB2 Web Services.

4.3 Solution overview

Our solution has to demonstrate the following characteristics to meet the requirements layout by HLED Investment:

► Supports distributed computing
► Supports platform independence; integrate systems regardless of their implementation
► Can be described, published, discovered, and invoked dynamically
► Can be built on open standards, protocol, and network transport
► Reduce coupling between business functions and process models
► Supports a well-understood programming model for connecting businesses via the Internet
► Reduces time and cost during systems integration

HLED Investments decides to use the DB2 federated server to extract the information from the distributed and disparate data sources and expose the same information through DB2 Web Services. Almaden Credit Union can request the information through a simple Web service request and the information will come back in XML format as a Web service reply. Once Almaden Credit Union gets the information through the Web service reply, it can choose to work with the information further or simply present the information out to its clients.
A review of HLED Investments’ data and data stores is given here. We are going to include some new information to complete our scenario:

1. HLED Investments stores its stock portfolio information in an INFORMIX database. The customer profile information; however, is found in VSAM files hosted on a Z series.

2. HLED Investment acquired MutualMax Investments to strengthen its position in the mutual fund market. MutualMax Investments stores its mutual fund portfolio information in a DB2 UDB database. Both customer profile (optional) and mutual fund portfolio information are stored in the same database.

3. The market price, stock, and mutual fund information are coming from some external sources. The information are delivered to HLED Investments in form of XML files.

4. Additional stock trading information is stored on a DB2 UDB database.

5. HLED Investments has to consolidate all of this information and deliver it to Alamaden Credit Union through Web Services interface.

Figure 4-1 illustrates both the logical as well as the physical location of all the data. Figure 4-2 shows the database schemas of the data with the respective data stores we are going to work with in this scenario.

![Figure 4-1: Physical location of data](image)
With our scenario, we federate the stock_portfolio located in the Informix database to the DB2 federated server. We use the MQSeries Integration to access the customer profile information in VSAM. The mutual_fund_portfolio and the customer profile information (optional) are stored in a DB2 UDB v7.2 database. We can either use federation to access data stored in the different versions of DB2 UDB. In our case, we federate the two tables from DB2 UDB v7.2 into DB2 UDB v8.1. We use the XML extender to decompose the stock and mutual fund information from XML files into a XML collection tables in the federated server. Lastly, we make the data available to our clients through Web Services interface.

The scenario has the following flow:
1. Import stock and mutual fund information into a DB2 UDB. This will enable customers to view their market values of their complete portfolio. This is done through decomposing XML files into DB2 XML collections.
2. View customers’ stock and mutual fund combined portfolio with customers’ profile information and with the associated stock or mutual fund information.
3. View a customer’s recent transactions.
4. Enable customers to do transactions as buy/sell stocks and mutual fund through the Web Services interface.
4.4 Infrastructure and technology solution

Before we start to build DB2 Web Services, we have to introduce the technologies and tools available in DB2 UDB that enable us to easily build Web Services. First, we introduce the Web Services Object Runtime Framework (WORF). Second, we present an overview of the Document Access Definition Extension (DADX) which is a document where we specify DB2 Web Services.

Web Services Object Runtime Framework

The Web Services Object Runtime Framework (WORF) provides an environment to create simple XML based Web Services that access DB2 UDB information. WORF uses Apache SOAP 2.2 or later and the Document Access Definition Extension (DADX). A DADX document specifies a Web Service using a set of operations that are defined by SQL statements or XML Extender Document Access Definition (DAD) documents. An overview of DADX is given in “DADX overview and structure” on page 127 and the format of a DADX file is provided as additional material (see Appendix E, “Web Services XML and DADX files” on page 343). For more information on DAD, please refer to Chapter 5, “Business case III: Incorporating XML data” on page 193.

Web Services, or functions invoked over the Internet, specified in a DADX file are called DADX Web Services, also referred to as DB2 Web Services.

WORF features

WORF provides the following features:

- Resource-based deployment and invocations, or using DADX and, optionally, other resources that help define the Web Service.
- Automatic service redeployment during development time. Changing resource definitions is reflected immediately without any server restart.
- HTTP GET and POST bindings, in addition binding to SOAP.
- Automatic WSDL and XSD generation, including support for UDDI Best Practices
- Automatic documentation and test page generation

WORF supports resource-based deployment of Web Services. Resource-based deployment means that you define your Web Services in a resource file that you place in a directory of your Web application.

When you request that resource file, WORF loads it and makes it available as a Web Services. In cases you modify the definitions (operations) in the resource file and you request the same resource again, WORF detects the changes and loads the modified version automatically. Automatic reloading makes DB2 Web Services development efficient and productive.
WORF generates a Web Services test client as a Web application, using Java servlet and JSPs. The test client can use simple HTTP or SOAP bindings. An HTTP binding is useful for testing a DB2 Web Service directly using a Web browser. The SOAP binding can be used by Web Services clients to create distributed applications.

WSDL is a general purpose XML language for describing the interface, protocol bindings and the deployment details of network services. WSDL complements the UDDI standard by providing a uniform way of describing the abstract interface and protocol bindings of arbitrary network services. UDDI best practices is an attempt to clarify the relationship between the two (WSDL and UDDI) and describe how WSDL can be used to help create UDDI business service descriptions. See Figure 4-3.

![Figure 4-3  WORF architecture](image)

This diagram shows how WORF uses the different components when processes a Web Service request: WORF receives an HTTP, SOAP, GET, or POST service request. The URL of the Web Service request includes the name of the Web Service’s resource file and a command. The command is either a built-in command or an operation of Web Service specified in the resource file. The built-in commands are: TEST, WSDL, and XSD (XML schema file). Upon receiving the command, WORF generates an HTML test page for TEST, a WSDL document for WSDL, or an XML schema file for XSD.
In case the command is an operation, WORF invokes the specified operation of the Web Service and returns the result document. The following describes the steps WORF performs to handle a Web Service request:

1. Loads the DADX file specified in the request.
2. Generates a response, based on the request:
   a. For operations:
      • Loads the associated DAD file, if required.
      • Replaces query parameters with requested values.
      • Connects to DB2 UDB and runs any SQL statements, including SQL calls.
      • Commits the database transaction.
      • Formats the result into XML, converting types as necessary.
   b. For commands:
      • Generates necessary files, test pages, or other responses required.
3. Returns the response to the service requestor.

**WORF supported environment**

WORF is available on Windows NT, Windows 2000, AIX, Solaris and Linux. WORF is available from the DB2 XML Extender Web site, or with DB2 UDB Version 8 and WebSphere Studio Version 4 and Version 5. When delivered with WebSphere Studio, WORF is supported with a set of tools that automate the building of DADX Web Services. These tools include a wizard to create DADX files based on SQL statements or DAD files, and tools to create DAD files. WORF also work with Informix.

**DADX overview and structure**

A DADX file specifies how to create a Web Service using a set of operations that are defined by SQL statements (including stored procedure calls) and, optionally DAD files if your Web services store and retrieve XML documents managed by DB2 XML Extender. WORF provides the run-time support for invoking DADX files as Web Services in Apache SOAP 2.2 (or later) engine supported by IBM WebSphere Application Server and Apache Jakarta Tomcat.

WORF allows you to specify storage and retrieval operations on XML data, it also allows you to expose stored procedures as well as SQL statements as invokable Web Service operations.

You can expose any database stored procedure as long as your stored procedures have result sets with fixed metadata (a fixed number and a fixed shape). The operation signature includes the input and output parameters. You can also specify SQL statements to select, insert, update and delete data. Simple mapping of XML schema to SQL data type is provided.
Exposing SQL statements and stored procedures do not require the XML Extender to be installed and active.

DADX files support two kinds of Web Services operations:

- **XML collection operations (requires DB2 XML Extender):** These are storage and retrieval operations that help you to map XML document structures to DB2 UDB tables so that you can either compose XML documents from existing DB2 data, or decompose (store untagged element or attribute content) XML documents into DB2 data. This method is useful for data documents are frequently updated.

  There are two elements that make up the XML collection operations:

  - `<retrieveXML>`: Generates XML documents
  - `<storeXML>`: Stores XML documents

  The DAD file provides the mapping of XML documents to a DB2 UDB database for both storage and retrieval operations.

- **SQL operations:** SQL-based querying is the ability to execute SQL statements, including stored procedure calls, to DB2 UDB and to return results with a default tagging. The data is returned using only a simple mapping of SQL data types, using column names as elements. There are three elements that make up the SQL operations:

  - `<query>`: Queries the database
  - `<update>`: Inserts into, deletes from, or updates a database
  - `<call>`: Calls stored procedures with multiple result sets

  The syntax of DADX file is presented in Appendix D, “DADX syntax” on page 335. A copy of the XML schemas of DADX (DADX.xsd) is shipped with WORF. The file is located inside the `<worf_unzipped_dir>/schemas` directory.

**DADX examples**

Here we provide a couple of examples of DADX files: Example 4-1 illustrates an XML collection operation, while Example 4-2 is an example of specifying SQL operation in a DADX file.

**Example 4-1  DADX using XML collection operation**

```xml
<xml version="1.0" encoding="UTF-8"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    Simple DADX example that access the Department table in the SAMPLE DB
  </wsdl:documentation>
</DADX>
```
Chapter 4. Business case II: Accessing your information through Web Services

Example 4-2  DADX using SQL operation

```xml
<?xml version="1.0" encoding="UTF-8"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    Simple DADX example that access the Department table in the SAMPLE DB
  </wsdl:documentation>
  <opeartion name="listDepartments">
    <wsdl:documenation>
      List all departments from the table
    </wsdl:documentation>
    <retrieveXML>
      <DAD_REF>departments.dad</DAD_REF>
      <no_override/>
    </retrieveXML>
  </opeartion>
</DADX>
```

4.5 Step-by-step: Building DB2 Web Services

In this section, we present two alternatives for creating your DB2 Web Services:

1. Use WORF and your favorite text editor to construct your DB2 Web Services.
2. Use IBM WebSphere Studio Application Developer to build DB2 Web Services.

You will find that these methods complement each other. For example, using WORF is fast and you can get your DB2 Web Service up and running in no time. However, packaging and deploying your Web Service as a standalone Web application may be too tedious of a task, so that we want to use the tool to aid us in packaging our Web application.
### 4.5.1 Build Web Services using WORF

Here we describe the first alternative for building Web Services, by using WORF. Table 4-1 summarizes the steps you must go through to build DB2 Web Services. We assume that all the steps here can be carried out by an application developer with DB2 UDB and SQL skills to write the DADX file.

#### Table 4-1 DB2 Web Services step-by-step implementation summary

<table>
<thead>
<tr>
<th>Step</th>
<th>Objective</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Setting up necessary hardware and software: prerequisites</td>
<td>Install and configure software and start working on creating DB2 Web Services.</td>
</tr>
<tr>
<td>1</td>
<td>Installing WORF</td>
<td>WORF is the enabling technology to allow DB2 UDB users to build Web Services by simply defining the service through SQL or XML operations.</td>
</tr>
<tr>
<td>2</td>
<td>Creating and populating the data sources</td>
<td>In order for our scenario to be demonstrable, we have to create the necessary data definitions, and populate the data sources with data for illustration purpose.</td>
</tr>
<tr>
<td>3</td>
<td>Creating a XML collection and a DAD file</td>
<td>In order to store XML data in DB2 UDB, we need the mapping definition between the XML file and a relational table. A DAD file provides the mapping. The XML collection is the relational table.</td>
</tr>
<tr>
<td>4</td>
<td>Creating a new DADX Group</td>
<td>A DADX group helps to organize all the resources for a Web Service</td>
</tr>
<tr>
<td>5</td>
<td>Creating a DADX file</td>
<td>Defining the Web Service with its methods using DADX file</td>
</tr>
<tr>
<td>6</td>
<td>Testing the Web Service</td>
<td>Ensure the Web Service and its methods are carrying out the correct operations.</td>
</tr>
<tr>
<td>7</td>
<td>Adding additional functions to an existing Web Service</td>
<td>One DADX file can support multiple functions. We add the rest of the functions to our DB2 Web Services</td>
</tr>
<tr>
<td>8</td>
<td>Packaging the Web Service</td>
<td>Package the Web Service in the way that is specified by J2EE.</td>
</tr>
<tr>
<td>9</td>
<td>Deploying the Web Service</td>
<td>Deploy our packaged Web Service and make it available to be consumed by clients.</td>
</tr>
</tbody>
</table>
We now start to build our Web services scenario, in the following order:

1. Update the DB2 XML Collection in DB2 UDB using an XML file.
2. List a customer’s portfolio including market values.
3. List a customer’s recent transactions.
4. Initiate a buy or sell order.

**Prerequisites**

You must have installed the following software products before you start to build your DB2 Web Services:

1. Microsoft Windows NT, version 4.0 with Service Pack 4 or higher, or Microsoft Windows 2000 Professional Service Pack 2 or higher.
2. Informix Dynamic Server 2000 version 9.x. Verify the installation by creating the demo store database. This will serve as one of our federated data stores.
3. DB2 UDB V7.2 and the DB2 XML Extender 7.2 or DB2 UDB V8.1 (XML Extender comes with DB2 UDB V8.1). Verify the installation by creating the DB2 UDB sample database. DB2 UDB V8.1 will serve as your federated server, while DB2 UDB V7.2 will serve as one of our federated data stores. You may consider installing DB2 UDB V7.2 on a separate machine to house some of the DB2 UDB data.
4. Make sure you are using the JDBC 2.0 driver. DB2 UDB V8.1 uses JDBC 2.0 by default. You have to execute `<db2_install_dir>\java12\usejdbc2.bat` to switch the driver from JDBC 1.0 to JDBC 2.0. for DB2 UDB V7.2.

You may have completed steps 2 through 4. These steps define the same data sources found in chapter 3 to set up your federated and MQ access. You can keep using the same databases definitions for this scenario.

5. WebSphere Application Server Advanced Edition 4.0. Verify your installation by requesting the snoop servlet from your Web browser of your choice. Netscape 4.72 or higher or Microsoft Internet Explorer 5.x or higher:

   http://localhost/servlet/snoop/

6. Apache SOAP 2.2 or higher. WebSphere Application Server Advance Edition 4.0 comes with Apache SOAP 2.2. Verify there is a soap.jar in the directory `<WebSphere_install_dir>\lib`. If you do not find the file, you have to download and unzip the file `soap-bin-2.2.zip` and copy the extracted file `soap-2_2\lib\soap.jar` into `<WebSphere_install_dir>\lib` and verify that the samples work. You can use the Web application services (shipped with WORF) to test after you have installed WORF see Step 1. Installing WORF. You can download the zip file from:

   http://xml.apache.org/dist/soap/version-2.2/
Step 1: Installing WORF

The objective of this step is to set up the environment to build DB2 Web Services. This step involves different tasks; you need to complete all the tasks as outlined in Table 4-2.

**Note:** You can use any SOAP release as long as it is SOAP 2.2 or higher. The latest version is Apache SOAP 2.3.1.

7. Java 2 SDK, Standard Edition Version 1.3.0 or higher.
8. WebSphere Studio Application Developer, Version 4.0.3 or higher.

### Table 4-2  Tasks summary for installing WORF

<table>
<thead>
<tr>
<th>Task</th>
<th>Task description</th>
<th>Addition information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Download or locate dxxworf.zip</td>
<td>WORF is distributed by dxxworf.zip. You have to have a copy of this file to install WORF</td>
</tr>
<tr>
<td>2</td>
<td>Copy worf.jar into WebSphere archive directory</td>
<td>Make WORF available to your Web application server.</td>
</tr>
<tr>
<td>3</td>
<td>Install the sample Enterprise Application</td>
<td>This task includes the sub tasks to deploy the sample enterprise application.</td>
</tr>
<tr>
<td>4</td>
<td>Enable the sample enterprise application</td>
<td>Stop and start your Web application server</td>
</tr>
<tr>
<td>5</td>
<td>Test the sample Web Services</td>
<td>You may need to create the necessary database to complete your testing.</td>
</tr>
</tbody>
</table>

The framework is delivered:

- With DB2 UDB V8.1, you can find WORF under the directory `<db2_install_dir>/samples/java/WebSphere`. The name of the package is: dxxworf.zip.


- With IBM WebSphere Studio, which provides a set of tools that make it easier to build DADX Web Services.
Complete these tasks to install WORF with WebSphere Application Server:

1. **Download or find the file dxxworf.zip** depending on which version of DB2 UDB you have installed. Unzip the file to a directory, such as c:\worf. You find the directory contains the following content (after unzip):
   - readme.html
   - lib\ - the jar and war file library
     - worf.jar - the WORF library, which must be installed on the classpath of the servlet engine
     - services.war - a sample Web application containing Web Services that use WORF
     - worf-servlets.jar - servlet implementations for the DB2 Web Services
   - schemas\ - XML schemas files
     - dadx.xsd - XML schemas for the DADX files
     - nst.xsd - XML schemas for the NST XML files
     - wsdl.xsd - XML schemas for the WSDL files
   - tools\lib\ - set of tools to help build DADX and DAD files
     - CheckersCommon.jar - common utilities for DAD and DADX syntax checkers
     - DADChecker.jar - DAD syntax checker
     - DADXEnvChecker.jar - DADX syntax checker

2. **Copy worf.jar to <WebSphere_install_dir>\lib.**

3. **Start WebSphere Admin Server:** This can be done through Windows services or through the Start menu. Once the Admin Server service is started, you can open WebSphere Administrator's console, and install services.war as an enterprise application. You can use the wizard menu to configure your application. You can give the Web application any context name. We use the name services. The wizard also asks you to select a virtual host (such as default_host) and select an application server (such as Default Server). Figure 4-4 shows the WAS Administrator's Console during the installation of the application using the Install Enterprise Application Wizard.
Getting Started on Integrating Your Information

Figure 4-4   Installing the sample services Web application

- Choose Install stand-alone module (*.war, *.jar)
- You can enter in: c:\worf\lib\services.war for the Path information if you have extracted the worf files into the directory c:\worf. If not, you should adjust the path to reflect your configuration.
- For Application name, we choose the name servicesApp.
- For Context root for Web module, we simply put /services
- You click Next eight times. You are asked to select a virtual host for your Web module. You should select default_host as your virtual host. In most cases, this is the default. You click Next after you have selected a virtual host.
- You are asked to select an application server. You should select Default Server (your hostname) as your application server. You click Next and Finish to complete the enterprise application installation process.
4. **Stop (if your application server is currently running) and start your application server** containing the newly installed services Web application. Test the installation by opening your browser on the services Web application welcome page -- which if you have used our example is: http://localhost:9080/services. The page should look like Figure 4-5.

![Web Services Sample Page](image)

**Figure 4-5  Web Services sample page**

5. **Test the sample Web Services**: Click some of the Java Bean Samples TEST links to see if the sample services work. If you want to explore the rest of the Web Services samples, you may elect to create the SALES_DB database. To create the database, run `setup.cmd` found in the directory:

   ```
   <WebSphere_install_dir>\installedApps\servicesApp.ear\services.war\WEB-INF\classes\groups\dxx_sales_db.
   ```
You should be able to use all the TEST links in SALES_DB Database Samples.

Now, you are ready to get started writing your own DB2 Web Services.

**Step 2: Creating and populating the data sources**

Before you can develop our scenario, you must create the necessary data sources and populate the data sources with some data for illustration purposes. Our scenario starts with an Informix database, some VSAM data on an IBM mainframe and a DB2 UDB V7.2 database.

We utilize DB2 XML Extender to help us work with XML files. Traditional SQL data is either decomposed from incoming XML documents or used to compose outgoing XML documents. If our data is to be shared with other applications, we might want to be able to compose and decompose incoming and outgoing XML documents and manage the data as necessary to take advantage of the relational capabilities of DB2 UDB. This type of XML document storage is called XML collection. We will use XML collection to work with our investment information XML files.

Table 4-4 summaries all the tasks and subtasks and in some cases, sub-subtask. Please make sure to complete all the tasks and subtasks in this section. Tasks 1 and 2 have been described in Business case I in 3.4, “Step-by-step implementation” on page 82.

### Table 4-3   Steps to create and populate the data sources

<table>
<thead>
<tr>
<th>Task</th>
<th>Task description</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create database and tables</td>
<td>This includes creating all the federated databases, VSAM files, database tables. Populate the table and file with appropriate data.</td>
</tr>
<tr>
<td>2</td>
<td>Install and setup a federated server</td>
<td>Use DB2 UDB Version 8.1 to serve as a federated server.</td>
</tr>
<tr>
<td>3</td>
<td>Add an additional orders table to the federated server</td>
<td>This table is essential for us to show the functionality on recent transactions</td>
</tr>
</tbody>
</table>

1. **Create all the databases and tables** as specified in Table 4-4. The table below serves to remind you of where the data is stored and how the data is organized. They should already have bee defined as specified in 2.4, “Target environment” on page 68 and used in Chapter 3, “Business case I: Federating data access using SQL” on page 77.
Table 4-4  Stock and mutual fund portfolio data and their data stores

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Database</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informix Dynamic Server 2000 v9.2</td>
<td>demo_store</td>
<td>STOCK_PORTFOLIO</td>
</tr>
<tr>
<td>VSAM</td>
<td>N/A</td>
<td>CUSTOMER_INFO</td>
</tr>
<tr>
<td>DB2 UDB v7.2</td>
<td>ACC</td>
<td>MUTUAL_FUND_PORTFOLIO CUSTOMER_INFO (optional)</td>
</tr>
</tbody>
</table>

2. **Add an additional table to keep track of all the recent transactions** happening on our server.

Table 4-5  Orders table and its data store location

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Database</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 UDB v8.1</td>
<td>IIREDBOK</td>
<td>Orders</td>
</tr>
</tbody>
</table>

The data definition for the Orders table is given in Example 4-3.

**Example 4-3  Orders table data definition**

```sql
CREATE TABLE <schemaname>.ORDERS
    ("ORDER_NUMBER" CHARACTER (10) NOT NULL ,
    "SYMBOL" CHARACTER (5) NOT NULL ,
    "ACTION_INDICATOR" CHARACTER (1) NOT NULL ,
    "TRANSACTION_DATE" DATE ,
    "QUANTITY" DECIMAL (12, 3) ,
    "PRICE" DECIMAL (12, 2) ,
    "CUSTOMER_ID" CHARACTER (10) )
DATA CAPTURE NONE IN USERSPACE1
```

**Step 3: Create an XML collection and a DAD file**

We work with investment information and the information is stored in form of XML in many different files. We want to decompose the information in the XML files and store them in a XML collection. We will create an XML collection by utilizing the tool: XML Extender Administrator on the federated server to hold the information for certain stock and mutual fund specified in XML files.

Table 4-6  Tasks summaries for build XML collection and a DAD file

<table>
<thead>
<tr>
<th>Task</th>
<th>Task description</th>
<th>Addition information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Locate XML Extender Administration tool</td>
<td>Discover where the tool is located on your DB2 UDB installation</td>
</tr>
</tbody>
</table>
1. **Locate the XML Extender Administration** and its files. These files are installed in the following directories as a standard install for DB2 UDB V8.1:

   `<db2_install_dir>\TOOLS\dxxadmin.jar`:
   XML Extender Administration .jar file

   `<db2_install_dir>\bin\dxxadmin.cmd`:
   Used to invoke the XML Extender Administration tool

   `<db2_install_dir>\bin\dxxadmin.exe`:
   XML Extender Administration executable file

   `<db2_install_dir>\db2xml\adm_gui\readme.me`:
   Information on how to install, configure, and run the XML Extender Administration tool

2. **Add the following .jar files to your CLASSPATH environment variable.**

   - `dxxadmin.jar`  `<db2_install_dir>\TOOLS`
   - `xalan.jar`  `<db2_install_dir>\TOOLS`
   - `xerces.jar`  `<db2_install_dir>\TOOLS`
   - `Common.jar`  `<db2_install_dir>\java`

---

<table>
<thead>
<tr>
<th>Task</th>
<th>Task description</th>
<th>Addition information</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Add Java Archive to your CLASSPATH to enable the tool</td>
<td>To run XML Extender Administration tool, you have to include a set of Java Archive to your CLASSPATH</td>
</tr>
<tr>
<td>3</td>
<td>Start the tools and logon</td>
<td>Start the tool and connect the tools to your database, preview the list of tasks available with the tool</td>
</tr>
<tr>
<td>4</td>
<td>Enable database with XML support</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Import a DTD to work with our XML document</td>
<td>This DTD file defines the elements and attributes in your XML file.</td>
</tr>
<tr>
<td>6</td>
<td>Create a DAD file</td>
<td>We create a RDB_node DAD file to map the XML attributes and elements to SQL data types and columns in DB2 UDB. In this task, you use the tool to build you the document tree for you XML collection. It may be a little tedious. Please be patient and work through all the tasks and sub tasks in this step.</td>
</tr>
<tr>
<td>7</td>
<td>Create the XML collection using the DAD file</td>
<td>This is the collection we use to store our XML data.</td>
</tr>
</tbody>
</table>
Your CLASSPATH should include the above Java archives before you can run the XML Extender Administration tool.

3. **Start the XML Extender Administration tool** by entering the command: `dxxadmin`. You see the application logon screen come up and it should be similar to the screenshot we show in Figure 4-6. On this screen, you have to supply the following information:

   - **Address**: This is your Java jdbc datasource name (dsn). The database we are working with has the name of IIREDBOK. Enter the database name after “jdbc:db2:” and in place of “//[host]:[port]/[database]”. Since our database is local to the administration tool, we can safely ignore the host and port when specifying the database. You can use the database of your choice; however, the database has to be your designated federated server.
   
   - **User ID**: Provide your DB2 user id. We used `db2admin` in our case.
   
   - **Password**: Provide the password with your userid. We used `db2admin` in our case. You can use the password that matches your DB2 UDB setup.
   
   - **JDBC Driver**: Leave it as-is; use the default.
   
   - **Click Finish** to complete the Logon.

![XML Extender Administration LaunchPad](image)

*Figure 4-6  XML Extender Administration logon screen*
Once you are logged on, you can choose to:

**Switch Database**
Disconnect from the connected database and connect to another database.

**Enable/Disable Database**
Enable/Disable the database XML support.

**Import a DTD**
Import an existing DTD file into the database DTD repository.

**Edit DAD**
Edit an XML DAD file.

**Work with XML Columns**
View XML Extender column-related tasks.

**Work with XML Collections**
View XML Extender collection-related tasks.

Figure 4-7 shows all the tasks you can work with using the tool.
4. If you have not enabled your database with XML support, you can do so by clicking Enable Database. Or you can do this in a command prompt and issue the following statement:

```
prompt> dxxadm enable_db IIREDB0K
```

5. We import a DTD for our XML collection by selecting Import a DTD. The DTD defines the information that comes with a stock or mutual fund. The stock_mutualFund_info.dtd definition is shown in Example 4-4.

```
Example 4-4  Stock_mutualFund data definition

<?xml version="1.0" encoding="US-ASCII"?>
<!ELEMENT stock_mutualFund_info (symbol, description, type, sector, price, market)>
<!ATTLIST stock_mutualFund_info key CDATA #REQUIRED>
<!ELEMENT symbol (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT sector (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT market (#PCDATA)>
```

**Tips:** In the process of working with this scenario, you will create a number of files. In order to keep track of all the different files, our recommendation is to create the following directories to hold all the different files:

- redbk_samples - a root directory
- redbk_samples\dad - a directory to hold your DAD files
- redbk_samples\dtd - a directory to hold your DTD files
- redbk_samples\java - a directory to hold your Java applications
- redbk_samples\sql - a directory to hold your SQL script files
- redbk_samples\xml - a directory to hold your XML files

Create the file stock_mutualFund_info.dtd in the DTD directory and import the file using the XML Extender Administration. To import the DTD file you just created, click **Import a DTD** from XML Extender Administration: Select a Task. In the “Import a DTD” screen, you fill out the necessary information (see Figure 4-8):

- **DTD file name**: Specify the file name with its path information
- **DTD ID**: Enter a unique name. We use stock_mutualFund_info.
- **Author**: Enter a valid DB2 user id. We use **db2admin**.

You can do this with the a DB2 command as in Example 4-5 to accomplish the task.
Example 4-5  DB2 command to import a DTD

DB2 INSERT into DB2XML.DTD_REF values
("<db2_install_dir>/redbk_samples/db2xml/dtd/stock_mutualFnd_info.dtd",
DB2XML.XMLClobFromFile('stock_mutualFund_info',
0,
'db2admin',
'db2admin',
'db2admin' 
)

Figure 4-8  XML Extender Administration: Import a DTD

Click Finish and the system responds with a confirmation as in Figure 4-9.

Figure 4-9  XML Extender Administration: Import a DTD confirmation
After you have imported the DTD, you are ready to build a DAD file. A DAD file maps the structure of the XML document to the DB2 UDB tables from which you compose and decompose your documents. In our case, we use a DAD file to help us in decomposing XML files into a DB2 XML collection.

6. Create a DAD file:
   - Click Edit DAD from the XML Extender Administrator “Select a Task” screen. You are presented with another window (Figure 4-10):

   ![Work with DAD Files Wizard](image)

   For our scenario, we select XML collection RDB node mapping as our type and we leave the file name field blank to create a new DAD file. Click Next.

   - The next screen, you have to decided whether you want your XML documents to be validated or not. Click Do not validate XML documents with the DTD. Click Next to open the Specify Text Window.

   - We are going to decompose an XML document. We can ignore the Prolog field. But if we are using the DAD file for both composition and decomposition, type the prolog name in the Prolog field:

   ```xml
   ?xml version="1.0"?
   ```
We can ignore the **Doctype** field since we are decomposing an XML document. Otherwise, type the document type of the XML document in the **Doctype** field. For example:

```xml
<!DOCTYPE stock_mutualFund_info SYSTEM "../dtd/stock_mutualFund_info.dtd"
```

**Note**: if you do not put any value in either fields, the **Next** button will stay inactive. Put one or more blanks in both field to activate the **Next** button.

Click **Next** to open the RDB Mapping window:

i. Select the Root icon to add a root node

ii. Click **New Element** to define a new node.

iii. In the **Details** box, specify **Node type** as **Element**.

iv. Enter the name of the top level node in the Node name field. We use `stock_mutualFund_info` as the name of our root element.

v. Click **Add** to create the new node:

You have created the root node or element, which is the parent to all the other elements and attribute nodes in the map. The root node has table child elements and a join condition (mandatory if you use more than one table for your mapping). In our case, we are using only one table, so we can ignore the join condition.

vi. Add table nodes for each table that is part of the collection:

Highlight the root node name and select **New Element**.

In the **Details** box, specify **Node type** as **Text**.

Click **Add** to add a text node.

Highlight the newly added text node and select **New Element**.

In the **Details** box, specify **Node type** as **Table**.

Select or type the name of the table from **Table name**. Your collection table has not been created yet. You can specify the intended table name here. We use `stock_mutualFund_info` as the table name.

Repeat if you have more table, in our case, we have finished specifying our collection table.

vii. Add attribute node:

a). Click a parent node in the field on the left to add a child element or attribute. If you have not selected a parent node, **New** is not selectable.

b). Click **New Element**.

c). Select a node type from the **Node type** menu in the **Details** box. The Node type menu displays only the node types that are valid at this point in the map: **Element** or **Attribute**.
We add an attribute to the root node. Select Attribute and type in a Node name key. Click Add.

d). Select the newly added Attribute node and click New Element:

The Node type menu displays only one valid type: Table at this point. We select Table for the node type and type in our table name which is stock_mutualFund_info. Leave the table key blank.

e). Select the Attribute node: key again and click New Element:

The node type menu displays more options this time: Table, column and condition. We pick column and specify a column name: info_key and the column type as INTEGER.

Step d) and e) are to map the contents of an attribute node to a relational table.

We only have one attribute in our XML document. In case you need to add more attribute, you can come back and repeat the steps to add in your additional attributes.

viii. Add element node:

Follow the same steps described in Add attribute node. Instead of selecting Attribute, you have to select Element. And one major different between attribute and element is you can not add the relational table mapping directly under the Element itself, you have to create a Text node before you can create your mapping for Table and Column. You have to:

a). Specify a text node. Click the parent node and click New Element. In the Node type field, select Text. Select Add to add the node.

b). Add a table node:

Select the text node you just created and click New Element. In the Node type field, select Table and specify our table name. Click Add to add the node.

c). Add a column node:

Select the text node again and click New Element. In the Node type field, select Column and specify a column name: symbol and the column type: CHAR(5).

Click Add to add the node.

Repeat for all the elements in the following table. All elements have the same parent node and that is the root element. Symbol is already done.
Table 4-7 Elements properties for stock_mutualFund_info XML document

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Table name</th>
<th>Column name</th>
<th>Column type</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbol</td>
<td>stock_mutualFund_info</td>
<td>symbol</td>
<td>CHAR(5)</td>
</tr>
<tr>
<td>description</td>
<td>stock_mutualFund_info</td>
<td>description</td>
<td>CHAR(100)</td>
</tr>
<tr>
<td>type</td>
<td>stock_mutualFund_info</td>
<td>type</td>
<td>CHAR(15)</td>
</tr>
<tr>
<td>sector</td>
<td>stock_mutualFund_info</td>
<td>sector</td>
<td>CHAR(30)</td>
</tr>
<tr>
<td>price</td>
<td>stock_mutualFund_info</td>
<td>price</td>
<td>DECIMAL(12,3)</td>
</tr>
<tr>
<td>market</td>
<td>stock_mutualFund_info</td>
<td>market</td>
<td>CHAR(5)</td>
</tr>
</tbody>
</table>

After you have added all the elements and their mappings, you can click Next.

- Now, you have to specify a file name for your DAD file. We type in: `<db2_install_dir>\redbk_samples\dad\stock_mutualFund_info.dad` Click Finish. This will generate and save your DAD file.

  You can verify your DAD file by importing it back to the tool. During import, the tool will validate the syntax of your DAD file.

  Your completed DAD file should be similar to the one shown in Example 4-6. If you have problem importing the DAD file, you can use an editor to edit the DAD file directly.

  **Note:** If the tool has trouble importing your DAD file, you can pay attention to the windows (command prompt) where you start your XML Extender Administration. There you can obtain more diagnostic information of why your file cannot be imported into the tools. Below, we show a screenshot of editing the DAD file in Figure 4-11. The upper left hand window shown the node structure of the DAD file.

  The resulted DAD file is included in Example 4-6 in its entirety.
Figure 4-11  XML Extender Administration: editing DAD

Example 4-6  stock_mutualFund_info.dad

```xml
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "../dtd/dad.dtd">
<DAD>
  <dtdid>stock_mutualFund_info</dtdid>
  <validation>NO</validation>
  <Xcollection>
    <prolog><?xml version="1.0"?></prolog>
    <doctype>!DOCTYPE stock_mutualFund_info SYSTEM "../dtd/stock_mutualFund_info.dtd"</doctype>
    <root_node>
      <element_node name="stock_mutualFund_info">
        <RDB_node>
```
Getting Started on Integrating Your Information
7. Create a db2 collection to store the XML documents:
   – Select Work with XML Collections in the Select a Task screen.
   – Click Enable an XML Collection, the Next button will show up. Click Next.
   – Enter a collection name, your DAD file and table space.
   – Collection name: stock_mutualFund_info
   – DAD file name: browse to find your newly created DAD file
   – Table space: this is optional, you can specify your table space here.
   – Click Finish. You will receive a confirmation on the collection is being enabled. A collection table will be created for you.

Step 4: Create a new DADX group
Before we get started with this step, it is important for you to have the Web Services samples from the WORF installation running on your machine. We are going to build our scenario on top of the WROF sample. If you have not gotten the sample running, please follow the instructions in “Step 1: Installing WORF” on page 132 and make sure you see the “Web Services Sample Page” come up on a Web browser.
The resources for all DADX Web Services groups are stored in the a directory WEB-INF/classes/groups where WEB-INF is the directory used by J2EE Web applications to store resources that are not directly available to HTTP requests. This means that users cannot see the contents of DADX files. DADX files contain the implementation of the Web Services and are therefore similar to Java classes.

Each DADX group contains database connection and other information that is shared between DADX files. This information is stored in a file called group.properties.

Table outlines the steps you have to go through.

<table>
<thead>
<tr>
<th>Task</th>
<th>Task Description</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create group folder</td>
<td>Create a folder to hold all the DADX resources.</td>
</tr>
<tr>
<td>2</td>
<td>Edit group.properties</td>
<td>Configure the properties for your DADX Web Service.</td>
</tr>
<tr>
<td>3</td>
<td>Add a new servlet mapping for you Web service</td>
<td>You modify the web.xml file to add in an additional servlet mapping.</td>
</tr>
</tbody>
</table>

1. **To create the new DADX group**, complete the following steps:
   - Go to WebSphere_install_dir>\installedApps\servicesApp.ear\services.war\WEB-INF\classes\groups and create a new folder named after your new group; for example, stockInfo.
   - Copy the group.properties file from another DADX group folder — for example, WEB-INF\classes\groups\dxx_sample — to your newly created directory, WEB-INF\classes\groups\stockInfo, and open the file in a text editor. Set the property values as in Example 4-7.

**Example 4-7  Set up the property values**

```java
initialContextFactory=
databaseJNDI=
dbDriver=COM.ibm.db2.jdbc.app.DB2Driver
dbURL=jdbc:db2:IIREDDB0K
userID=db2admin
password=db2admin
namespaceTable=namespacetable.nst
autoReload=true
reloadIntervalSeconds=5
groupNamespaceUri=http://schemas.ibm.com/stockInfo
enableXmlClob=true
```
2. Edit the group.properties file

The properties have the meanings described in Table 4-9.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialContextFactory</td>
<td>The Java class name of the JNDI initial context factory that is used to locate the DataSource for the database.</td>
</tr>
<tr>
<td>datasourceJNDI</td>
<td>The JNDI name of the DataSource for the database.</td>
</tr>
<tr>
<td>dbDriver</td>
<td>The Java class name of the JDBC driver for the database.</td>
</tr>
<tr>
<td>dbURL</td>
<td>The JDBC URL of the database.</td>
</tr>
<tr>
<td>userID</td>
<td>The user ID for the database</td>
</tr>
<tr>
<td>password</td>
<td>The password for the user ID for the database.</td>
</tr>
<tr>
<td>namespaceTable</td>
<td>The resource name of the namespace table.</td>
</tr>
<tr>
<td>autoReload</td>
<td>The boolean automatic reloading mode.</td>
</tr>
<tr>
<td>reloadIntervalSeconds</td>
<td>The integer automatic reloading time interval in seconds.</td>
</tr>
<tr>
<td>groupNamesapceUri</td>
<td>The URI prefix for automatically generated namespaces for the WSDL and XSD documents.</td>
</tr>
<tr>
<td>enableXmlClob</td>
<td>The boolean to enable using XMLCLOB for the database.</td>
</tr>
</tbody>
</table>

- `initialContextFactory` and `datasourceJNDI` are optional and define a DataSource for the database connection. Use of a DataSource allows WORF to benefit from connection pooling and distributed transactions.
- `dbDriver` and `dbURL` are optional and define the JDBC access to the database. Either the DataSource or the JDBC connection must be defined. If both are defined, then the DataSource is tried first. If WORF is unable to obtain the DataSource then the JDBC connection is tried.
- `userID` and `password` are required and are used to obtain the connection to the database.
- `namespaceTable` is a reference to a Namespace Table (NST) resource that defines the mapping from DB2 XML Extender DTDID to XSD namespace and locations.
- `autoReload` and `reloadInterval` control resource loading and caching, and are optional. If autoReload is absent or false then resource reloading is not done and reloadInterval is ignored. When autoReload is true, then whenever WORF accesses a resource (as DAD, DADX, DTD, NST), it compares the current time with the time the resource was previously loaded.
If more than reloadInterval seconds have passed, then WORF checks the file system for a newer version and will reload the resource if it has changed.

Automatic reloading is useful at development time, in which case reloadInterval can be set to 0. If used in production, reloadInterval should be set to a large value such as 60 to avoid reducing server performance.

- groupNamespaceUri is optional. If not specified then the default prefix urn:<context>/group is used where <context> is the name of the Web application and <group> is the name of the servlet path for the Web Services group for example urn:/services/stockInfo

3. Add a new servlet mapping

Edit <WebSphere_install_dir>installedApps\servicesApp.ear\services.war\WEB-INF\web.xml to add the new DADX group to the Web application.

Example 4-8     Edit group.properties

...<servlet id="Servlet_4">
    <servlet-name>stockInfo</servlet-name>
    <servlet-class>com.ibm.etools.webservice.rt.dxx.servlet.DxxInvoker</servlet-class>
    <init-param id="InitParam_4">
        <param-name>faultListener</param-name>
        <param-value>org.apache.soap.server.DOMFaultListener</param-value>
    </init-param>
    <load-on-startup>-1</load-on-startup>
</servlet>
...
<servlet-mapping id="ServletMapping_4">
    <servlet-name>stockInfo</servlet-name>
    <url-pattern>/stock/*</url-pattern>
</servlet-mapping>
...

The number 4 in our example can be changed to match the number on your system.

Note: We use the name stock as our servlet mapping name. In order to access our stockInfo Web Service, we have to use the following URL: http://localhost:9080/services/stock/...

You may also encounter a problem with the namespace table resource resolution. Make sure you map the dtdid in your namespaceTable.NST file matches the one in your DAD file. Figure 4-12 highlights the mapping as well as give you an example of the namespaceTable.nst file:
Step 5: Creating the DADX files

We will create a DADX file. The DADX file can have multiple operations. We will be creating operations for:

- Decomposing XML into a DB2 collection
- Viewing a customer’s stock and mutual fund portfolio
- Viewing a customer’s recent trading transactions
- Making a trade (updating the database)

In this step, you will complete one of these operations. We will complete the entire Web Service in a later step. Essentially, Table 4-10 shows the different operations.

**Table 4-10 Creating the DADX file outline**

<table>
<thead>
<tr>
<th>Task</th>
<th>Task description</th>
<th>Addition information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Clone from an existing DADX file</td>
<td>Copy an existing DADX file and use it as your base</td>
</tr>
<tr>
<td>2</td>
<td>Add an operation</td>
<td>Start editing your DADX file and adding an operation</td>
</tr>
<tr>
<td>3</td>
<td>Add in documentation for your operation</td>
<td>Work with &lt;documentation&gt; tag.</td>
</tr>
</tbody>
</table>
1. **Copy an existing DADX file**, for example `dxx_sample\list.dadx`, and rename it to `stockInfo.dadx`.

2. Open your DADX file with your favorite editor, **rename one of the operations** to `storeStockMutualFundInfo` and delete all other operations.

3. **Change the documentation** to reflect both the DADX file and the operation: for the DADX file, you can replace the documentation with, for example **-- Web Services interface to manage your investment portfolio.** For the operation, you can replace the documentation with, for example **-- Decomposing an XML document and storing the information into a DB2 XML collection table.**

4. **Replace the `<query>` tag with a `<storeXML>` tag.**

   The `<storeXML>` tag instructs DB2 XML Extender to shred XML documents into relational data, which means that it decomposes the document and stores untagged elements or attributes content, which can be indexed by DB2 UDB and accessed using SQL. It also provides new data types, functions, and stored procedures to manage XML data in DB2 UDB. When the data in the XML file is changing from time to time, it should be stored in XML collection. XML column is more suitable for large and relatively static XML document.

   Figure 4-13 shows the processing flow of the `storeStockMutualFundInfo` request. The XML document containing the stock or mutual fund information is sent to the Web Service. This service calls the XML Extender's stored procedure, `db2xml.dxxShredXML()`, and passes values for the DAD and XML document parameters. The XML Extender then shreds the stock or mutual fund information data in the XML document into the DB2 UDB database. The data is stored in the `Stock_MutualFund_info` table according to the generic relational database mapping specified in the DAD file.
Nested inside the <storeXML> tag, you have to specify the DAD files you are going to use with this operation. In our case, we use:

```xml
<DAD_ref>stock_mutualFund_info.dad</DAD_ref>
```

This is the DAD file we created in “Step 2: Creating and populating the data sources” on page 136.

Your DADX file should look like Example 4-9

**Example 4-9  DADX file**

```xml
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
        Working with your stock and mutual funds portfolio.
    </documentation>

    <operation name="storeStockMutualFundInfo">
        <documentation xmlns="http://schemas.xmlsoap.org/wsd1/">
        </documentation>
    </operation>
</DADX>
```
Decomposing an XML document and storing the information into a DB2 collection table.

```xml
<storeXML>
  <DAD_ref>stock_mutualFund_info.dad</DAD_ref>
</storeXML>
</operation>
</DADX>

Step 6: Testing the stockInfo Web Service
To test the storeStockMutualFundInfo method, complete the following steps:

1. **Launch the Web browser** and enter the following address as shown in Figure 4-14:

   http://localhost:9080/services/stock/stockInfo.dadx/TEST

   ![Figure 4-14 Test client for stockInfo.dadx](image)

   - Select a method to test.

   Enter input parameters and invoke the method.

2. **Select the storeStockMutualFundInfo method** in the Methods panel.
   You will see a text input area appear inside the Input panel area as in Figure 4-15.
3. **In the Inputs panel, fill in an XML document** in the provided text area. The XML document should not include the XML tag. For our scenario, we provide a number of XML files for the different stock and mutual fund. We give an example in Example 4-10.

**Example 4-10  XML file example**

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE stock_mutualFund_info SYSTEM "stock_mutualFund_info.dtd">

<stock_mutualFund_info key="1">
  <symbol>FGFND</symbol>
  <description>Fred Growth Fund</description>
  <type>Mutual Fund</type>
  <sector>Hi Tech</sector>
  <price>7.65</price>
  <market>NYSE</market>
</stock_mutualFund_info>
```
When inputting the XML file through our Web client, you do not need to include the `<xml>` and `<DOCTYPE>` tags. So, copy the rest of the xml document and paste the XML into the input area.

4. Invoke the method by clicking **Invoke**.
   
   You should see Figure 4-16 after invoking the method.

![Figure 4-16 Result from invoking the storeStockMutualFundInfo method.](image-url)
5. To further verify, you start a DB2 command line processor or a command window, connect to the database and issue the SQL command:

```sql
select * from stock_mutualfund_info.
```

You should see 1 row of data which matches the data in the XML file.

**Attention:** Be sure to import the rest of the stock and mutual fund information. Else your subsequent functions may not yield results as expected. Another important point is the decomposing operation acts like an SQL INSERT. If you want to update the XML Collection by changing the XML and submitting the new values to DB2 UDB, you have to delete the row of information from the collection first. We could also build a SOAP client that does the delete and call the storeStockMutualFundInfo Web Service function.

Now you are ready to complete the rest of the methods for our stockInfo Web Service.

**Step 7: Add additional functions to an existing Web Service**

Open up the file stockInfo.dadx with your favorite text editor. You will do this for each method you are going to add to your Web service.

1. **View a customer’s portfolio information:** To view a customer’s portfolio requires us to first place a request to WebSphere MQ to obtain customer profile information from WebSphere MQ. Then we have to issue an SQL statement to join data from Informix database, DB2 UDB database, WebSphere MQ queues and XML data.

   Complete the following steps to add in a method to view a customer’s current portfolio:
   a. Add in a new operation, name it makeMQRequest.
      ```xml
      <operation name="makeMQRequest">
      </operation>
      ```
   b. Optionally, add documentation to your newly added operation. Make sure you add the documentation between the opening and closing operation tags.
      ```xml
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
      Call a stored procedure to make a MQ request to obtain customer profile information from VSAM.
      </documentation>
      ```
We are calling a stored procedure because DADX supports query operations with SQL SELECT statements only. Sending requests to MQ, we need to use a VALUES statement. At the same time, we can demonstrate the integration between DB2 Web Service and stored procedures.

c. Add in a call tag to specify the stored procedure you want to invoke in DB2 UDB using Web Service.

```
<call>
  <SQL_call>
    CALL MQ_REQUEST(:custID, :rowsAffected, :errorCode, :errorLabel)
  </SQL_call>
</call>
```

The last piece of information you add is the parameters definition:

```
<parameter name="custID" type="xsd:string" kind="in"/>
<parameter name="rowsAffected" type="xsd:int" kind="out"/>
<parameter name="errorCode" type="xsd:int" kind="out"/>
<parameter name="errorLabel" type="xsd:string" kind="out"/>
</call>
```

You will quickly realize where does the stored procedure MQ_REQUEST come from. The answer is you have to develop the stored procedure.

We show you the code for the stored procedure and how to catalog your stored procedure in DB2 UDB. We develop the store procedure using Java. We implemented the stored procedure in a file named SpMakeMQRequest.java

The source code for the stored procedure is listed in Example 4-11.

---

**Example 4-11   Stored procedure**

```java
import java.sql.*;              // JDBC classes
import java.io.*;               // Input/Output classes
import java.math.BigDecimal;    // Packed Decimal class

///////
// Java stored procedure is in this class
///////
public class SpMakeMQRequest
{
  public static void mqRequest(String customerID,
                              int[] numberOfRequest,
                              int[] errorCode,
                              String[] errorLabel) throws SQLException
  {
    try
    {
      String senderService = "IIR.BCA1.SENDER";
      String amiServicePolicy = "IIR.TEST,POLICY";
```
Chapter 4. Business case II: Accessing your information through Web Services

String requestCmdPrefix = "GET CUST INFO   ";

StringBuffer tempCustID = new StringBuffer(customerID);

for (int i = 0; i < (10 - customerID.length()); i++)
    tempCustID.append(' ');

// Initialize variables
    counter = 0;
    errorCode[0] = 0; // SQLCODE = 0 unless SQLException occurs

    // Get caller's connection to the database
    Connection con =
        DriverManager.getConnection("jdbc:default:connection");
    errorLabel[0] = "GET CONNECTION";

    StringBuffer query = new StringBuffer("VALUES db2mq.MQSEND('" +
        senderService + ",' + amiServicePolicy + ",',");
    query.append("' + requestCmdPrefix + tempCustID.toString() + ",");
    query.append("' + tempCustID.toString() + ")");

    errorLabel[0] = "CREATE VALUES DB2mq.MQSEND() statement";
    Statement stmt = con.createStatement();
    errorLabel[0] = "GET THE RESULT SET";
    ResultSet rs = stmt.executeQuery(query.toString());

    // move to first row of result set
    rs.next();

    // set value for the output parameter
    errorLabel[0] = "GET AFFECTED NUMBER OF ROWS";
    numberOfRows[0] = rs.getInt(1);

    // clean up first result set
    rs.close();
    stmt.close();

} catch (SQLException sqle)
{
    errorCode[0] = sqle.getErrorCode();
}

}
In the stored procedure, we construct the **VALUES** statement to call **MQSend**. Upon completing the WebSphere MQ request, you should get one row back from the SQL statement and the stored procedure reports no error.

Before you can use the stored procedure you have to catalog the procedure in the database. You have to:

i. Compile your stored procedure into class file.

ii. Move the class file to `<DB2_installed_dir>`\function directory

iii. Catalog your stored procedure using the following commands after you have connected to your database:

```sql
DROP PROCEDURE MQ_REQUEST (CHAR(10), INTEGER, INTEGER, CHAR(32))

CREATE PROCEDURE MQ_REQUEST (IN customerID CHAR(10), OUT
numberOfRowsAffected INTEGER, OUT errorCode INTEGER, OUT errorLabel
CHAR(32))
DYNAMIC RESULT SETS 0
LANGUAGE JAVA
PARAMETER STYLE JAVA
NO DBINFO
FENCED
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'SpMakeMQRequest.mqRequest'
```

After you catalog the stored procedure, you can use your newly created stored procedure.

d. Save your DADX file. Launch a Web browser and use the same URL you used before:

http://localhost:9080/services/stock/stockInfo.dadx/TEST

Your Web Service should have one more method by the name of `makeMQRequest`. If you select it, the Inputs pane will prompt you for a custID. You can input 1001, 1002, or 1003 for testing purposes. Let us enter 1001 and you should see results similar to Figure 4-17.

We place a MQ request through our Web Service method. Our CICS program will be triggered and places a reply into the reply queue. You get the value “1” in the out parameter `numberOfRowsAffected`. 
e. Add in a new operation, name it viewMyPortfolio.

   <operation name="viewMyPortfolio"/>

f. Optionally, add documentation to your newly added operation. Make sure you add the documentation between the opening and closing operation tags.

   <documentation xmlns="http://schemas.xmlsoap.org/wsdl/"
   View my current portfolio.
  </documentation>
g. Add in a query tag as in Example 4-12 to specify what data you want to expose in DB2 to the Web Service.

**Example 4-12  Query tag**

```sql
<query>
<SQL_query>
SELECT
  B.ID,
  rtrim(T.LAST) || ', ' || rtrim(T.FIRST) as Name,
  B.SYMBOL,
  A.DESCRIPTION,
  B.QUANTITY,
  A.PRICE,
  B.QUANTITY * A.PRICE AS MARKET_VALUE,
  T.CITY
FROM
  (select
    CUST_ID as ID,
    STOCK_SYMBOL as SYMBOL,
    TOTAL_SHARES as QUANTITY
  from ids_ifmx92nt.stock_portfolio_tab where CUST_ID = :custID1
  union
  select
    CUSTOMER_ID as ID,
    MUTUAL_FUND_SYMBOL as SYMBOL,
    QUANTITY as QUANTITY
  from acc_mf_portfolio where CUSTOMER_ID = :custID2 ) AS B ,
  STOCK_MUTUALFUND_INFO A,
  (SELECT
    substr(msg,1,20) as REQ,
    substr(msg,21,4) as RETCODE,
    substr(msg,25,60) as ERR_MSG,
    substr(msg,85,10) as ID,
    substr(msg,95,30) as FIRST,
    substr(msg,125,1) as MI,
    substr(msg,126,30) as LAST,
    substr(msg,156,40) as ADRLN1,
    substr(msg,196,40) as ADRLN2,
    substr(msg,236,40) as ADRLN3,
    substr(msg,276,40) as CITY,
    substr(msg,316,20) as STATE,
    substr(msg,336,10) as ZIPCODE
  FROM
    TABLE(db2mq.MQRECEIVEALL('IIR.BC1A.RECEIVER',
      'IIR.TEST.POLICY', '1001      ', 1)) AS X ) AS T
WHERE A.SYMBOL=B.SYMBOL AND B.ID=T.ID
</SQL_query>
```
The last piece of information you add to your DADX file is the definitions of the parameters:

```xml
<parameter name="custID1" type="xsd:string"/>
<parameter name="custID2" type="xsd:string"/>
</query>
```

The last piece of information you add to your DADX file is the definitions of the parameters:

```xml
<parameter name="custID1" type="xsd:string"/>
<parameter name="custID2" type="xsd:string"/>
</query>
```

Note: DADX has a problem with reusing the same parameter name throughout your query. To alleviate the problem, we created two parameters that hold the same value.

Another oddity you will find in the query is, we can not use a variable to replace the correlation ID in the table function MQRECEIVEALL. If we place a variable there, WORF fails to obtain metadata information from DB2 UDB to validate our Web Service method. You may want to work out how you want to handle the correlation ID in building your future Web Services applications that integrate with WebSphere MQ.

h. Save your DADX file. Launch a Web browser and use the same URL you used before:

http://localhost:9080/services/stock/stockInfo.dadx/TEST

i. Select viewMyPortfolio in the Methods panel, enter value 1001 for custID1 and custID2 in the Inputs pane. Click **Invoke**. If you have successfully send a request to MQ, you should be able to obtain the following result. (Figure 4-18)

In this query, we joined the federated stock portfolio table from Informix, federated mutual fund table from DB2 UDB v7.2, DB2 XML collection from the XML files and the MQ reply as a data store.
2. View a customer’s recent trading transactions

Adding this function in our Web Service is very straightforward, repeat what we have done in the View a customer’s portfolio. The SQL statement is different but more straightforward since we are getting data from just one table.

Complete the following steps to add in a method to view a customer’s recent transactions:

a. Add in a new operation, name it viewMyRecentTransactions.

    <operation name="viewMyRecentTransactions"/>

</operation>
b. Optionally, add documentation to your newly added operation. Make sure you add the documentation between the opening and closing operation tags.

   <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
   View a list of a customer’s recent trading transactions.
   </documentation>

c. Add in a query tag select as in Example 4-13 to specify what data you want to expose in DB2 to the Web Service.

   Example 4-13  Query tag: select

   <query>
   <SQL_query>
   Select cust_id,
   order_number,
   transaction_date,
   action_indicator,
   symbol,
   quantity,
   price
   from orders
   where cust_id = :custID and
   current_date - transaction_date &lt;= :daysOfHistory
   </SQL_query>
   </query>

   The last piece of information you add to the DADX file is the definitions of the parameters:

   <parameter name="custID" type="xsd:string"/>
   <parameter name="daysOfHistory" type="xsd:int"/>
   </query>

d. Save your DADX file. Launch a Web browser and use the same URL you used before:

   http://localhost:9080/services/stock/stockInfo.dadx/TEST

e. Select viewRecentTransactions in the Methods pane, enter value 1001 for custID and 30 for numOfDays in the Inputs pane. Click Invoke. You may not be getting any data back.

   Note: You can't use the '>' and '<' SQL comparison operators in the DADX file. You have to use the XML escape sequences to replace these characters.
3. Initiating a buy or sell order (inserting a row in the database)

We simply add a row of data into the orders table. Complete the steps same as the above two methods. This function demonstrates the update capability of the DB2 Web Services.

   a. Add in a new operation, name it makeAnOrder.

   <operation name="makeAnOrder"/>

   </operation>
b. Optionally, add documentation to your newly added operation. Make sure you add the documentation between the opening and closing operation tags.

<documentation xmlns="http://schemas.xmlsoap.org/wsdl/"
    >
Make an order to buy or sell a stock or mutual fund.
</documentation>

c. Add in a query tag update as in Example 4-14 to specify what data you want to expose in DB2 to the Web Service.

Example 4-14  Query tag: update

<update>
  <SQL_update>
    insert into orders ( 
      cust_ID, 
      symbol, 
      action_indicator, 
      transaction_date, 
      quantity, 
      price
    ) values ( 
      :custID, 
      :symbol, 
      :action, 
      :current_date, 
      :quantity, 
      :price 
    )
  </SQL_update>
</update>

You have to add the following parameters to your DADX file:

<parameter name="custID" type="xsd:string"/>
<parameter name="symbol" type="xsd:string"/>
<parameter name="action" type="xsd:string"/>
<parameter name="quantity" type="xsd:decimal"/>
<parameter name="price" type="xsd:decimal"/>
</update>

d. Save your DADX file. Launch a Web browser and use the same URL you used before:

http://localhost:9080/services/stock/stockInfo.dadx/TEST

e. Select makeAnOrder in the Methods panel, enter the Table 4-11 values in the Inputs panel.
Table 4-11   Input values to test Web Service method makeAnOrder

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>custID</td>
<td>1001</td>
</tr>
<tr>
<td>symbol</td>
<td>SCGFD</td>
</tr>
<tr>
<td>action</td>
<td>B</td>
</tr>
<tr>
<td>quantity</td>
<td>100.00</td>
</tr>
<tr>
<td>price</td>
<td>65.87</td>
</tr>
</tbody>
</table>

f. Click **Invoke**. You should see the similar results as shown in Figure 4-20.

Figure 4-20   Web Service method makeAnOrder sample results
You can verify the row has been added to the database by using the viewRecentTransactions method.

**Step 8: Packaging DB2 Web Services into Web application**

It is easier to package the DB2 Web Services using a development tool. We are going to defer this topic to the section “Packaging and deploying previously built Web Service” on page 185.

**Step 9: Deploying DB2 Web Services**

We are going to defer the discussion to the section “Packaging and deploying previously built Web Service” on page 185.

Congratulations! You have successfully created the entire stockInfo Web Service. The complete listing of the stockInfo.dadx can be found in Appendix D.

We will build a Web Service method with IBM WebSphere Studio Application Developer.

### 4.5.2 Build Web Services with WebSphere Studio Application Developer

Here we describe the second alternative for building Web Services. In this section, we are going to build one of the methods that were specified in the previous section, but this time, we will use WebSphere Studio Application Developer. We will explore the SQL wizard in WebSphere Studio Application Developer to create a Web Service from a DADX file. An alternative solution might also be to use the wizard in WebSphere Studio Application Developer to generate a Web Service from an URL.

This example illustrates:
- Creating an SQL statement using the SQL wizard
- Creating a DADX group using the DADX group configuration wizard
- Creating a DADX file from the SQL statement
- Running the Web Service wizard from the DADX file
- Testing the Web Service using the generated sample application
- Generating the DB2 Web application
- Deploying DB2 Web Services

Table 4-12 summarizes the steps you have to go through to build DB2 Web Services using WebSphere Studio Application Developer. We assume all the steps outlined here can be carried out by an application developer.
Complete the following steps to create a DB2 Web Service:

**Step 1: Prepare a project**
Create a new workspace directory, name it `myStockInfoWorkspace` on the root of your C:\ drive. In this directory, WebSphere Studio Application Developer will store all your project files and information. Create a shortcut of WebSphere Studio Application Developer on your desktop and give the shortcut the following target:

```
c:\program files\IBM\Application Developer\wsappdev.exe -data c:\myStockInfoWorkspace
```

Assuming that you installed WebSphere Studio Application Developer in the directory `c:\program files\IBM\Application Developer`.

Start WebSphere Studio Application Developer (Application Developer) by double clicking the newly created shortcut on your desktop. Do not start Application Developer from the Start menu because the Application Developer in your Start menu will be using a default workspace. We want Application Developer to be using the workspace we have just created.
You will have to set up:

- An EAR project, name it StockInfoEAR.
- A Web project, name it StockInfoWeb.
- A WebSphere server configuration, name it StockInfoWebSphere.

To create an EAR project, complete the following steps:

1. Select **File** from the menu bar.
2. Click **New**, then **Other**...
3. Select **J2EE** and **Enterprise Application Project**.
4. Fill in the information to create the EAR project and the Web project. Make sure to deselect Application client project name and EJB project name. Once again, name the EAR project StockInfoEAR and the Web project StockInfoWeb.
5. Click **Finish**.

You have created the EAR as well as the Web project. To create a WebSphere server configuration, complete the following steps:

1. Right-click the **Server Configuration** in the J2EE View window using the J2EE Perspective, and select **New** and **Other**....
2. Select **Server** and **Server Configuration** from the New window. Click **Next**.
3. Enter the name of the server configuration and a name for the folder. Use StockInfoWebSphere to be the name of the configuration and enter StockInfoWebSphereServer for Folder. For configuration type, expand WebSphere Servers and select **WebSphere v4.0 Configuration**. Click **Next** if you want to specify a different port number for your WebSphere test environment, the default is **8080**, else, you can click **Finish** to create the configuration as well as the server project.
4. Once the server configuration is created, you want to add our StockInfoEAR into our server configuration. Expand **Server Configurations**, and right-click **StockInfoWebSphere**. Select **Add Project** and select the only project that is available, our **StockInfoEAR**.

You have created the server configuration and add the EAR project to the server configuration. You have to create a folder StockInfoDB in the StockInfoWeb project to contain the database information and the SQL statement, complete the steps as follow:

1. Expand Web Modules in the J2EE View pane and right-click **StockInfoWeb**.
2. Select **New** and **Folder**.
3. Select StockInfoWeb and enter the name of the folder: **StockInfoDB**.
4. Click **Finish**.
You need to verify the creation of the folder in another perspective. Open the Web perspective and expand on StockInfoWeb. You should see the StockInfoDB folder under StockInfoWeb.

**Step 2: Create an SQL statement**

Follow the steps to create an SQL statement:

1. Open or switch to the data perspective. Choose the Data view by clicking the Data tab at the left bottom of the Application Developer window.

2. Select **File -> New -> Other -> Data -> SQL Statement**, and click **Next**.

3. In the SQL Statement wizard select Be guided through creating an SQL Statement and Connect to a database and import a new database model. Click the Browse button next to the Connect to a database and import a new database model Folder and select StockInfoDB as your target folder. Specify viewRecentTransactions for the SQL Statement Name (Figure 4-21)

4. Click **Next**.

5. Complete the Database Connection panel (Figure 4-22)
   a. Specify StockInfoDBCon for the Connection Name, IIREDBOK for the database and make sure to specify a valid userid and password for DB2 (or leave the fields empty). In our case, we use db2admin for the User ID as well as password.

   **Note:** Please keep the database aliases consistent when you are working through the scenario.

   b. Click **Connect to Database** to verify connectivity. During the process, the database metadata is gathered and downloaded. This opens the Construct an SQL Statement wizard at the same time:

   - On the Tables tab, expand db2admin and Tables. Select the Orders table and click the arrow button to move the table to the Selected table panel.

   - Switch to the Columns tab of the Construct an SQL statement wizard, expand Orders. Select CUST_ID, ORDER_NUMBER, SYMBOL, ACTION_INDICATOR, TRANSACTION_DATE, QUANTITY, and PRICE columns and move them to the Selected columns list.
• Click **Next** and the SQL statement is displayed:

```plaintext
SELECT
    DB2ADMIN.ORDERS.CUST_ID,
    DB2ADMIN.ORDERS.ORDER_NUMBER,
    DB2ADMIN.ORDERS.SYMBOL,
    DB2ADMIN.ORDERS.ACTION_INDICATOR,
    DB2ADMIN.ORDERS.TRANSACTION_DATE,
    DB2ADMIN.ORDERS.QUANTITY,
    DB2ADMIN.ORDERS.PRICE,
FROM
    DB2ADMIN.ORDERS
```

![Figure 4-21 Create an SQL statement using Application Developer](image)

• Click **Execute** if you want to test the SQL Statement.
c. Click Finish to end the wizard. You can find the database information (connection, database, tables, SQL statement) in the StockInfoDB folder in the Data perspective under the Navigator tab.

Step 3. Create a DADX group

The Web Service DADX group configuration wizard enables you to create a DADX group that is used to store DADX documents. A DADX group contains connection (JDBC and JNDI) and other information that is shared between DADX files. You can import DADX files into the DADX group, and then start the Web Service wizard to create a Web Service from a DADX file. Complete the following steps:

1. Open or switch to the Web perspective.
2. Select File -> New -> Other -> Web Services -> Web Services DADX Group Configuration. Click Next.
3. In the Web Service DADX Group Configuration wizard, select the StockInfoWeb project and click Add group.

4. Specify StockInfoGroup as the name for the DADX group. Click OK.

5. Expand the StockInfoWeb project, select the StockInfoGroup and Group properties.

6. In the DADX Group Properties dialog box, change the database name in the DB URL to jdbc:db2:IIREDBOK and provide a valid User ID and Password (or empty fields). Click OK (Figure 4-23).

7. Click Finish to end the wizard. A folder groups\StockInfoGroup with the properties is added under the source folder.

**Step 4. Create a DADX file from the SQL statement**

Follow these steps:

1. Open or switch to the Data perspective, use the Data view.

2. Select File -> New -> Other -> XML -> XML from SQL Query. Click Next.

3. Select Create DADX from SQL query. Click Next.

4. Expand StockInfoWeb until you find the viewRecentTransactions SQL statement and select it. Click Next.
5. Skip the Select DAD files panel. If you want to work with a DAD file, this is where you import your DAD file.

6. In the DADX generation wizard specify StockInfo.dadx as the file name and select /StockInfoWeb/source/groups/StockInfoGroup (the folder of the DADX Group created in the previous step) as the output folder (Figure 4-24). Click Finish.

7. The wizard generates the DADX file, StockInfo.dadx, in the source/groups/StockInfoGroup folder in the StockInfoWeb project and opens the file in Design mode of the editor. Click the Source tab and you will see:
This file defines one operation, viewRecentTransactions, which refers to the SQL statement we defined. The SQL statement itself, wrapped within the CDATA block, is also contained within the DADX file.
Step 5. Run the Web Service wizard from the DADX file
We start the Web Service wizard from the DADX file:

1. In the Web perspective, select the StockInfo.dadx file and New -> Other -> Web Services -> Web Service. Click Next.

2. On the first panel make sure DADX Web Service and StockInfoWeb is selected. Select Generate a proxy and Generate a sample. Click Next.

3. The StockInfo.dadx file is preselected, click Next.

4. On the Web Service DADX Group properties, the properties panel defined in Figure 4-23 on page 177 is displayed click Next.

   Be patient, the server will be started and the Web application is published.

5. On the Binding Proxy Generation panel, change the proxy class to:
   itso.wsad.dadx.proxy.soap.StockInfoProxy
   select Show mappings and click Next.

6. On the Web Service XML to Java Mappings panel, select Show and use the default DOM Element mapping. Click Next.

7. Skip the rest of the remaining panels, or go through them by clicking Next, and click Finish.

At this point, Application Developer generates a bunch of associated files:

- A new servlet StockInfoGroup, with DxxInvoker as the servlet class, is added to the Web application deployment descriptor. This servlet is a subclass of the rpcrouter servlet. It handles the incoming SOAP requests and sends the response back. The URL mapping, /StockInfoGroup/*, matches the name of the DADX group.

- The client proxy class in itso.wsad.dadx.proxy. See Figure 4-26

- A mapping class, ViewRecentTransactionsResult_ElementContentType.java, in a new mappings folder.

- The SOAP admin application in the webApplication\admin folder.

- The test client in the sample StockInfo folder.

- The WSDL service interface file StockInfo-binding.wsdl and the service implementation file StockInfo-service.wsdl are generated in the wsdl folder.

- The StockInfo.isd file in the WEB-INF\isd\dad folder. This is the deployment descriptor.

- The SOAP deployment files, dds.xml and soap.xml.

- A worf folder with an HTML-based Web Service test facility.
Step 6: Test the Web Service

To test the Web Service, launch the sample application from if the application has not been started:

```
webApplication\sample\StockInfo\TestClient.jsp
```

Figure 4-26 Sample client proxy code (extract)

Note: We could not overwrite the URI of the WEB service in the wizard:

```
http://tempuri.org/StockInfoGroup/StockInfo.dadx
```

This value is in the ISD file (and therefore in dds.xml) and in the client proxy.

The target URL points to /StockInfoGroup, which is mapped to the DxxInvoker servlet defined in the Web application:

```
http://localhost:8080/StockInfoWeb/StockInfoGroup/StockInfo.dadx/SOAP
```

Chapter 4. Business case II: Accessing your information through Web Services 181
If your application is already started, you can use this URL to invoke your application:

http://localhost:8080/StockInfoWeb/StockInfoGroup/StockInfo.dadx/TEST

- Click the viewRecentTransactions method link in the left pane. Click Invoke in the right pane. The results of the query should appear in XML format in the right bottom pane as shown in Figure.

Figure 4-27  Testing the Web service using the sample

**Step 7: Export application as EAR file**

You can export the StockInfo Enterprise Application to an EAR file and use the EAR file to deploy to WebSphere Application Server. To export the application to an EAR file:

1. Open or switch to a J2EE perspective.
2. Expand Enterprise Applications and right-click StockInfoEAR.

4. On the EAR Export screen, StockInfoEAR is preselected for you. You have to enter a destination as to where you want to place the EAR file. We choose to export the application into c:\temp\StockInfo.ear.

5. Click Finish.

6. Close Application Developer and start WebSphere AdminServer.

Step 8: Deploy the EAR file in WebSphere Application Server

Complete the steps below to deploy your StockInfo application to WebSphere Application Server:

1. Start WebSphere AdminServer through Windows Services or Start -> Programs -> IBM WebSphere -> Application Server V4.0 AE -> Start Admin Server.

   Be patient, it will take a a minute or two for the admin server to be started.

2. Start the WebSphere Administrator's Console by going to Start -> Programs -> IBM WebSphere -> Application Server V4.0 AE -> Administrator's Console.

3. Expand WebSphere Administrative Domain, right-click Enterprise Applications, and select Install Enterprise Application.

4. Using the Install Enterprise Application Wizard, we choose Install Application. Enter the EAR we just exported from WebSphere Studio Application Developer and provide an application name, stockInfoApp. (Figure 4-28)

5. Click Next eight times. We are prompted to select a Virtual Host. Click Next to use the default_host.

   Note: Your Web Module is StockInfoWeb. This is the name we use to create the Web Module in Application Developer. In addition, since we are installing the application as an enterprise application, we can not change the context root of the application at this point. We change the context root later.

6. Click the Select Server button to choose a server the application is going to run on. Select Default Server(<your hostname>) to be the application server. Click Next.

7. Click Finish.

8. Expand Enterprise Applications to see your newly installed application. Select your application and start your application by pressing the Start button in the action bar, or right-click the application name and select Start.

9. Test your application entering the following URL in a Web browser:

We are going to switch it to use the SOAP default configManager. This is done by renaming the soap.xml file found under the directory:

c:\WebSphere\AppServer\installedApps\stockInfoApp.ear\StockInfoWeb.war

Rename the soap.xml to soap-ibm.xml

Stop and restart your application using the WebSphere Administrator's Console. You should be able to run the sample now. (Figure 4-29)
Chapter 4. Business case II: Accessing your information through Web Services

Packaging and deploying previously built Web Service

We are going to package and deploy our DADX file we built in the last section. To do so, following the steps:

- Move the following files from the sample services WebSphere installedApp directory into the Application Developer workspace. Using our example the WebSphere directory is:
  
  c:\WebSphere\AppServer\installedApp\servicesApp.ear\services.war\WEB-INF\classes\groups\stockInfo

  The Application Developer workspace directory is:
  
  c:\myStockInfoWorkSpace\StockInfoWeb\source\groups\StockInfoGroup
You are moving the following files:
- group.properties
- namespacesetable.nst
- Stockinfo.dadx
- stock_mutualFund_info.dad

By placing the file into the workspace area, we are essentially adding the files into our StockInfoWeb project.

- Stop the WebSphere Admin Server and the IBM HTTP Server and start Application Developer with the workspace we created. There should be an icon on your desktop.
- Open or switch to the Web perspective. Right-click the StockInfoWeb project and select Refresh From Local. This will bring the copied files into the workspace.
- Right-click the StockInfoWeb project and select properties.
- Select Web and change the context root. We named it stockInfoService. We can do it after we deploy the application on WebSphere also. We chose to do it here.
- Expand webApplication under the StockInfoWeb and expand the WEB-INF folder to find web.xml.
- Open web.xml into the editor and you can change your servlet mapping.
- Click the Servlets tab and add URL mappings. We add this URL mapping: /stockInfo/*. Go ahead and remove the old one. Save the xml file by pressing CTRL-S.
- Right-click StockInfoWeb project and select Rebuild Project.
- Open and switch to the J2EE perspective. Expand Enterprise Applications and right-click the StockInfoEAR project. Select Export EAR File...
- After you export the EAR file, do steps 1 through 9 again to deploy the newly exported enterprise application. You may need to remove the application you installed last since they shared the same module name.
- You can test your new stockInfo Web Service application by using the URL: (see Figure 4-30):
4.6 Further steps

This scenario can be enhanced using DB2 Information Integrator to expand its federated access to heterogeneous relational databases as ORACLE and access to other unstructured data as XML files. Federation example on ORACLE instead of Informix is provided in “Further steps” on page 110.
4.7 Integration considerations

We have learned step-by-step and in detail how to construct DB2 Web Services using the different type of operations. Through Web Services, we can INSERT, UPDATE, and DELETE data stored in DB2 UDB. We can easily access information stored or federated in DB2 through the SQL query operation. A more powerful integration between Web Services and DB2 is the ability to invoke stored procedures in DB2 UDB. The DB2 stored procedures can eliminate many restrictions when you are defining your Web Services through DADX.

What we presented here is just a small fraction of what you can do with the Web Services and DB2 integration. No doubt, the integration between Web Services and DB2 will continue to evolve. A richer set of functionality will soon be on the horizon. But what we experience today with DB2 Web Services is sufficient to benefit those enterprises that need to integrate their data and information with their business partners.

Web Services is here to stay and soon you will be working on a Web Service project. So far, Web Services is all good, so, where is the bad? Before you start any Web Services project, there are a few issues you might want to consider, as presented in the following sections.

4.7.1 Does Web Services mean interoperability?

The concept of Web Services is very abstract. The current definition proposed by the W3W's Web Services Architecture working group roughly boils down to a software application accessible via Internet protocols using XML for messaging, description, and discovery. Depending on whom you ask, Web Services are simply an incarnation, a generalization, or a reinvention of distributed computing. It is the Web's version of CORBA or DCE but much more than a general means for remote procedure calls. Even though the idea of Web Services is very general, the software community has focused on three specific standards as the core of Web Services infrastructure:

- SOAP
- WSDL
- XML Schema

These give us, respectively, a wire format, an interface definition, and a type system, all of which are platform neutral. In addition, they are also transport independent, language independent and extensible. So we have a distributed computing framework which can potentially be implemented above the transport that is most suited to the task and can be extended with technologies such as new type systems that are suited best to a particular task. This generality does wonders for the shelf-life of the standards but has unfortunately added to the complexity to the system.
In order to communicate with a Web Service, it is not enough to know the WSDL, SOAP and XML Schema. Instead it involves, among other things, understanding particular extensions to WSDL, the specific transports over which to send SOAP envelopes, and the data encoding expected by the service. Just as XML is often joked about as being a standard whose sole purpose is to create more standards. The family of Web Services standards seems to share that same property. We will soon see some higher level specifications making use of WSDL and SOAP in some specific manner. In Web Services terminology, these higher level specifications would be referred to as bindings. Each of the main specifications contains ambiguities that can stifle interoperability, thus requiring understanding of the specific interpretations of the specifications.

Before jumping on the band wagon of Web Services, other than the interoperability issue you are going to face with, you have to deal with other issues such as revenue, security, and management.

### 4.7.2 Revenue and contracts

The provision of Web Services has costs and revenue opportunities associated with it. Service providers want to earn revenue based upon service consumption. This typically involves a negotiated contract between a consumer and a provider. However, the provider have to ensure that none of these contract specifics is incorporated in development and deployment of its Web Service since the provider may have many consumers. The managing and maintaining of these contracts are not as simple as the Web Services discovery and binding.

Further complicating the issue is the lack of deployed standards for usage and billing. If enterprises fail to charge for their services, it does not make enough business sense to stay in the business as service providers.

### 4.7.3 Web Services security

Similar to any applications, it is essential for Web Services have some security model built in. For our portfolio Web service, we should restrict access to the functions to those who are doing business with us and not any arbitrary user. Web Services must be guarded by an authentication and authorization model. The service provider will specify how authentication information will be encoded in the message. There are currently no standards widely deployed for encoding security credentials in a SOAP message.

To address the Web Services security problem, IBM, Microsoft and VeriSign have published a Web Services security (WS-Security) specification that provides a foundation for companies to develop interoperable Web Services without the fear of exposing sensitive data or violating privacy practices.
WS-Security defines a standard set of SOAP extensions you can use to ensure integrity and confidentiality in Web Services applications. The WS-Security specification will help you exchange secure, signed messages in a Web Services environment.

WS-Security is not a complete security solution; rather, it is a building block to be used with other Web Service and application-specific protocols to accommodate a wide variety of security models and encryption technologies. However, no standards organization is supporting the specification at the time of this writing. You can download the specification from the Web site and get a head start on Web Services security:


4.7.4 Quality of Service

The majority of today’s Web Services are generally not concerned about the level of Quality of Service (QoS) presented to their users. However, the number of is small but increasing. New Web Services demand delivery of multimedia data in real time, and the information transfer via the Internet is becoming one of the principal paradigm for business: online sales, banking, finance, collaborative work, are few examples of this. The QoS perceived by it users is thus becoming a dominant factor for the success of an Internet based Web Service.

The principal QoS attributes matter the most to users are:

- Responsiveness
- Availability
- Timeliness

A service that is frequently unavailable is detrimental to the reputation of the service provider. A service exhibits poor responsiveness is virtually equivalent to an unavailable service.

The performance perceived by the users of a Web Service depends on the performance of the protocols that operate between Web clients and servers. An IP network currently provides only a best effort service and QoS is not guaranteed. Meanwhile, the middleware must cope with the insufficient bandwidth and high latency.

The focus on the design and analysis of middleware techniques and protocols for providing Web Services with QoS guarantees by addressing the following issues:

1. Document retrieval latency
2. Data availability
3. Amount of data to transfer
4. Redistributing the network accesses to avoid network congestion
Some proposed architectures that adopt servers's replication to construct Web Services with QoS guarantees. Other present and evaluate an approach for providing a geographically replicated Web Service.

No matter what the approach is, we still have a lot of research ahead of us on QoS for Web Services.

### 4.7.5 Web Services management

The question here is how do we enable an enterprise to manage Web Services as business resources and to fine-tune operations of the different Web Services. Web Services management can be thought of as a type of intelligent resource management that must understand both the business context of a service request, and the underlying technologies that deliver the server.

When businesses deploy Web Services, they need to understand how these Web Services are impacting their IT infrastructure. In addition, they need business-oriented control of these Web Services. A Web Services management system provides information, measurement and monitoring function of the IT systems as business resources, with the ability to dynamically take action to optimize Web Services behavior.

In any Web Service deployment environments, it is vital to manage resources such as:

- Real time or historical Web services activity, monitoring and visualization, load balancing and Web service traffic information.
- Performance tuning that addresses the Service Level Agreement (SLA), QoS monitoring, metering and alerting.
- Prioritizing requests based on enterprise’s rules and policies.
- Billing and rating of services, allocation and tracking of costs and revenue.
- Maintaining and controlling different versions of a service to different consumers.

There are tools available now and many more players are promising to bring their own solutions to the table. For the time being, we can wait and see how the industry is going to provide solutions to address the different aspects of Web Services deployment.
4.8 Conclusion

In this chapter, we have learned to build and construct Web Services through DADX files — that means to be able to build Web Services by editing a DADX file. Alternatively, we can use wizards provided in WebSphere Studio Application Developer to construct Web Services. We also considered how to create a DAD file that works with DB2 XML Extender, and how we can use DADX and DAD to provide an integrated solution that utilizes Web Services and DB2 XML Extender.

WORF, the framework that enables the integration with Web Services, was also introduced in this chapter.

Web Services provides an architecture for integrating applications, and DB2 UDB and DB2 Information Integrator provide the capability to manage data and provide intelligent, optimized access to data. With Web Services, DB2 UDB, and information integration, we can provide a quick and easy integration of diverse back-end data and provide an industry standard interface for application integration.
Chapter 5. Business case III: Incorporating XML data

In this chapter we use the XML, WebSphere MQ, and federated integration capabilities of DB2 UDB to address the business problem defined in Chapter 4, “Business case II: Accessing your information through Web Services” on page 119. The solution to this business problem is presented as three possible alternatives to shred, compose, and manipulate XML data:

- The first alternative uses the DB2 XML Extender feature.
- The second alternative uses the SQL/XML extensions.
- The third alternative uses the MQSeries Integration feature.

In addition, a heterogeneous vendor virtual (federated) database is accessed and updated by each of the three alternatives.
5.1 Business problem statement

The problem defined in 4.1, “Business problem statement” on page 121 through Web Services chapter will be reused as the problem domain for this chapter.

In summary, though the Almaden Credit Union (Almaden) currently provides internet access to customers for general self-service banking (as checking account balances), investment banking services are not presently an option.

As a result, Almaden has outsourced the investment banking Web site development and management duties to HLED Investments. Almaden will provide customer information to HLED Investments, however, to present a common customer view in processing in investment transactions. This integrated approach is seamless and has little impact to any Almaden client that is accessing the system. Using a Web browser, Almaden clients are able to achieve the following:

- View investment portfolios
- Buy stocks and mutual funds
- Sell stocks and mutual funds
- Receive status updates
- View investment transactions history online or downloaded in an XML document.

Although DB2 UDB functionality is categorized and presented in this chapter as separate options, in practice, a unique mix of DB2 UDB functions may be used to address a given business problem. Table 5-1 can be used to determine the most appropriate combination for choosing a particular implementation.
Table 5-1 Alternatives diagram

<table>
<thead>
<tr>
<th>Alternatives</th>
<th>Requirements</th>
</tr>
</thead>
</table>
| 1. DB2 XML Extender        | ▶ Store XML documents in DB2 UDB. The following options may be used:  
 |                            |   – Store the XML document, as is, in a relational table’s XML Column.  
 |                            |   – Shred the XML document and store in multiple relational tables as an XML Collection  
 |                            | ▶ Apply DB2 analytics, mining and business intelligence in processing XML documents.  
 |                            | ▶ Generate or compose an XML document as output from DB2 UDB.  
 |                            | ▶ Use synchronous communication between applications and DB2 UDB.  
| 2. DB2 SQL/XML             | ▶ Use vendor-neutral standard languages (SQL/XML and SQL) for XML and database processing. DB2 UDB will follow the standards to provide full support for the SQL/XML specification.  
 |                            | ▶ Only compose an XML document from data in DB2 UDB relational tables.  
 |                            | ▶ Synchronous communication between applications and DB2 UDB needed  
 |                            | ▶ Performance requirement for the most optimum solution, in comparison to alternative one, for dynamically composing XML documents.  
| 3. MQSeries Integration    | ▶ Send and receive text and/or XML data between DB2 UDB databases and external applications  
 |                            | ▶ Asynchronous and/or publish/subscribe communication between applications and DB2 UDB needed  
 |                            | ▶ Assured delivery quality of service between applications and DB2 UDB.  

5.2 Assumptions

The common assumptions for the three alternatives are:
Almaden client information is stored in a federated database (DB2 UDB and Informix database data sources). In addition, a WebSphere MQ queue is used as a data source and provides access to VSAM data.

A Web browser is used by Almaden clients to perform investment banking transactions.

XML documents are used as the canonical business event implementation for business transactions.

A Java servlet is used, via the Web browser, by Almaden clients to submit investment portfolio transactions to DB2 UDB.

For each alternative, additional assumptions are:

- **Alternative one**
  - DB2 XML Extender functions will be used to store (shred) XML documents into a relational format.
  - The database must be enabled with XML Extender.

- **Alternative two**
  - DB2 SQL/XML extension functions will be used to compose complete XML documents.

- **Alternative three**
  - MQSeries Integration functions will be used to send plain text and/or XML messages between DB2 UDB and remote client WebSphere MQ queues.
  - The WebSphere MQ system will be used to transport XML business events outside of the information integration framework.
  - The MQSeries Integration functions (basic) must be enabled. It should be noted that the database does not have to be enabled with the advanced XML Extender functions.

### 5.3 Solution overview

The XML and WebSphere MQ features available in DB2 UDB are implemented as DB2 user-defined functions and stored procedures. In particular, these functions provide a bridge between DB2 UDB and a wide variety of external systems. DB2 XML functions provides the capability to store XML data in a DB2 UDB database either intact using an XML Column or shredded (or decomposed) into relational tables using an XML Collection. Document Access Definition (DAD) XML files are used to configure a bi-directional mapping between the XML and relational data formats. MQSeries Integration functions provide the capability to send and receive plain text or XML messages between DB2 UDB and external systems. All DB2 XML and WebSphere MQ functions can be accessed from user
written DB2 UDB clients, user-defined functions or stored procedures using synchronous RPC calls (via SQL, SQLJ, ODBC and JDBC).

Please refer to Table 5-1 on page 195 for an explanation of the requirements that were considered in determining an implementation solution for each of the three outlined alternatives.

**Alternative one**
The DB2 XML features will be used as a solution to store investment transactions in DB2 UDB as XML documents. In addition, transactions summaries and portfolio information will be returned to clients in an XML format. Please see Figure 5-1 for an architecture diagram that highlights the components and information flow for alternative one.
Alternative two
The DB2 SQL/XML extension features will be used as a solution to compose well formed XML documents.

This alternative is the best option for Almaden clients that require the receipt of investment information in an XML format. The Web browser is used to submit and receive data in a synchronous manner. That is, a transaction request is submitted to the investment system and the Almaden investment client waits briefly until a reply is received in the browser. It should be noted that this is the normal behavior we all experience in using a Web browser interface to access the internet. The delivery of a request or a reply is not guaranteed. And, a communications failure would require re-transmission of a transaction using the Web browser. Please see Figure 5-2 for an architecture diagram that highlights the components and information flow for alternative two.
**Alternative three**

We will use the basic MQSeries Integration functions (for example, `MQSend()`) to place XML data on a WebSphere MQ queue that is located at an Almaden client's site. This alternative provides an option for Almaden clients that prefer to receive XML documents synchronously (using a Web browser) or asynchronously (using a WebSphere MQ queue). The XML data can be generated when requested by an Almaden client (using a Web browser) or automatically by Almaden at predetermined intervals.

It should be noted that we are not using the DB2 XML Extender MQSeries functions (for example, `MQSendXML()`) to send the XML data. Our decision to use the MQSeries functions is based on the fact that we have a need for high performance and low latency in sending XML data/fragment to Almaden clients. And, the native SQL functions are inherently more efficient than using XML Extender functions. This is due, in part, to the fact that the XML Extender functions require an additional processing layer to perform SQL queries. Whereas, the MQSeries functions bypass this extra processing layer in performing SQL queries. Please see Figure 5-3 for an architecture diagram that highlights the components and information flow for alternative three.

![Figure 5-3 Alternative 3 flow](image-url)
5.4 Step-by-step implementation

This section specifies the different steps to go through to implement each XML alternative.

5.4.1 Alternative one

This alternative will allow an Almaden client to use a Web browser and submit an XML document, via a Java servlet, to HLED Investments. The XML document is then shredded (broken apart) and the XML document elements are stored in appropriate fields of a database table.

A summary of the implementation steps for this alternative is given in Table 5-2.

Table 5-2 Alternative one: implementation steps

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Performed by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Enable the database with XML Extender.</td>
<td>Database Administrator, Database Developer</td>
</tr>
<tr>
<td>2</td>
<td>Generate the XML transaction request document.</td>
<td>Database Developer</td>
</tr>
<tr>
<td>3</td>
<td>Run a Java program to store the XML document in a DB2 UDB table.</td>
<td>Database Administrator, Database Developer</td>
</tr>
</tbody>
</table>

The following step-by-step procedure and DB2 XML Extender functions will be used to store XML documents in DB2 UDB. These documents will be stored in different fields of the Stock_MutualFund_Info table.

The Stock_MutualFund_Info.DAD file will be used for relational database (RDB) node mapping of the XML document elements to fields of the Stock_MutualFund_Info table. In addition, the XML document will be validated, with the stock_mutualFund_info.dtd file, prior to processing by DB2 UDB. It should be noted that the Stock_MutualFund_Info.DAD file is being reused from Business Case II (see Table 4-7 on page 146).

Each Web browser transaction request is converted to XML and submitted to DB2 UDB by a Java application along with the DAD file and target database name.

Step 1: Enable the database with XML Extender
To enable the database with XML Extender, use a DB2 UDB window to issue the command in Example 5-1 and verify that no errors are returned.
Example 5-1  Enable_db command

dxxadm enable_db IIIBOOKDB

After this command has completed, the output in Figure 5-4 is displayed to show completion status.

Figure 5-4  Enable_db command

In addition, the new XML Extender stored procedures, contained in the DB2XML schema, are cataloged in the database. Please see Figure 5-5 for the stored procedures that are cataloged after the enable_db command has been issued.

Figure 5-5  Database stored procedures after enable_db
Step 2: Generate the XML transaction request document

- A Java servlet (please see Appendix F, “Additional material” on page 351) is used to transform the Web browser request (in Figure 4-15) to an XML document. The generated XML document will look similar to the output in Figure 5-6.

![ClientTransaction.xml](image1)

Figure 5-6  ClientTransaction.xml

- The associated stock_mutualFund_info.dtd file contents are displayed in Figure 5-7.

![Stock_MutualFund_info.dtd](image2)

Figure 5-7  Stock_MutualFund_info.dtd

- The previously created Stock_MutualFund_Info.DAD file contents are displayed in Figure 5-8.
Step 3: Run a Java program: store the XML document

A Java program is used to store the XML document in a DB2 UDB table.

- Use the operating system command shell to issue the command in Example 5-2 and verify that no errors are returned.
Example 5-2  Shred Java Command

```
java Shred IIRED0KDB stock_mutualfund_info.dad ClientTransaction.xml
```

After the Shred command has completed, the output in Figure 5-9 is displayed to show completion status. The Shred Java program uses the database DB2XML.dxxShredXML stored procedure, listed in Figure 5-9 to decompose (or shred) the XML document.

```
C:\Alt 1>java Shred IIRED0KDB stock_mutualfund_info.dad ClientTransaction.xml
DAD size = 2384
XML size = 361

Java Shred Stored Procedure Sample
Connect to IIRED0KII
Calling stored procedure : db2xml.dxxShredXML
errCode = 0
msgText = 0800251 XML decomposed successfully.
C:\Alt 1>
```

Figure 5-9  Shred command

Table 5-3 shows a mapping of XML elements contained in the ClientTransaction.xml file to fields in the Stock_MutualFund_Info table.

<table>
<thead>
<tr>
<th>Client Transaction File XML Attributes and Elements</th>
<th>Stock_MutualFund_Info Table Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>INFO_KEY</td>
</tr>
<tr>
<td>Symbol</td>
<td>SYMBOL</td>
</tr>
<tr>
<td>Description</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>Type</td>
<td>TYPE</td>
</tr>
<tr>
<td>Sector</td>
<td>SECTOR</td>
</tr>
<tr>
<td>Price</td>
<td>PRICE</td>
</tr>
<tr>
<td>Market</td>
<td>MARKET</td>
</tr>
</tbody>
</table>
In addition, the updated STOCK_MUTUALFUND_INFO table is displayed in Figure 5-10.

5.4.2 Alternative two

This alternative will use an SQL/XML query to create XML data/fragment containing customer record(s). Given that the SQL/XML functions are included in the SQL specification, a DB2 UDB database does not need to be enabled. This is in contrast to requirement in alternative one that the database must be enabled prior to using the XML Extender functions.

A summary of the implementation steps for this alternative are listed as in Table 5-4.

<table>
<thead>
<tr>
<th>Steps</th>
<th>Description</th>
<th>Performed By</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Issue a DB2 SQL/XML query to generate an XML document containing customer record(s)</td>
<td>Database Administrator, Database Developer</td>
</tr>
</tbody>
</table>

**Step 1: DB2 SQL/XML query to generate XML data**

There is only one step that is to issue a DB2 SQL/XML query to generate XML data containing customer record(s).

The database table in Figure 5-11, MUTUALFUNDPORTFOLIO, will be used as a data source for DB2 SQL/XML query that is used to generate the XML data.
The DB2 SQL/XML query in Example 5-3 uses the XML2CLOB function to generate XML data (please see Figure 5-12) containing elements (tags) created by the XMLELEMENT function for two customer records from the MUTUALFUNDPORTFOLIO table. The XML2CLOB function is used to return an argument as an XML character large object. In Example 5-3, the argument for the XML2CLOB function is the composed XML document. The CLOB variable was chosen to hold the XML data, because the size of XML data/fragment is unknown before it is generated. The maximum size of two gigabytes available with a CLOB variable is well suited to store most generated XML data. The XMLELEMENT function is used to create an XML element, in the XML data, for each listed field of the MUTUALFUNDPORTFOLIO table.

**Example 5-3 SQL/XML query**

```sql
SELECT XML2CLOB(XMLELEMENT(NAME "Customer", XMLEMENT(NAME "Customer_ID", c.customerid), XMLEMENT(NAME "M_F_Symbol", c.mutual_fund_symbol), XMLEMENT(NAME "Quantity", c.quantity), XMLEMENT(NAME "F_Name", c.firstname), XMLEMENT(NAME "Initial", c.mi), XMLEMENT(NAME "L_Name", c.lastname), XMLEMENT(NAME "Address1", c.streetadr1), XMLEMENT(NAME "Address2", c.streetadr2), XMLEMENT(NAME "Address3", c.streetadr3), XMLEMENT(NAME "City", c.city), XMLEMENT(NAME "State", c.state), XMLEMENT(NAME "Zip", c.zip)) FROM ADMINISTRATOR.MUTUALFUNDPORTFOLIO c WHERE c.customerid='1001' OR c.customerid='1002'
```

The XML data is generated as shown in Figure 5-12 without XML header.
It should be noted that no whitespaces exist between the tags in this XML document, however, whitespaces were added for clarity.

If this XML fragment is to be successfully parsed by an XML parser, it needs to be modified, by using a Java program or other tool, to create a complete (well-formed) XML document. A header label, as in Example 5-4, must be added at the beginning of the SQL/XML generated XML fragment.

**Example 5-4  XML header.**

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

The complete well-formed XML document is now displayed (please see Figure 5-13) in the Microsoft Internet browser.
5.4.3 Alternative three

The DB2 XML Extender MQSeries function, MQSend(), will be used to send an XML document, created using the same process as alternative two, to a remote WebSphere MQ queue that resides at an Almaden client's location. It should be noted that both XML Extender and SQL/XML DB2 UDB functions are used in this alternative.

A summary of the implementation steps for this alternative are listed in Table 5-5.
Table 5-5  Alternative three: implementation steps

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Performed By</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Enable basic MQSeries functions in the database.</td>
<td>Database Administrator, Database Developer</td>
</tr>
<tr>
<td>2</td>
<td>Start the MQSeries AMI administration tool.</td>
<td>MQSeries Administrator</td>
</tr>
<tr>
<td>3</td>
<td>Use the AMI administration tool to create a service point and policy for the client’s remote WebSphere MQ queue.</td>
<td>MQSeries Administrator</td>
</tr>
<tr>
<td>4</td>
<td>Create a stored procedure in the database to send an XML document to the client’s queue.</td>
<td>Database Administrator, Database Developer</td>
</tr>
<tr>
<td>5</td>
<td>Call the stored procedure.</td>
<td>Database Administrator, Database Developer</td>
</tr>
</tbody>
</table>

**Step 1: Enable basic MQSeries functions in the database**

Use a DB2 UDB window to issue the command in Example 5-5 and verify that no errors are returned.

**Example 5-5  Enable_MQFunctions Command**

```
enable_MQFunctions -n IIREDBO0DB -u db2admin -p db2admin
```

After this command has completed, the output in Figure 5-14 is displayed to show completion status.

![Enable_MQFunctions command output](image.png)

**Figure 5-14  Enable_MQFunctions command**

Please see Figure 5-15 for the stored procedures that are cataloged after the `enable_MQFunctions` command has been issued.
Step 2: Start the MQSeries AMI administration tool.

In a Microsoft Windows environment, use the Start menu as in Example 5-6.

Example 5-6  Starting the MQSeries AMI GUI

Start-Programs-IBM MQSeries AMI-IBM MQSeries AMI Administration Tool

Step 3: Use the AMI administration tool

Next step is to use the AMI administration tool to create a service point and policy for the client's remote WebSphere MQ queue.

- The AMI GUI can be used to configure a service point as in Figure 5-16.
Chapter 5. Business case III: Incorporating XML data

Figure 5-16  AMI service point configuration

- The AMI GUI is also used to configure a policy as in Figure 5-17.

Figure 5-17  AMI policy configuration

Step 4: Stored procedure to send an XML document to MQ

Next step is to create a stored procedure in the database to send an XML document to the client's remote WebSphere MQ queue.

- Please refer to 3.4.12, “Develop the GETPORTFOLIO stored procedure” on page 104 for the detailed steps in creating a DB2 UDB stored procedure.
The Java stored procedure in Figure 5-18 was created with the DB2 UDB Development Center tool:

```
VALUES DB2MQ.MQSEND (ACUCLIENT1_SP, ACUCLIENT1_POLICY, (SELECT XML2CLOB(XMLELEMENT(NAME "Customer", XMLELEMENT(NAME "Customer_ID", c.customerid), XMLELEMENT(NAME "M_F_Symbol", c.mutual_fund_symbol), XMLELEMENT(NAME "Quantity", c.quantity), XMLELEMENT(NAME "First_Name", c.firstname), XMLELEMENT(NAME "Initial", c.mi), XMLELEMENT(NAME "Last_Name", c.lastname), XMLELEMENT(NAME "Address1", c.streetadr1), XMLELEMENT(NAME...
```

The actual SQL statement for dynamically generating and sending the XML document to a client's remote WebSphere MQ queue is displayed in Example 5-7. Basically, the SQL/XML query issued in Example 5-3 is enclosed as an argument for the DB2MQ.MQSend() function. The service point and policy arguments of the DB2MQ.MQSend() function are used to reference a WebSphere MQ destination queue and connection details. In Example 5-7, the ACUCLIENT1_SP service point and the ACUCLIENT1_POLICY policy arguments are defined for a remote WebSphere MQ queue that is located at an Almaden client's site.

**Example 5-7  MQSend() Query**

```
VALUES DB2MQ.MQSEND (ACUCLIENT1_SP, ACUCLIENT1_POLICY, (SELECT XML2CLOB(XMLELEMENT(NAME "Customer", XMLELEMENT(NAME "Customer_ID", c.customerid), XMLELEMENT(NAME "M_F_Symbol", c.mutual_fund_symbol), XMLELEMENT(NAME "Quantity", c.quantity), XMLELEMENT(NAME "First_Name", c.firstname), XMLELEMENT(NAME "Initial", c.mi), XMLELEMENT(NAME "Last_Name", c.lastname), XMLELEMENT(NAME "Address1", c.streetadr1), XMLELEMENT(NAME...
```

212  Getting Started on Integrating Your Information
"Address2", c.streetadr2), XMLELEMENT(NAME "Address3", c.streetadr3),
XMLELEMENT(NAME "City", c.city), XMLELEMENT(NAME "State", c.state),
XMLELEMENT(NAME "Zip", c.zip)) FROM ADMINISTRATOR.MUTUALFUNDPORTFOLIO c WHERE
  c.customerid="1001" OR c.customerid="1002")

Step 5: Call the stored procedure.
The stored procedure, displayed in Figure 5-19, is called by the Java servlet after
a client issues a Web browser request or automatically by Almaden at
pre-determined intervals. Upon completion of the Java stored procedure, the
XML document is placed on a remote WebSphere MQ queue that is maintained
and located at an Almaden client’s site. The Almaden client’s WebSphere MQ
queue contents for the generated XML message is displayed in Figure 5-19.

Figure 5-19 SQL-XML message on a client WebSphere MQ queue
5.5 Integration considerations

The technology alternatives presented in this chapter will be most useful to database/integration architects, designers, developers, administrators and others having a background with DB2 UDB, WebSphere MQ, Web Services and XML or similar technologies.

In regard to the traditional business integration model, the solution alternatives will employ the database, application and portal integration types. In addition, RPC, message-oriented and database-oriented middleware will be used to initiate synchronous, asynchronous, fire and forget, request response, direct and queued communication between the information integration framework and external applications.

The industry standard secure sockets layer (SSL), version 3.0, will be used to provide digital certificates and 128-bit encryption for all investment transactions submitted by Almaden clients.

5.6 Conclusion

The combination of XML, database and messaging technologies provides a powerful and compelling option that can be applied to a variety of business problem domains. In this chapter, we have provided a gentle introduction to the IBM DB2 UDB implementation of a combined offering. Building on the business scenario outlined in Chapter 4, "Business case II: Accessing your information through Web Services" on page 119, we presented three alternatives to highlight the DB2 UDB XML and messaging technologies.

The first alternative used DB2 XML Extender to show the process for storing an XML document in DB2 UDB as an XML collection. The second alternative used the ISO standard SQL/XML extension functions to compose an XML document that was written to the file system. In the third alternative, we placed the composed XML document, created in the second alternative, on a remote WebSphere MQ queue that is maintained at a client location. We encourage further exploration of these technologies, by all business entities, in addressing current business problems that have an XML, messaging, and/or database requirement.
Chapter 6. Business Case IV: Populating a Common Data Store

At the heart of information integration lies the idea of integrating information from diverse data sources into a single consolidated view.

As described in the previous chapters, we can achieve this by leveraging data federation technology. This technology allows us to build a virtual database in which data from diverse data sources can be integrated. In this way we can integrate information across distributed and disparate data sources.

Some business requirements (data warehouses, other specific ones) may need to materialize a consolidated view of diverse data sources into a dedicated database.

In this chapter we focus on a scenario where we integrate information from distributed and disparate data sources into a single data base, enabling the data federation capabilities for accessing the data. We call this single data base Common Data Store as it represents a subject oriented, consolidated view across diverse data sources.
6.1 Business problem statement

HLED Investment wants to extend its customer reach and service through multiple services channels.

Today customer service is mainly provided by a call center application. This allows customers to check their investment portfolios, their recent investment transaction and to place a buy and/or sell order for stocks and mutual funds.

In the future HLED Investment wants to expand its call center services and make most of these services also available via Web Services and a new HLED Investment portal. For the call center service HLED Investment plans to enrich customer data with purchased demographic data.

Today HLED Investment stores customer related data in several legacy systems. In order to get a consolidated view of a customer across the enterprise HLED Investment has to assemble information from disparate systems. This is done by their current call center application that retrieves data from multiple data sources (see Figure 6-1) and reconciliates them.

For the new applications HLED Investment demands that a consolidated view of all customer related information must be available and that this view must be consistent over all three "channels". Furthermore, that the new applications will not impose any security risk or performance degradation on their trading systems.

![Figure 6-1 Collect and assemble](image-url)
6.2 Assumptions

Assumptions regarding the new call center application are as follows:

- Today customer related information is stored either in relational database systems (DB2 UDB and Informix) or flat files (VSAM files).
- External demographic data will be used to enrich customer related information. This demographic data is delivered as flat file.
- The call center and portal applications are Web based applications (WebSphere); DB2 Web Services infrastructure can be used for implementing Web Services.
- All customer service applications (call center, HLED Investment portal and Web Services) will use the same infrastructure components (Application server, customer related data) to provide services to customer.
- Operating hours for the Customer Services Applications are between 5 a.m. - 12:00 p.m.
- The customer related Information accessible to the customer services applications reflects status of the day before.

6.3 Solution

The core of the proposed solution is to set up a Common Data Store (CDS) as depicted in Figure 6-2, in which all customer related information will be integrated and consolidated. The concept requires that all customer related information is accessed, extracted from the various data sources, and loaded into a single database that HLED Investments’ customer service applications will have access to.

Figure 6-2   Common Data Store
The CDS solution will replace the current approach of integrating information (collect and assemble) by the application itself by an approach that leverages data management technology. This allows to push the information integration effort from the application layer down to the data management layer (see Figure 6-3).

![Figure 6-3 Pushing information integration effort to data management layer](image)

The integration effort at the data management layer is accomplished when data gets accessed from the various data sources into the Common Data Store. The entire procedure of data extraction, transfer, and load into the target object will be handled by a data population process. The process will also include steps that mask any differences in data structure and semantics. In this way the data populating process is an information integration layer when accessing source systems to populate the target Common Data Store.

The data copy process will run once a day. Because of moderate data volume and high network bandwidth a full refresh for all target tables is possible.

The proposed solution addresses HLED Investments requirements in the following way:

- All customer service applications have access to the same customer related information. The provided customer data is consistent regardless from which channel the information is requested.
- A dedicated system can be used for the Common Data Store. In this way the impact on the operational systems can be minimized.
Web based customer service applications like HLED Investment Portal or Web Service do not need to access the operational data. This reduces the security risk on that systems.

Furthermore, introducing a Common Data Store reduces application complexity by simplifying the data access flow. Instead of dealing with different data sources on diverse system platforms all required information can be found in a single database system accessible through standard SQL interface. As depicted in Figure 6-4 coding becomes less complex and - to a certain extend - the application gets shielded from changes in the underlying data sources. In this way the maintenance effort for the application will be reduced.
6.4 Infrastructure and technology solution

Implementing a Common Data Store solution requires the setup of a dedicated data store and an infrastructure that feeds data into this data store.

This section gives a brief overview of the architecture that we chose for the CDS scenario. The main architectural component of this solution, the data population infrastructure, is discussed in more detail under 6.4.3, “Data population subsystem” on page 224.

The system environment that were used for the CDS scenario is described in 2.4.4, “Technical Infrastructure: Business Case IV” on page 73.

Note: For this scenario we used only a subset of the technology that is available with Data Warehouse Center today. The focus here was not to demonstrate all Data Warehouse Center capabilities. Instead we want to demonstrate on a simplified example, how Data Warehouse Center can be used as the information integration layer between source systems and target Common Data Store.

6.4.1 Architecture overview

Figure 6-5 depicts the architecture that was chosen for the CDS solution. On this high level diagram we can identify the Operational Systems, the Common Data Store and the Data Warehouse Center as the major building block of these architecture.
The idea of a Common Data Store as a place where all subject oriented data from diverse sources can be concentrated is represented by the arrows pointing from the data sources via the Data Warehouse Center into the Common Data Store (rectangle Common Data Store).

The Data Warehouse Center in the diagram represents the information integration layer for the Common Data Store in this architecture. It is the place/layer were data from diverse sources get accessed and differences in data structure/semantics are compensated or masked. The integration effort to produce a single, consolidated, and consistent view of all subject orientate data across the different data sources takes place at this layer. This layer is implemented as “Data Populating Subsystem” in our solution. It is used for handling the entire source-to-target data transfer process. This includes the following process steps:

- Data extraction on the source and transfer to the target system (move)
- Transform source data to adjust it to target structure/semantics (transformation)
- Data load/insert into the target source (load)
The Data Warehouse Center was chosen as the foundation for the Data Populating Subsystem because it offers a open framework that:

- Supports data extraction from a wide range of data sources like relational databases, OLAP Servers, and flat files
- Supports several data transfer options, including plain "ftp" for flat files, direct SELECT/INSERT, message queuing, and data replication
- Comes with a set of pre-built transformation routines and let you build transformation routines using SQL or Stored Procedures with the DB2 Development Center
- Includes features to define, test, deploy, and monitor the entire process of source-to-target data transfer.

For the source-to-target data transfer process Data Warehouse Center supports different types of data acquisition (extract, transfer, transform, load). For source systems with relational databases we chose federated access (depicted by the arrows labeled “Full Refresh” in the diagram). It allows to extract and transfer data from a source to a target object with a single standard SQL-Statement (SELECT/INSERT).

A different type of source-to-target data transfer was selected for the customer data on VSAM. Here we used WebSphere MQ as a data broker for managing the data transfer between source and target system. On the sender side of the queue (source) system a CICS transaction is used to extract data from the VSAM file and place it into the queue. On the receiver side, DB2 UDFs (User Defined Functions) were used by the data populating process to retrieve data from the queue.

### 6.4.2 Information integration

On the Common Data Store we implemented a data model which represents a consistent, consolidated view of the data that we integrated from the various data sources. In cases where source and target differs either in structure, semantics, or both we used transformation routines to compensate or mask that difference.

Figure 6-6 gives a high level overview of the source-to-target mapping that were used.
Mutual Fund and Stock Portfolio
For the Mutual Fund Portfolio and Stock Portfolio table definition on CDS we used the structure and semantics of source tables (please refer to 2.3, "Information integration at HLED Investments" on page 65 for a detailed description of the table structure).

Mutual Fund Orders and Stock Orders
The source tables Mutual Fund Orders and Stock Orders were consolidated into a single table Customer Orders in the CDS data model. The layout chosen for the Customer Orders table differs from the source tables for the following three columns:

- TRANSACTION_DAY: In target table the column is defined with data type DATE. In source table Stock_Orders, however, the column is defined with data type CHAR.

- ORDER_SYMBOL: In the target table this column is defined with data type CHAR. In the source tables, however, the column is defined with data type INTEGER. Beside this, the representation of ORDER_SYMBOL in source and target is different. In the source tables a code is used to represent a certain order symbol, whereas as textual representation is used in the target table.
ORDER_TYPE: This column is used to distinguish between Mutual Fund and Stock Orders records in the Customer_Orders table. It has no equivalent in the source tables.

To handle the differences in source and target table structure the following transformation steps are included in the data population process for table Customer_Orders:

- CHAR to DATE type conversion: DB2 built-in functions are used when the data gets extracted from Stock_Orders.
- ORDER_SYMBOL decoding: A CASE clause in the extracting SQL statement is used to decode from integer code to textual representation.
- Generated column value: The extracting Select statement is used to generate an appropriate value for the target column ORDER_TYPE.

Customer VSAM record
The definition for table Customer is based on the record structure that is used in the VSAM data set on the mainframe (Appendix “C” on page 345). For the table columns, we selected CHARACTER data type.

Demographic data
For the external flat file sources, a table called Demographic was defined in the CDS.

6.4.3 Data population subsystem
Data transfer from the source systems to the target system is crucial for the Common Data Store concept. We use the term “Data Population Subsystem” for the infrastructure - and it components - that were used to set up the data populating process for the Common Data Store solution.

This section details the implementation of our solution. We start with an overview of the Data Population Subsystem and the setup of its infrastructure components. This is followed by a description of the data acquisition types we used and the process we defined in the Data Warehouse Center Process Model.

Data populating subsystem: overview
The data populating system we implemented for the CDS scenario covers the various aspects of a data populating process:

- Connectivity between source/target (SQL, connector, adapter, extract utilities)
- Transport mechanism to transfer data from A to B.
- Extract, transfer, transform, and load data
- Process handler that let you automate the necessary process steps (define, test, deploy, monitor)

**Infrastructure implementation**

The setup of the Data Population Subsystem includes the implementation of the software that forms the Data Population Subsystem infrastructure. The following gives a brief overview of the setup infrastructure components. This includes Data Warehouse Center, WebSphere MQ, and federated access connectivity.

**Data Warehouse Center implementation**

The Data Warehouse Center was setup in its default configuration. In this configuration type all Data Warehouse Center components (data warehouse server, warehouse agent, and warehouse control database) are installed on the same server. This configuration offers the best performance when the source and the target reside on different systems (for a detailed discussion of the various configuration types, please refer to *DB2 Data Warehouse Center Administration Guide, SC27-1123-00*).

The setup/configuration we chose is depicted in Figure 6-7. For the implementation we decided to have a dedicated DB2 UDB instance for the Data Warehouse Center components and the Common Data Store environment (depicted with the label DB2 and CDS in the diagram).

![Diagram of Warehouse Server implementation](image)

**Figure 6-7  Warehouse Server implementation**
For data transfer between relational sources and the target system the data populating process required to set up federated access connectivity between the systems.

For data transfer between mainframe system (customer data on CICS/VSAM) and target, we set up WebSphere MQ (for a detailed description of the required steps, please refer to 6.5, “Step-by-step implementation” on page 231).

**Federated access connectivity**
For data transfer between relational sources and the target system the data populating process uses DB2 federated access over DB2 UDB V8.1 wrappers.

**Note:** Wrappers are mechanisms by which the federated server interacts with data sources. The federated server uses routines stored in a library called a wrapper module to implement a wrapper. These routines allow the federated server to perform operations such as connecting to a data source and retrieving data from it iteratively.

Typically, the DB2 federated instance owner uses the CREATE WRAPPER statement to register a wrapper in the federated system.

The DB2 UDB V8.1 wrappers allow the Data Warehouse Center to access tables objects that are located on the DB2 UDB AIX (Mutual Fund Data) and Informix (Stock Data) by means of simple SQL. Data extraction from source tables and data transfer to the target tables can be completed within a single SELECT/INSERT statement.

The process to set up the required connectivity is straightforward and described in detail in “Step-by-step implementation” on page 231.

**WebSphere MQ**
In our data populating subsystem message queuing is one of the transport mechanism we use for source-to-target data transfers between the mainframe source (CICS/VSAM) and the CDS target. On the receiver site of this transfer DB2 UDFs are used to receive data from a queue.

This infrastructure requires the following components:

- WebSphere MQ server on source and target
- MQSeries AMI on target
- WebSphere MQ enabled DB2 UDB target

As depicted in Figure 6-8, WebSphere MQ server were setup on the source and target system.

On top of this we needed on the target site:
CICS transaction that extracts all customer data from the VSAM file and put it into the remote queue DB2MQ_CV_Q().

**Note:** The program to implement is similar to Business Case I (see the definition of this source and the sample data shown in Appendix “C” on page 345)

On the target side we need:
- MQSeries AMI
- MQSeries Integration

MQSeries AMI on the target side helps to simply the MQ queue handling for DB2 UDFs. Through the concept of service points and policies DB2 UDFs that want access a queue - other then the default queue - just need to know by which services and policies this certain resources is controlled.

The MQSeries Integration is basically a set of DB2 User Defined Functions (UDFs) that provide seamless access to MQSeries messaging via standard SQL statements. Using this UDFs within a Data Warehouse Center SQL process steps allows us to directly receive data from a queue and put it into a target table.

---

**Data acquisition**

We use the term *data acquisition* for a process that includes the steps extract, transfer, transform and load.
Data Warehouse Center offers various ways of set up data acquisition for populating a target system. This ranges from simple unload-to-file/transfer/load to the use of sophisticated data replication infrastructure.

What type of data acquisition is to implement is mainly determined by the following requirements:

- Refresh frequency
- Data volume to transfer
- Infrastructure (bandwidth, technology available)

**Data acquisition considerations**
Beside an initial load/transfer of data from the source to the target we also need a frequent update of the CDS to reflect changes that were made to the source data. By moving data from the source to target we can synchronize a Common Data Store with its operational data sources. We get a snapshot or point-in time copy of the operational data sources, if all changes - reflecting a certain point in time - are moved from source to target.

The frequency of doing this “refresh” of the CDS determines how current the a CDS copy becomes. As we will see further done in this section, this can have a major influence on the type of data acquisition we need to implement. Beside refresh cycles, the type of data refresh is worth to consider. Basically we have here two options: The first is a full refresh, the second to propagate only changes in the source data to its target. This too influences what type of data acquisition we have to select. A third thing to consider are differences in structure and semantics between source and target. To cope with this so-called transformers can be applied to that data during the refresh process.

**Data acquisition types implemented**
For our scenario we made the assumption that the data volume that needs to be transferred to the target is moderate, and that a refresh cycle everyday would be sufficient to meet the requirements of customer service applications. Therefore, we decided to implement the data population process as a daily full refresh. For data acquisition we select the following methods:

- Data Warehouse Center built-in load for flat files
- Federated access over DB2 UDB V8.1 wrappers
- Message queueing to propagate changes from source to target

**Load from flat files**
Data Warehouse Center offers various options for loading local and remote flat files. Different file types, access methods, and data transfer protocols are supported (for a complete description, please refer to *DB2 Data Warehouse Center Administration Guide, SC27-1123-00*).
As depicted in Figure 6-9 we selected Data Warehouse Center built in “Remote files” load program to move the flat file Demographic into the target table (Assumption in the scenario: flat file is received from external sources).

![Figure 6-9 Data Acquisition - Load from flat file](image)

**Federated access using DB2 UDB V8.1 wrappers**

This method lets you define an SQL step in the Data Warehouse Center Process Model that selects the source data and insert it into a target table. Furthermore, you can use the SQL step to define some basis transformations. In this way the extract, transfer, transform, and load can be accomplished in one step.

For the Mutual Fund and Stock Portfolio data sources on DB2 UDB for AIX and Informix respectively, we decided to use the wrappers provided by DB2 UDB V8.1 to federate the access. (see Figure 6-10).

![Figure 6-10 Data acquisition: federated access over wrappers](image)
Message queuing
WebSphere MQ can be used as a high level transport mechanism to asynchronously transfer data from source to target. In our scenario we use this mechanism to replicate customer data from the VSAM source files into the target CDS.

On the source system (z/OS) a CICS program extracts the data from the VSAM files and put it into a MQSeries queue. On the target system we use an SQL Table Function and a MQSeries Integration UDF to define a virtual view on that queue. In this way the queue appears as a source view to the populating process. This view is accessed by a Data Warehouse Center process step that selects the data from the view and inserts it into the target table Customer (see Figure 6-11).

Process model
In order to transfer the information from our source system into the Common Data Store we defined a Data Warehouse Center process, which contains a series of process steps. Each of these steps defines how the source data is to be transformed and moved into the target data.

For the scenario we defined a Data Warehouse Center process with the following seven process steps:

- Stock Portfolio - transfer
- Mutual Fund - transfer/transform
- Mutual Fund Orders - transfer/transform
- Stock Orders - transfer/transform
- Customer Data - transfer
- Demographic Data - transfer
- Load Demographics Data
For the Stock Portfolio we defined an SQL process step that uses federated access over a wrapper to select data from source and insert it directly into target table.

For the Mutual Fund Portfolio we used the same kind of SQL process and federated access. The SQL statement used, however, was enhanced so that we could compensate and mask some differences between the source and the target table (see 6.5, “Step-by-step implementation” on page 231 for a detailed description of the SQL statement we used).

In order to consolidate the source tables Mutual Fund Orders (MFO) and Stock Orders (SO) into a single target table (Customer Orders), we defined two separate SQL process steps each using the same target table. For the MFO process step the processing option “REPLACE” is defined; for process step SO we defined “APPEND”. In the process flow we let MFO execute before the SO process step - so MFO process rebuild Customer table from scratch, whereas SO process steps appends inserted records. The necessary transformations for these process steps are carried out with SQL. The SQL statements we use for the process step was enhanced to handle the following transformations:

- Type conversion (for example CHAR to DATE type conversion)
- Decoding (for example, CASE clause is used in the extracting SQL statement to decode values for columns SYMBOL_CODE from integer code to textual representation).
- Compensation (for example, extracting through select statement is used to generate an appropriate value for the target column ORDER_TYPE)

For customer data we defined an SQL process step that access data from a local MQSeries queue and inserts it into the target table. In order to make this happen the local MQSeries queue had to be defined as a (virtual) local view first. To set up a MQSeries queue as virtual view source to Data Warehouse Center, we created a DB2 table function that uses a MQSeries Integration UDF to access the queue.

In order to move the Demographic from flat file into the target table we defined two program steps. The first transfers the data to the CDS server and the second loads it into the target table.

### 6.5 Step-by-step implementation

This section details the features and product components that were configured to implement the solution. Our implementation steps are covered in detail below, and summarized in Table 6-1. In this table are specified the skills required:
To set up the basic infrastructure of this scenario, as a prerequisite, all software that is listed under Section 6.4, “Infrastructure and technology solution” on page 220 must be installed. A complete description of the planning and implementation of the software can be found in DB2 Quick Beginnings for DB2 Servers, GC09-4836, and DB2 Data Warehouse Center Administration Guide, SC27-1123-00.

### Table 6-1 Common Data Store setup: basic steps

<table>
<thead>
<tr>
<th>Steps</th>
<th>Who</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DB2 DBA</td>
<td>CDS server initialization</td>
</tr>
<tr>
<td>2</td>
<td>DB2 DBA</td>
<td>Install MQSeries Integration</td>
</tr>
<tr>
<td>3</td>
<td>DB2 DBA/Informix DBA</td>
<td>Setup federated access to sources: DB2 UDB for AIX and INFORMIX</td>
</tr>
<tr>
<td>5</td>
<td>MQ Admin</td>
<td>WebSphere MQ implementation</td>
</tr>
<tr>
<td>6</td>
<td>MQ Admin/DB2 DBA</td>
<td>MQSeries AMI implementation</td>
</tr>
<tr>
<td>7</td>
<td>DB2 DBA/DWC Admin</td>
<td>Data Warehouse Center implementation</td>
</tr>
<tr>
<td>8</td>
<td>DWC Admin</td>
<td>Defining Common Data Store sources</td>
</tr>
<tr>
<td>9</td>
<td>DWC Admin</td>
<td>Defining Common Data Store targets</td>
</tr>
<tr>
<td>10</td>
<td>DWC Admin</td>
<td>Defining data transformation and movement</td>
</tr>
<tr>
<td>11</td>
<td>DB2 DBA/DWC Admin</td>
<td>Process flow and scheduling</td>
</tr>
</tbody>
</table>

### 6.5.1 Step 1: CDS server initialization

To initialize the CDS server, three tasks have to be done:

- Create a DB2 UDB instance
- Initialize the warehouse control database for metadata
Catalog the Common Data Store as an ODBC source

Task 1 - Create the CDS instance
Our first task is to create a new DB2 UDB instance for our CDS environment on server C55VK69R. This is followed by the creation of the Common Data Store database CDS_DB. The environment variable DB2DEFAULT is set to the new created instance CDS (see Example 6-1)

Example 6-1  CDS Instance setup commands

db2icrt CDS -s ese -u db2admin,db2admin

db2 create database CDS_DB on C:\

set DB2INSTANCE=CDS

Task 2 - Initialize the Warehouse Control database
For our scenario we decided to have a dedicated DB2 UDB instance for our Data Warehouse Center and not to use the default settings. Instance creation and Warehouse Control Database setup are described in the following.

Create DB2 UDB instance for the warehouse control database
This step (see Example 6-2) is necessary if the warehouse control database should reside in a dedicated DB2 UDB instance. Otherwise skip this step.

Example 6-2  Create Control Database instance

db2icrt DB2 -s ese -u db2admin,db2admin

Initialize the warehouse control database
The DB2 UDB feature Warehouse Control Database Management is used to initialize the warehouse control database. Before executing the program, ensure that the DB2INSTANCE environment variable is set to the appropriate instance (in our case the instance “DB2”).

To run the program, go to Start --> Programs--> IBM DB2 --> Set-up Tools --> Warehouse Control Database Management.
Figure 6-12  Initiate warehouse control database

The Data Warehouse Center - Control Database Management requires the following entries:

- In the **New control database** field the name of the database which the program is to create. We entered “CDS_MD”
- The schema name for the new warehouse control database. We accepted the default value “IWH”.
- In the **Password** field a userid that has privilege to create the new warehouse control database. We entered “db2admin” - which is the instance owner “DB2”.
- In the **Password** and **Verify password** field, the password of the entered userid.

**Task 3 - Catalog CDS database as ODBC source**

In order to make the Common Data Store on instance CDS accessible to the Data Warehouse Center we had to catalog the Common Data Store as a remote database to instance DB2. Furthermore, we had to bind the Common Data Store database as ODBC source - see Example 6-3.

**Example 6-3  Catalog CDS as remote database**

```
SET DB2INSTANCE=DB2
```
6.5.2 Step 2: install MQSeries Integration

The MQSeries support in DB2 UDB is activated by two programs which come with DB2 UDB. On the window platform the two programs `enable_MQFunctions` and `enable_MQXMLFunctions`, are located in the "...\SQLLIB\bin" directory.

The MQ/MQXML functions were implemented by executing both programs against the CDS database. The commandos we entered on the command prompt window are depicted in Example 6-4.

`Example 6-4  MQ/MQXL commands`
```
enable_MQFunctions -n CDS -u db2admin -p db2admin
enable_MQXMLFunctions - CDS -u db2admin -p db2admin
```

The command syntax for both programs is depicted in Example 6-5.

`Example 6-5  Command syntax`
```
enable_xxx -n <database name> -u <userid> -p <password>
```

6.5.3 Step 3: Set up federated access to remote sources

In our case the remote relational sources are DB2 UDB and Informix.

Set up federated access to DB2 UDB for AIX

The Mutual Fund data is located in a DB2 UDB database on server DB2 for AIX V8. In order to make this source accessible for the Data Warehouse Center, we had to set up federated access to it.

To establish the connectivity we had to complete the following tasks:

1. Gather DB2 UDB for AIX information
2. Enable the CDS server for federated access
3. Catalog the node.
4. Catalog the database.
5. Create the wrapper for DB2 UDB
6. Create the server
7. Create the user mapping
8. Create nicknames to access remote DB2 UDB tables
Task - 1 Gather DB2 UDB on AIX information
In order to set up connectivity to the remote instance DB2 UDB for AIX V8.1, we need to get some database specific information from remote system first (see Table 6-2).

### Table 6-2  DB2 UDB V8.1 server information

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Name</td>
<td>FRANKFURT</td>
</tr>
<tr>
<td>Instance Name</td>
<td>DB2AIX8</td>
</tr>
<tr>
<td>Database Name</td>
<td>MTRADE</td>
</tr>
<tr>
<td>Service Name</td>
<td>db2aix8c</td>
</tr>
<tr>
<td>Port Number</td>
<td>50000</td>
</tr>
</tbody>
</table>

Task 2 - Enable DB2 for federated access
To enable the DB2 UDB server to access federated data sources, we have to set the database manager FEDERATED parameter to Yes as in Example 6-6.

#### Example 6-6  Enable federated access

```
UPDATE DATABASE MANAGER CONFIGURATION USING FEDERATED YES
```

Task 3 - Catalog the remote node
On the CDS server the remote instance DB2AIX8 was cataloged as TCPIP node. The DB2 CATALOG TCP/IP NODE command used is depicted in Example 6-7.

#### Example 6-7  Catalog remote node

```
CATALOG TCP/IP NODE DB2AIX8 REMOTE FRANKFURT SERVER DB2AIX8C REMOTE_INSTANCE DB2AIX8
```

Task 4 - Catalog the DB2 UDB remote database
With the DB2 command CATALOG DATABASE the database MTRADE on the remote instance DB2AIX8 was cataloged as a remote database (see Example 6-8).

#### Example 6-8  Catalog remote database

```
CATALOG DATABASE MTRADE AT NODE DB2AIX8 AUTHENTICATION SERVER
```

Task 5 - Create the wrapper for DB2 UDB
For the data sources on our DB2 UDB for AIX server the standard DRDA wrapper was defined. The CREATE WRAPPER statement that we issued against the
CDS database is depicted in Example 6-9. This has to be done once, whatever the platform is.

**Example 6-9  Create wrapper for DB2 UDB for AIX**

```sql
CONNECT TO CDS_DB
CREATE WRAPPER DRDA
```

**Task 6 - Create the server to the DB2 UDB server**  
After creating a wrapper, the information on the remote DB2 UDB data source needs to be defined in the federated server database, using the `CREATE SERVER` statement, as depicted in Example 6-10. This has to be done for each different DB2 UDB server accessed.

**Example 6-10  Create server definition for DB2 UDB for AIX**

```sql
CONNECT TO CDS_DB
CREATE SERVER "DB2_FRANKFURT" TYPE DB2/UDB VERSION '8.1' WRAPPER "DRDA" AUTHID "db2aix8" PASSWORD "*******" OPTIONS ( DBNAME 'MTRADE')
```

**Task 7 - Create the user mapping**  
In our environment the user ID and password to access the federated server (CDS server) differs from the user ID and password to access the data source on the DB2 UDB for AIX server. In order to make the data source accessible to the federated server we have to define the association or mapping between the two servers. The `CREATE USER MAPPING` we issued is shown in Example 6-11.

**Example 6-11  Create user mapping CDS-DB2AIX**

```sql
CONNECT TO CDS_DB
CREATE USER MAPPING FOR "DB2ADMIN" SERVER "DB2_FRANKFURT" OPTIONS(
    REMOTE_AUTHID 'db2aix8', REMOTE_PASSWORD '*******')
```

**Task 8 - Create the nicknames to DB2 UDB tables to access**  
Making data source tables accessible over the federated server wrapper requires the definition of nicknames for those tables. The `CREATE NICKNAME` statements we issued for our *Mutual Fund* tables are shown in Example 6-12. A nickname has to be defined for each table to access.

**Example 6-12  Create nickname for Mutual Fund table**

```sql
CONNECT TO CDS_DB
CREATE NICKNAME "FEDTAB.MUTUAL_FUND_ORDERS" FOR "DB2_FRANKFURT"."MMI"."MUTUAL_FUND_ORDERS";
```
CREATE NICKNAME "FEDTAB"."MUTUALFUNDPORTFOLIO" FOR "DB2_FRANKFURT"."MMI"."MUTUALFUNDPORTFOLIO"

Set up federated access to Informix

To make the Stock data sources on the Informix database available to the populating process, we federated that source using the INFORMIX wrapper.

The information needed to establish the connectivity and to set up the federated access for our scenario are outlined in the following tasks:

To we had to complete the following tasks:
1. Get Informix server information
2. Install and configure Informix Client
3. Set up Windows registry for Informix
4. Set up Windows environment variables
5. Install IBM Relational Connect for Informix
6. Create the wrapper for Informix
7. Create the server
8. Create user mapping
9. Create nicknames to access Informix tables

Task 1 - Gather Informix server information

In order to get connected to our Informix database the following information about the server and instance needs to be gathered (see Table 6-3.)

Table 6-3  Informix server information

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Name</td>
<td>MUNICH</td>
</tr>
<tr>
<td>IP-Address</td>
<td>9.153.16.105</td>
</tr>
<tr>
<td>Instance Name</td>
<td>XPS</td>
</tr>
<tr>
<td>Database Name</td>
<td>STRADE</td>
</tr>
<tr>
<td>Service Name</td>
<td>8310</td>
</tr>
<tr>
<td>Protocol</td>
<td>olsoctcp</td>
</tr>
</tbody>
</table>

Task 2 - Install the Informix Client component

In order to access the remote Informix Server the IBM Informix Client code or more specifically Informix Connect component was installed on the CDS Server (IBM Informix Connect is part of the Informix Client SDK).

For complete instructions, please refer to the Informix Getting Started Manual.
Task 3 - Set up Windows Registry
After installing Informix Connect we used the Setnet32 program, which gets installed with Informix Connect, to set up the Windows registry entries.

To start the program:
- Click Start->Programs->IBM Informix Connect ->Setnet32.
  The IBM Informix Setnet32 window opens.
- In the IBM Informix Setnet32 Wizard click the Server Information tab; see Figure 6-13.

![IBM Informix Setnet32](image)

On the Server Information panel we entered:
- “XPS” in the IBM Informix Server field.
- The hostname of our Informix server “pcaw3.ismaning.de.ibm.com” in the HostName field.
- “olsoctcp” in the Protocolname field.
- “8310” in the Service Name field.
We checked the registry entries by enter “regedit” from the command prompt. The entries can be found in the following folder `HKEY_LOCAL_MACHINE-->SOFTWARE-->Informix-->SQLHOST`.

**Task 4 - Set up Windows environment variables**
On the CDS server the following system environment variables were manually added (Start-->Settings-->System-->Advanced).

- INFORMIXDIR=C:\Program Files\Informix\Connect\n- INFORMIXSERVER=XPS

**Task 5 - Install IBM Relational Connect for Informix**
For the federated access to Informix data sources the *IBM Relational Connect for Informix Data Sources* must be installed. Make sure that this component is installed, otherwise run the DB2 UDB setup to install it - for a detailed description of the installation procedure please refer to *IBM Federated Systems Guide, GC27-1224-00*.

**Task 6 - Create the wrapper for Informix**
For the data sources on our Informix server the standard Informix wrapper was defined. The `CREATE WRAPPER` statement we issued against the CDS database is depicted in Example 6-13. This has to be done once.

*Example 6-13  Create wrapper for Informix*

```
CONNECT TO CDS_DB
CREATE WRAPPER INFORMIX
```

**Task 7 - Create the server**
After creating a wrapper, the remote Informix data source accessed needs to be defined in the federated database, using the `CREATE SERVER` statement as depicted in Example 6-14. This has to be done for each different Informix server accessed.

*Example 6-14  Create server definition for Informix*

```
CONNECT TO CDS_DB
CREATE SERVER "INF_MUNICH" TYPE INFORMIX VERSION '9.3' WRAPPER "INFORMIX"
OPTIONS ( NODE 'XPS', DBNAME 'STRADE' )
```
Task 8 - Create the user mapping
In our environment the user ID and password to access the federated server (CDS server) differs from the user ID and password to access the data sources on the Informix server. In order to make the data sources accessible to the federated server we had to define an association between the two authorizations. The `CREATE USER MAPPING` issued is shown in Example 6-15.

Example 6-15   Create user mapping for CDS-Informix

```
CONNECT TO CDS_DB

CREATE USER MAPPING FOR "DB2ADMIN" SERVER "INF_MUNICH" OPTIONS( REMOTE_AUTHID 'dirker', REMOTE_PASSWORD '******')
```

Task 9 - Create the nicknames to access Informix tables

In order to make the Stock Portfolio tables accessible over the federated server, a nickname was defined for each table to be accessed. The `CREATE NICKNAME` statements issued are shown in Example 6-16.

Example 6-16   Create nickname for Stock Portfolio table

```
CONNECT TO CDS_DB

CREATE NICKNAME "FEDTAB."STOCK_ORDERS" FOR "INF_MUNICH"."INF".STOCK_ORDERS";
CREATE NICKNAME "FEDTAB."STOCKPORTFOLIO" FOR "INF_MUNICH"."INF".STOCKPORTFOLIO";
```

6.5.4 Step 4: WebSphere MQ implementation

For the scenario we installed WebSphere MQ V5.3 on the CDS server. A complete documentation on installing WebSphere MQ on various platforms can be found in the WebSphere MQ Quick Beginnings Version 5.3 standard documentation for the appropriate platform.

Once WebSphere MQ was installed two queue manager DB2MQ_CV_MQM and MQV1, were defined for this scenario. The required set up is described in the following two tasks.
Task 1 - Create MQ resources on the CDS Server
On the CDS server the MQSeries Explorer program were used to create the queue manager DB2MQ.CV_MQM. The configuration parameter we used are listed in Table 6-4

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Resource Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>QLOCAL</td>
<td>DB2MQ.CV_Q</td>
</tr>
<tr>
<td>CHANNEL RECEIVER</td>
<td>MQv1.TO.CDS</td>
</tr>
</tbody>
</table>

Task 2 - Create MQ resources on the OS/390 server
The queue manager definitions for MQV1 on the /390 were entered via the ISPF panels. A file of definitions that can be input via the CSQINP2 DD on the queue manager start up is listed in Appendix “C” on page 345. A list of the definitions is given in Table 6-5

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Resource Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>QREMOTE</td>
<td>DB2MQ.CV_Q</td>
</tr>
<tr>
<td>QLOCAL(XMITO)</td>
<td>MQV1.TO.CDS</td>
</tr>
<tr>
<td>CHANNEL SENDER</td>
<td>MQV1.TO.CDS</td>
</tr>
</tbody>
</table>

6.5.5 Step 5: MQSeries AMI Implementation
To use DB2/MQSeries integration features the MQSeries Application Messaging Interface (AMI) feature must be implemented. On DB2 UDB for Windows the AMI installation image can be found under the “\SQLLIB\cfg\mg" directory. AMI is installed by running the setup program found in this directory.

The AMI Administration Tool was used to define the AMI Service Point DB2MQ.CV.RECEIVE and AMI Policy DB2MQ.VC.POLICY. The required setup is described in the following tasks.
Task 1- Start the AMI Administration Tool

To start the AMI Administration Tool:

- Go to START -> Programs -> MQSeries AMI -> IBM MQSeries AMI Administration Tool.

The AMI Administration Tool window opens (see Figure 6-14).

![AMI Administration Tool](image)

Figure 6-14   AMI Administration Tool

On the AMI Administration Tool:

- Click right mouse bottom on the Service Points icon and select Create -> Service Point.

The Create a new Service Point panel appears (Figure 6-15).
Task 2 - Define AMI Service Point

To create the Receiver Service Point, enter the following:

- “DB2MQ.CV.RECEIVE” in the NAME field.
- “DB2MQ_CV_Q” in the Queue Name field.
- “DB2MQ_CV_MQM” in the Queue Manager Name field.
- Click OK.

Figure 6-15  AMI Administration Tool: new service point panel
Task 3 - Define AMI policy

In the AMI Administration Tools window:

- Click right mouse bottom on the Policies icon and select Create -> Policy. The Create a new Policy panel appears (Figure 6-16).

![Create a new Policy](image)

Figure 6-16  AMI Administration Tool: policy initialization

On the Initialization panel enter the following:

- "DB2MQ.CV.POLICY" on the Name field.
- "connectionCVMQ" on the Connection Name field.
On the Receive Panel (see Figure 6-17) we must ensure that the convert option is checked. This option controls whether WebSphere MQ will convert message automatically.

**Task 4 - Save the AMI definition**

Before leaving the AMI Administration Tool, save the AMI Service and Policy definitions to the file AMI.XML. The default directory for the file is "...\MQSeries\amt". If you should change this, make sure that the DB2 UDB environment variable **AMT_DATA_PATH** is adjusted accordingly.
6.5.6 Step 6: DB2 Data Warehouse Center implementation

To set up the data population process for the CDS, you need to log on to the Data Warehouse Center and specify the agent sites that the Data Warehouse Center will use when it accesses the source and target database. You also need to set up security for the objects that you will define for that environment. Finally, you need to set up subject areas that will contain the information about the processes that are required to populate your CDS.

Task 1 - Start Data Warehouse Center

To start the Data Warehouse Center:

1. Start the warehouse server and logger. This task is necessary only if the warehouse server and logger services are set to start manually, or if you stop the DB2 UDB server.

2. Start the warehouse agent daemon. This task is required only if you are using an iSeries or zSeries warehouse agent, or if you are using a remote Windows NT or Windows 2000 warehouse agent that must be started manually.

3. Start the Data Warehouse Center administrative interface.

4. Define an agent site to the Data Warehouse Center.

5. Define security for the Data Warehouse Center.

1 - Start the warehouse server and logger

To start the Data Warehouse Center, you must start the warehouse server and logger if they are not started automatically. Both processes run as services on Windows NT, Windows 2000, or Windows XP. To start the services, you must restart the system after you initialize the warehouse control database. Then the warehouse server and logger will automatically start every time you start Windows NT, Windows 2000, or Windows XP unless you change them to a manual service, or unless you stop the DB2 UDB server.

If you stop the DB2 UDB server, connections to local and remote databases are lost. You must stop and restart the warehouse server and logger after you stop and restart the DB2 UDB server to restore connections.

To start the warehouse server and logger manually do the following:

- Click Start-->Settings-->Control Panel-->Services.

- Scroll down the list until you find DB2 Warehouse Server. Click Start, and OK. Starting the warehouse server process will automatically start the warehouse logger process.
For a detailed description of how to set up and start warehouse server and warehouse logger on other platforms, please refer to Data Warehouse Center Administration Guide, SC27-1123-00.

2 - Start the warehouse agent daemon
In our environment we use the default setup for the warehouse agent daemon. The warehouse server comes with a warehouse agent, called the local agent. The local agent is defined as the default warehouse agent for all Data Warehouse Center activities. The local agent starts automatically when the warehouse server starts.

You can manually start the warehouse agent daemon just as you start any Windows NT, Windows 2000, or Windows XP service:

- Click Start-->Settings-->Control Panel-->Services.
- Scroll down the list until you find warehouse agent daemon. Click Start, and OK.

For a detailed description of how to set up and start remote warehouse agent daemons, please refer to Data Warehouse Center Administration Guide, SC27-1123-00.

3 - Start the Data Warehouse Center GUI
To start the Data Warehouse Center administrative interface:

- Click Start-->Programs-->IBM DB2-->Business Intelligence Tools-->Data Warehouse Center. The Data Warehouse Center Logon windows opens.
- Click Advanced if you are logging on for the first time. The Advanced windows opens as Figure 6-18.

![Advanced Window](image)

**Figure 6-18  DWC Logon: Advanced Window**

- In the Control Database field, we type CDS_MD - the ODBC system data set name of the warehouse control database.
- In the Server host name field, we type localhost - because Control Database is setup the same workstation where the warehouse server is installed.
Click **OK**. The Data Warehouse Center Logon window appears as in Figure 6-19.

![DWC Logon Window](image)

*Figure 6-19  DWC Logon Window*

- In the **User ID** field, we type `db2admin` - the User ID that we specified when we initialized the Control Database for the Data Warehouse Center (see 6.5.1, “Step 1: CDS server initialization” on page 232).
- In the **Password** field, we type the password for user `db2admin`
- Click **OK** to get access to the Data Warehouse Center.

**Task 2 - Define agent sites**
The workstation where a warehouse agent daemon is installed, becomes the AgentSite for this daemon. The Data Warehouse Center uses this definition to identify the workstation on which to start the agent. The default AgentSite is the server where the warehouse server is installed.

In our scenario warehouse agent and warehouse server are located on the same machine. Therefore, the default AgentSite can be used for warehouse agent daemon.

**Note:** If you intent to set up a warehouse agent on a separate machine you must define this machine as agent site to the Data Warehouse Center.

**Task 3 - Define security**
During Data Warehouse Center initialization a Default DWC User and Default Security Group is created. The User-ID that is used for the initialization becomes the Default DWC User - in our scenario `db2admin`.

We used the default settings to obtain the required privileges to work with Data Warehouse Center objects.

**Task 4 - Define a Subject Area**
A subject area in the Data Warehouse Center identifies and groups process that relate to a logical area of business.
To define the subject area:

- From the Data Warehouse Center tree, right-click the **Subject Areas** folder, and click **Define**.

  The Subject Area Properties notebook opens as in Figure 6-20.

![Define Subject Area Panel](image)

- In the **Define Subject** panel enter “CDS Data Population” as the name for our Subject Area in the **Name** field.
- Accept the default value in the **Administrator** field.
- Click **OK**.

### 6.5.7 Step 7: Define Common Data Store sources

This section describes the necessary tasks to define the different data sources we used in our scenario in the Data Warehouse Center. The Data Warehouse Center uses the specifications in the warehouse sources to access and select the data.

In this section we describe the following tasks:

- Defining the federated database CDS_DB as warehouse source and select *Stock Portfolio* and *the Mutual Fund* tables as source tables
- Defining a view for MQSeries messages.
- Defining the flat files dempgraphics.txt as data source.
Before you can define data sources to the Data Warehouse Center you have to set up the connectivity to the different data sources you plan to use. For our scenario the connectivity setup is described in sections 6.5.2, “Step 2: install MQSeries Integration” on page 235 - 6.5.5, “Step 5: MQSeries AMI Implementation” on page 242.

Defining federated data sources
Defining the Stock Portfolio and Mutual Fund tables as federated data sources to the Data Warehouse Center requires to complete the following tasks:
1. Open the Warehouse Source notebook.
2. Add information about the warehouse source that is the federated database.
3. Specify an agent site to access the source data.
4. Specify information about the Common Data Store database
5. Import source tables and views information
6. Authorize access to the defined warehouse source

Task 1 - Open the Warehouse Sources notebook
- In the Data Warehouse Center click Warehouse Source --> Define-->DB2 Family. The Define Data warehouse window opens.
- On the Warehouse Source panel, we enter “Federated Data Sources” in the Name field as in Figure 6-21.

![Figure 6-21 Define Stock Data Source: warehouse source panel](image)

Task 2 - Specify agent site to access the source data
On the AgentSite panel we accept the default values.
The default agent site is the machine where the warehouse server is located. In our scenario warehouse agent and warehouse server are on the same machine. Therefore, we can accept the default value for the agent site.

**Task 3 - Specify information about the federated database source**

On the **Data Source** panel (Figure 6-22) enter the following:
- CDS_DB in the **Data source name** field.
- The hostname C55VK69R in the **System name** field.
- Username db2admin in the **User Id** field.
- The db2admin user password in the **Password/Verify** field.

![Figure 6-22 Specify information on the federated database](image)

**Task 4 - Import source tables and views information**

We select the tables (in this case correlated to nicknames) from the **Tables and Views** panel that we need as data sources from the CDS_DB database. In the **Available tables** box on the left side (see Figure 6-23):
- Expand the **Table** icon to see all available federated tables existing in CDS_DB database
- By pushing the “>” button, select all available tables to the **Selected table** box.

**Note:** When a new data source (another RDBMS for example) and its tables are federated in the federated database CDS-DB, the only thing to do will be to select the tables to include them as new data sources tables.
Chapter 6. Business Case IV: Populating a Common Data Store

Figure 6-23 Define Stock data sources: tables and views panel

**Note:** Enabling federated access to remote data sources simplifies the ETL administration as the federated database is defined once. Add more federated data sources only consists of selecting more available tables. DB2 Warehouse Manager benefits also of the federated server optimizer to optimize the access to any federated source.

**Task 5 - Authorize access to the defined warehouse source**
On the Security panel we accepted the default value “Default Security Group” for the **Selected warehouse groups** - because user db2admin belongs to that group.

**Defining a view for WebSphere MQ messages**
To create a view for WebSphere MQ message queue we first defined a new data source to the Data Warehouse Center, and then added a view definition to this data source.

**Task 1 - Define a data source object**
To define the data source object

- Click in the Data Warehouse Center on **Warehouse Source --> Define-->DB2 Family**. The Define Data warehouse window opens.
- On the **Warehouse Source** panel, enter “Customer CICS/VSAM Queue Access” in the **Name** field.
- On the **AgentSite** panel, accepted the default settings.
- On the **Database** panel enter the following:
  - “CDS_DB” in the **Database name** field.
“CDS” in the **System name** field.
– User-ID “db2admin” in the **User Id** field.
– User the password for db2admin in **Password/Verify** field.

▶ We skipped the **Table** panel
▶ On the Security panel we accepted the default value “Default Security Group” for the **Selected warehouse groups** - because user db2admin belongs to that group.

**Task 2 - Define the view for the MQSeries message**
To create a view for MQSeries messages the following actions are required:
▶ From the Data Warehouse Center window, expand the **Warehouse Sources** tree.
▶ Expand the warehouse source that is to contain the view.
▶ Right-click the **Views** folder, and click **Create for MQSeries messages**....
The MQSeries wizard opens as in Figure 6-24.

---

**Figure 6-24** MQSeries Table Function Wizard: UDF panel
As depicted in Figure 6-24, we checked the first box on the UDF panel. So the UDF we are going to create will do destructive reads from the underlying queue (messages are deleted from queue after the read request).

![Figure 6-25 MQSeries Table Function Wizard - Name Panel](image)

On the **Name** panel (Figure 6-25) enter the following:

- MQRECEIVE_CV as the name for the UDF in the **Name** field.
- Any comment you want in the **Comment** field.
On the **Source MQ** panel (Figure 6-26) enter the name of the AMI Service and AMI Policy the UDF is to use. On the panel enter:

- **DB2MQ.CV.SERVICE** in the **Service point name** field. In AMI this points to the MQSeries queue manager and message queue the UDF is to use.
- **DB2MQ.CV.POLICY** in the **Policy name** field.
On the Message Format panel (Figure 6-28) the delimiter character is used in MQ messages needs to be defined. For our setup we selected ";".

Click the Show Sample Content button to test whether the UDF can access the queue that is defined in the AMI ServicePoint.
On success the wizard should come up with the pop-up window **Sample MQSeries Message Content** (Figure 6-28), either showing the first message in the queue or nothing (the queue is empty). If the wizard returns with an error, you must fix that before you proceed (check whether the queue manager is started and the queue that is defined in the AMIServicePoint is available).
Figure 6-29  MQSeries Table Function Wizard - Column Definition Panel

On **Column Definition** panel define the columns that are in the MQ message. The column definition depicted in Figure 6-29 reflects the message format we use in our scenario (number of columns and data type).

Click the **Sample result** button to see whether the UDF returns the correct column order and column format.
Click the **Show Sample Content** button to test whether the UDF can access the queue that is defined in the AMI ServicePoint.

On success the wizard should come up with the pop-up window **Sample MQSeries Result Data** (Figure 6-30), either showing the first message in the queue or nothing (the queue is empty). If the wizard returns with an error, you must fix that before you proceed.
Figure 6-31  MQSeries Table Function Wizard: options panel

On panel Option (Figure 6-31) check the **Create a corresponding table view** box. This creates a view over the table UDF. Instead of referring to the MQSeries during the Data Warehouse Center source setup, the view defined over the table UDF is used.

In the panel field **View name for receive UDF** enter CUSTOMER_CV_MSGQ.
The `Summary` panel (Figure 6-32) gives an overview of previous panel entries. Click **OK**, to create the UDF.

When you complete the wizard, a new view is created in the Data Warehouse Center. When you select the view, the MQSeries queue is accessed and each message is parsed as a delimited string according to your specifications in the wizard.

**Defining remote flat file data source**

In this task, the remote flat files that contains the demographic data is defined as a file data source to the Data Warehouse Center.

To define a file source to the Data Warehouse Center:

- Click in the Data Warehouse Center on **Warehouse Source** --> **Define**--->**Flat File**--->**Remote files**. The Define Data Warehouse Source window opens.
- On the **Warehouse Source** panel, enter “Demographic Data” in the **Name** field
- On the **AgentSite** panel, accept the default settings for Agent Site.
On the Files panel click bottom **Advanced**. The pop-up window **Advanced** opens as in Figure 6-33).

![Advanced pop-up window](image)

**Figure 6-33   Define Remote File Source: remote server**

In the Advanced pop-up window enter the following:

- In the **System name** filed enter the IP-Address of the server where the flat file is located.
- In the **User ID** filed the ID of a user who has authority on the remote server to access the flat file.
- In the **Password/Verify password** fields the password for the UserID.
- Click **OK**.
- On the **Files** panel go to the empty record. Right-click **Define**. The **Define Warehouse Source File** windows opens as in Figure 6-34.
On the Warehouse Source File panel, enter "/database/gateway/demographics.txt" in Name field.

On the Parameters panel, check that Character is selected for the File type field and "," for the Field delimiter character. For the Transfer type chose Binary is the source file is in ASCII format.

Click the Fields tab. Based on your entries in the Warehouse Source File and Parameters panel, a sample of the flat file content is displayed in the Fields panel.
Chapter 6. Business Case IV: Populating a Common Data Store

6.5.8 Step 8: Defining Common Data Store targets

After we defined the sources in the previous step we need now to define the target tables in our Common Data Store to the Data Warehouse Center.

To define warehouse target:

- Right-click the Warehouse Targets folder, on the Data Warehouse Center.
- Click Define->DB2 Family. The Define Warehouse Target notebooks opens as in Figure 6-36.
In the Data Warehouse Target panel, enter “Common Data Store - Target” in the Name field.

On the AgentSite panel we accept the default values.

The default agent site is the machine where the warehouse server is located. In our scenario warehouse agent and warehouse server are on the same machine. Therefore, we can accept the default value for Agent Site.
On the Database panel (see Figure 6-37) the target database is defined. Enter the following:

- The target database name “CDS_DB” in the Database name field.
- The DB2 instance name “CDS” in the System name field.
- The User-ID “db2admin” in the User ID field.
- The password of user ad2admin in the Password/Verify field.

We do not use Data Warehouse Center supplied Transformer in our scenario. Therefore, we checked Drop or not create transformers in the Target database field and Do not enable target for transformer in the Warehouse target field.
Click the **Tables** tab to get to the **Tables** panel (see Figure 6-38). Expand the **Table** icon in the **Available tables** box on the left side. By pushing the ">>" bottom we select all available tables to the **Selected table** box. Click **OK**. The **Define Warehouse Target** notebook closes. The Common Data Store is added to the Warehouse Targets.

**Defining Common Data Store target flat files**

In this step, a local flat files is defined. It is used as a target for the file transfer of the remote flat file demographic data for.

To define a file target to the Data Warehouse Center:

- Click in the Data Warehouse Center on **Warehouse Target --> Define--->Flat File--->Local files**. The Define Data Warehouse Source window opens.
On the **Warehouse Source** panel, enter “Demographic Load File” in the **Name** field.

On the **AgentSite** panel, accept the default settings for Agent Site.

On the **Files** panel go to the empty record. Right-click **Define**. The **Define Warehouse Source File** window opens as in Figure 6-39.

![Figure 6-39 Data Warehouse Center Target: flat file definition](image)

On the **Warehouse Source File** panel, enter `C:\AA_DIRKER\II-MO-Residency\data gateway\demographics.txt` in the **Name** field.

On the **Parameters** panel, check that Character is selected for the **File type** field and “,” for the **Field delimiter character**.

Click **OK**.
6.5.9 Step 9: Defining data transformation and movement

We used the Data Warehouse Center Process Builder to define a process, which moves data from our source systems into the Common Data Store.

The process we defined consists of several steps. Each of these steps defines how the source data is extracted, transformed and moved into a target object into the Common Data Store.

The process we defined consists of the following seven process steps:

- Stock Portfolio - transfer
- Mutual Fund - transfer/transform
- Mutual Fund Orders - transfer/transform
- Stock Orders - transfer/transform
- CV-Queue to CustTab
- Demographic Data - Copy file using FTP
- Load Demographics Data

For each of this process steps we had to specify the following:

- One or more source tables, views, or files from which the Data Warehouse Center is to extract data.
- A target table to which the Data Warehouse Center is to write data.
- How the data is to be transferred/transformed. For this we used either SQL statements or warehouse programs (for example DB2 UDB Load utility to load source data from a flat file into a target table).

In the following we will give a brief overview on the definition we chose for each process step. For a detail description of how to define processes an process step in Data Warehouse Center, please refer to DB2 Data Warehouse Center Administration Guide, SC27-1123-00.

Process step: Stock Portfolio

In this process step we define the data transfer from the Stock Portfolio table (accessed on the Informix server through the federated database and nicknames) into the target CDS table.

In the Process Model the following objects were defined:

- Source Table: FEDTAB.STOCKPORTFOLIO.
- Target Table: CDS.STOCKPORTFOLIO.
- Process Step Type: SQL Process Step of type Select/Insert
- Process Step Name: Stock Portfolio.
Process step: Mutual Fund Portfolio

In this process step we define the data transfer from the Mutual Fund Portfolio table (accessed on the remote DB2 UDB server through the federated database and nicknames) into the target CDS table.

In the Process Model the following objects were defined:

- Source Table: FEDTAB.MUTUALPORTFOLIO.
- Target Table: CDS.MUTUALPORTFOLIO.
- Process Step Type: SQL Process Step of type Select/Insert
- Process Step Name: Mutual Fund Portfolio

Process step: Mutual Fund Orders transfer/transform

This process step defines the data transfer from the Mutual Fund Orders table (accessed on the DB2 UDB server through the federated database and nicknames) into the target Customer Orders table.

In the Process Model the following objects were defined:

- Source Table: FEDTAB.MUTUAL_FUND_ORDERS.
- Target Table: CDS.CUSTOMER_ORDERS.
- Process Step Type: SQL Process Step of type Select/Insert
- Process Step Name: Mutual Fund Orders transfer/transform.

The SQL statements we use for the process step was enhanced to handle the transformation we describe in the following.

Decoding

The integer value for MUTUAL_FUND_SYMBOL the MUTUAL_FUND ORDERS is decoded to a textual representation in the CUSTOMER_ORDERS table. For this decoding we enhanced the generated SQL-Statement by with the SQL CASE clause that is listed in Example 6-17.

Example 6-17 Decoding

```sql
CASE FEDTAB.MUTUAL_FUND_ORDERS.MUTUAL_FUND_SYMBOL
  when 1 then 'MLP'
  when 2 then 'BDD'
  when 3 then 'MF50'
  when 4 then 'MF100'
  when 5 then 'Z100'
  when 6 then 'AAAA'
  ELSE 'TAX10' END AS MUTUAL_FUND_SYMBOL
```
Compensation

The column ORDER_TYPE in the CUSTOMER_ORDERS table is used to distinguish whether an order is a Mutual Fund or Stock order. As the column ORDER_TYPE is not implemented in the source tables, we have to generate the appropriate value. For this compensation the Select statement is used to generate the appropriate column value (see Example 6-18). In case of the Mutual Fund Orders this value is "MF".

Example 6-18  Compensation

```sql
SELECT
    FEDTAB.MUTUAL_FUND_ORDERS.CUST_ID AS CUST_ID,
    FEDTAB.MUTUAL_FUND_ORDERS.ORDER_NUMBER AS ORDER_NUMBER,
    CASE FEDTAB.MUTUAL_FUND_ORDERS.MUTUAL_FUND_SYMBOL
        when 1 then 'MLP'
        when 2 then 'BDD'
        when 3 then 'MF50'
        when 4 then 'MF100'
        when 5 then 'Z100'
        when 6 then 'AAAA'
        else 'TAX10' end AS MUTUAL_FUND_SYMBOL,
    FEDTAB.MUTUAL_FUND_ORDERS.ORDER_STATUS AS ORDER_STATUS,
    FEDTAB.MUTUAL_FUND_ORDERS.QUANTITY AS QUANTITY,
    FEDTAB.MUTUAL_FUND_ORDERS.PRICE AS PRICE,
    'MF',
    FEDTAB.MUTUAL_FUND_ORDERS.TRANSACTION_DAY AS TRANSACTION_DAY,
    FEDTAB.MUTUAL_FUND_ORDERS.TRANSACTION_TIME AS TRANSACTION_TIME
FROM
    FEDTAB.MUTUAL_FUND_ORDERS
```

- Type conversion (CHAR to DATE type conversion)
- Decoding (CASE clause in the extracting SQL statement to decode values for columns SYMBOL_CODE from integer code to textual representation).
- Compensation (extracting Select statement is used to generate an appropriate value for the target column ORDER_TYPE)

Process step: Stock Orders transfer/transform

This process step defines the data transfer from the Stock Orders table (accessed on the Informix server through the federated database and nicknames) into the target Customer Orders table.

In the Process Model the following objects were defined:

- Source Table: FEDTAB.STOCK_ORDERS.
- Target Table: CDS.CUSTOMER_ORDERS.
The SQL statements we use for the process step was enhanced to handle the transformation we describe in the following.

**Compensation**
The column ORDER_TYPE in the CUSTOMER_ORDERS table is used to distinguish whether a order is a Mutual Fund or Stock order. As the column ORDER_TYE is not implemented in the source tables, we have to generate the appropriate value. For this compensation the Select statement is used to generate the appropriate column value. In case of the Stock Orders this value is “SP”.

**Process step: CV-Queue to Customer Table**
In this process step we define the data transfer from the Stock Portfolio table (accessed on the Informix server through the federated database and nicknames) into the target CDS table.

For the process step the following objects were defined in the Process Model:

- Source View: CDS.MQCUSTOMER.
- Target Table: CDS.CUSTOMER.
- Process Step Type: SQL Process Step of type Select/Insert
- Process Step Name: CV_QUEUE to CustTab.

**Process step: copy demographic data file using ftp**
In this process step the file transfer program ftp is used to copy the flat file demographic.txt from the DB2AIX to the CDS server.

For the process step the following objects were defined in the Process Model:

- Source File: /database/gateway/demographic.txt
- Target File: “C:\AA_DIRKER\II-MO-Residency\data
gateway\demographics.txt”
- Process Step Type: File Programs - Copy files using FTP
- Process Step Name: Demographic Data - Copy files using FTP

**Process Step: load demographic data**
In this process step we define the data transfer from the flat file Demographic (on our CDS server) into the target CDS table.
In the Process Model the following objects were defined:

- **File Source:** “C:\AA_DIRKER\II-MO-Residency\data
gateway\demographics.txt”.
- **Target Table:** `CDS.DEMOGRAPHIC`.
- **Process Step Type:** DB2 Programs - DB2 UDB Load
- **Process Step Name:** `Load Demographics`
- **DB2 Program Load Mode:** REPLACE.

**Defining the process steps flow**

The sequence in which the process steps are executed by the Data Warehouse Manager are defined with **On Success** link in the Process Model.

Figure 6-40 depicts the On Success sequence that we defined for the process steps.

![Diagram of the process steps flow](image)

**6.5.10 Step 10: Process deployment and scheduling**

Process deployment and scheduling can be handled by the integrated Data Warehouse Center functions.

In the Process Model component of the Data Warehouse Center a process can either be in mode **Development**, **Test**, or **Production**. In order to deploy a process to your production environment its mode must be promoted from either Development or Test to Production (for a detail description, please refer to *DB2 Data Warehouse Center Administration Guide, SC27-1123-00*).
The scheduling for a process can be setup with the integrated Data Warehouse Center Scheduler.

To set up the scheduling for a process:

- Right-click Data Store Population and select from the pop-up menu Schedule.

The Schedule - Common Data Store Data Population window opens as in Figure 6-41.

![Schedule - Common Data Store Data Population window](image)

**Figure 6-41  Process scheduling: Schedule panel**

On the Schedule panel the schedule policy for a process is defined. In our example we selected:

- “Daily” for the Interval field and “Every day” for the Frequency field in the Occurs box.
- “October 22, 2002” for the Date field and “1:00” a.m. for the Time field in the Start box.
- “Run indefinitely” in the End box.
- Right-click the cravens and select ADD.
- Select the Step Task Flow panel as in Figure 6-42 - we skip the Process Task Flow, because we have only a single process to schedule.
On the **Step Task Flow** panel the sequence of process step, as defined in the process model is listed.

On the **Notification** panel (see Figure 6-43) the different types and modes of notification for a process schedule can be defined. In our example we selected “On success” and entered the required entries for the notification type **E-mail**.
6.6 Further steps

This scenario can be enhanced using DB2 Information Integrator to expand its federated access to heterogeneous relational databases as Oracle and access to other unstructured data as XML files.

An explanation of how to configure federation access to Oracle instead of Informix is provided in “Further steps” on page 110.

Data sources and processes definitions are the same in Data Warehouse Center. The federated access to the same table is transparent to Data Warehouse Center whatever the table is on Informix or Oracle.

6.7 Integration consideration

This chapter focused on how the Common Data Store approach can be used as a solution for information integration scenarios.

When to consider CDS

The CDS approach is another solution to integrate information at the data management layer by enabling federated access technology.

One reason for that could be performance. In cases where data needs to be joined over several diverse sources or the number of concurrent user is potential high, the impact on the source system might be not acceptable, especially in cases where critical operational systems are involved (stock trading). A CDS can be a solution to off load that workload to a dedicated server. Or think of call center application where you want to combine specific customer data and click-stream data that get captured when a customer visits your companies Web sites. You might want to apply data mining techniques on that data. In this case you need customer related data, that might be distributed over several data sources. In such a case the mining process could become a costly business. For mixing OLTP/Web transaction like workload with Business Intelligence techniques a CDS might be a far better solution.

Another reason could be security. Allowing customer over the internet to access data might be a risky business if that data resides on operational systems.

A third reason could be maintenance. If you have a bunch of diverse applications with the same operational pattern and the same need for information integration, why not set up a dedicate data base that can serve all.
Extending the CDS example
For our scenario we used only a subset of the technology that is available with DB2 UDB. The focus here was not to demonstrate all available functions and features, instead we want to demonstrate on a simplified example, how this solution can enable federated access and how the technology can be used as the information integration layer between source systems and a target Common Data Store.

For some business scenarios our assumption concerning operational availability, data refresh frequency, and the read-only nature of the CDS might be to restrict. The following outlines how potential objections against a CDS solution can be resolved.

**Operational availability**
Doing a full refresh each day might need to have the CDS offline for quite a while - depending on the data volume that needs to loaded. An alternative approach would be to capture only the changes in the data sources and to propagate it to the target system. In this way the down time needed for the data maintenance process could be reduced (for a detailed discussion this replication methods, please refer to Building the Operational Data Store on DB2 UDB, SG24-6513-00).

**Data refresh cycles**
Doing a data refresh once a day leaves you with a copy of your data in the CDS that is one day behind your operational sources. An alternative approach to this would be that changes on the source system get propagated to the CDS as soon as the occur. The propagation could be either directly into the target objects or to intermediate areas (where synchronization and transformation can take place before data is fed into target objects).

In this way a CDS can represent a near-time copy of the underlying data sources. (for a detailed discussion this replication methods, please refer to Building the Operational Data Store on DB2 UDB, SG24-6513-00). Beyond this, it would allow to reduce the maintenance window to a minimum.

**Read/write access**
For business scenarios where applications require more then read-only access a asynchronous “write through” to operational data could be implemented (for example to allow customer to place orders). This could be accomplished by the same techniques that are used to propagated changes from the source systems to a CDS (for a detailed discussion this replication methods, please refer to Building the Operational Data Store on DB2 UDB, SG24-6513-00).
Data redundancy
In times with constantly decreasing storage costs, data redundancy might be negligible compared to the gain in performance and security that can be achieved with a CDS solution.

Availability
In today’s business environment and especially in the area with Web based self service applications 24x7 availability is among the key requirements. Beside the things that can be done to improve operational availability - and thus minimizing planned downtime of the system -, the planning for a CDS solution should include the following:

- Maintenance concept that minimizes planned downtime for hardware and software.
- Backup/Recovery policy that reduces unplanned downtime.
- Failover -over concept - minimize the impact of losing a system component.

Maintenance
Setting up and maintain the infrastructure that hosts and feeds a CDS comes not for free. This effort, however, has to be compared with the gains that can be achieved through reduced application complexity (less complex data access flow) and reduced application maintenance costs, because applications get shielded from changes in the underlying data sources. These changes can include anything from server location or address, platform, to data formats. When changes are needed, they are made in the data population process, not in client code or client connections.

Furthermore, for some type of applications the CDS might be the only feasible alternative in terms of performance and security.
Test MQSeries integration

This appendix provides the steps to test the MQSeries Integration. It uses the DB2 command line processor, the MQSeries Explorer, and the MQSeries API Exerciser all of which are provided as part of the product implementations on the Windows platform.

For more complete information on the DB2 functions demonstrated, please see the SQL Reference Volume 1, SC09-4844 and SQL Reference Volume 2, SC09-4845. For information on the MQSeries tools, please see WebSphere MQ for Windows Quick Beginnings Version 5.3, GC34-6073.
A.1 Step 1: Connect to database

1. Start a DB2 Command Line Processor window:
   - Click Start -> Programs -> IBM DB2 -> Command Line Tools -> Command Line Processor.
2. In the CLP window enter:
   - connect to IIREDB0K

A.2 Step 2: Test MQSEND function

1. Enter the MQSend function
   - Enter the MQSend request as shown in Figure A-1.

```
db2 VALUES db2mq.MQSEND('IIR.TEST.SEND','IIR.TEST.POLICY','Testing 678','TEST3')
```

Figure A-1 MQSEND request
A.3 Step 3: Verify MQSEND results

A.3.1 Verify that the message was sent

- Start the MQSeries Explorer application: Click Start->Programs-> IBM WebSphere MQ -> WebSphere MQ Explorer
- The WebSphere MQ Console root panel should be displayed, as shown in Figure A-2.

![Figure A-2 MQSeries Explorer](image-url)
– Click the **Queues** folder to expand the list of queues
– Right-click the **COPPER.TEST** queue and select the **Properties** option.
– Select the **Statistics** tab
– Current Depth should have a value of one or more as shown in Figure A-3. If not, check the queue and queue manager name in the send service point, IIR.TEST.SEND. Please remember that capitalization counts

*Figure A-3  COPPER.TEST statistics*
A.3.2 Verify the message contents

- From the Console Root panel, right-click the COPPER.TEST queue and select the Browse Messages option.
- The Message Browser panel should be displayed, and the information should look as shown in Figure A-4. Pay particular attention to the message format, it should be set to MQSTR.

![Figure A-4 First message browser panel](image-url)
– Click the message and select the **Properties** box.

– Select the **Identifiers** tab from the Properties panel. The information displayed should look as shown in Figure A-5. Verify that the Correlation ID is ‘TEST3’, if not check your input.

![Figure A-5  Identifiers tab](image-url)

**Figure A-5  Identifiers tab**
A.4 Step 4: Test the MQRECEIVE function

This step shows how to put a sample message on the queue:

1. Using the WebSphere MQ API exerciser to put a sample message onto the CUSTOMER.INFO.REPLY queue.
   - Start the MQSeries First Steps Application: Click Start -> Programs -> First Steps
   - Click API Exerciser -> Launch API Exerciser
   - The WebSphere MQ Exerciser panel with the Queue Managers tab displayed. Select the queue manager you are using for this test, in this case DB2MQ_DEFAULT_MQM and click the MQCONN button.
   - The reply should be an MQCC_OK as shown in Figure A-6.

![WebSphere MQ API Exerciser Panel](image)

Figure A-6  MQ API Queue Mangers Tab
– Click the **Queues** tab
– Select CUSTOMER.INFO.REPLY from the Selected Queue drop down box.
– Click the **MQOPEN** button. The reply should be an MQCC_OK as shown in Figure A-7.

![WebSphere MQ API Exerciser](image)

*Figure A-7   MQ API Queues tab*

– Check the Advanced mode box and hit the MQPUT button.
– The MQ Arguments panel should be displayed on the Message tab. Type in the message content, it can be anything you want, we used a sample valid reply from our CICS transaction as shown in Example A-1.
Example: A-1 Sample Message

GET CUST INFO 0000
0000000000FIRST NAME                  MLAST NAME
123456789 DRURY LANE                   SECOND
LINEXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXTHIRD
LINEYYYYYYYYYYYYYYYYYYYYYYYYYYYYCITYZZZZZZZZZZZZZZZZZZZZZZZZZZZZSTATE
STATESTATESTATE9999999999

– Click the Message Descriptor tab and page forward to the third page.
   Enter the Correlid field as ‘TEST3’ as shown in Figure A-8.

Note: Please make sure the Correlid field is blank filled for the full 24 characters allowed. If not the Correlid may contain non blank data, and you will be unable to retrieve the message.

Figure A-8 Message descriptor options

– Select the Put Message Options tab, and select the
  MQPMO_NEW_MESSAGE_ID option as shown in Figure A-9.
Figure A-9  Put Message Options

- Click the OK button.
- Return to the MQSeries Explorer, Console Root panel.
- Check the depth of the CUSTOMER.INFO.REPLY queue. You should have one message that, when you browse the message, is 345 bytes long.
A.5 Step 5: Verify the MQRECEIVE function

1. Return to the DB2 Command Line Processor
   – Enter the MQReceive request as shown in Example A-2.

   \textit{Example: A-2  MQRECEIVE request}

   \begin{verbatim}
   VALUES db2mq.MQreceive('IIR.BC1A.RECEIVER',
   'IIR.TEST.POLICY','TEST3')
   \end{verbatim}

   \begin{itemize}
   \item \textbf{Note:} Again, please pad the Correlid field (the third parameter on the MQReceive function) to the full 24 characters.
   \item The message returned should look exactly like the message entered in the WebSphere MQ API exerciser
   \end{itemize}

   \begin{itemize}
   \item \textbf{Attention:} You have now completed testing!
   \end{itemize}
WebSphere MQ implementation and object definitions

This appendix provides detailed instructions to perform the WebSphere MQ environment implementation steps outlined in Table B-1.

Table B-1  WebSphere MQ implementation steps

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create Queue Manager QM_copper</td>
</tr>
<tr>
<td>2</td>
<td>Define WebSphere MQ objects on COPPER</td>
</tr>
<tr>
<td>3</td>
<td>Define WebSphere MQ objects on MQV1</td>
</tr>
<tr>
<td>4</td>
<td>Test the Channels</td>
</tr>
</tbody>
</table>

For the distributed queue manager, QM_copper, we will illustrate the use of the MQSeries Explorer GUI, a tool provided as part of the WebSphere MQ queue manager installation, to create the queue manager and define the objects. In addition, if you prefer using a more batch oriented approach, the MQSC commands to define these same resources are given.
For the z/OS queue manager, MQV1 we provide a list of the definitions required. Once all definitions are in place, a simple ‘round robin’ message test is outlined to make sure the channels are functioning properly.

B.1 Create the queue manager QM_copper

In this section we describe how to create the WebSphere MQ queue manager QM_copper on Windows 2000, using MQSeries Explorer.

There are two steps to create the queue manager using the MQExplorer:

- Start the WebSphere MQExplorer
- Create queue manager

B.1.1 Step 1: Start WebSphere MQExplorer

To start the explorer after WebSphere MQ has been installed, click Start->Programs-> IBM WebSphere MQ -> WebSphere MQ Explorer.

You should see the WebSphere MQ Console Root panel as shown in Figure B-1.

![WebSphere MQ Console Root Panel](image)

**Figure B-1**  MQ Console Root Panel
B.1.2 Step 2: Create queue manager

Right-click the Queue Managers Folder and select **New -> Queue Manager**

You should see the Create Queue Manager (Step 1) panel.

Enter the following values in the box specified in column 1 of Table B-2.

<table>
<thead>
<tr>
<th>Box Label</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue Manager</td>
<td>QM_copper</td>
</tr>
<tr>
<td>Def Transmission queue</td>
<td>SYSTEM.DEFAULT.XMITQ</td>
</tr>
<tr>
<td>Dead Letter queue</td>
<td>COPPER.DLQ</td>
</tr>
</tbody>
</table>

The panel should look like the one in Figure B-2.

![Create Queue Manager (Step 1)](image)

**Figure B-2 Create queue manager - step 1**

Click the **Next** button on this panel and the next one (accept the defaults on step 2 unless otherwise advised by your WebSphere MQ administrator).

On Create Queue Manager (Step 3) panel, **check** the ‘Create Server Connection...’ option as shown in Figure B-3.
Click the **Next** button.

On Create queue Manager (Step 4) panel enter the port you want the TCP/IP listener to use. This is normally 1414 for WebSphere MQ, but in the example shown we have used 1416 because we already had queue managers listening on ports 1414 and 1415.
B.2 Define WebSphere MQ objects on QM_copper

In this section we demonstrate using the MQExplorer to create the local queues used in business case I. It includes the following sections:

- Opening and tailoring the queues folder
- Using MQExplorer panels to define the first local queue
- Defining the rest of the local queues for business case I.

B.2.1 Step 1: Opening and tailoring the queues folder

Click the + to the left of QM_copper, which should expand the folder to show two more folders, Queues and Advanced.
Click the **Queues** folder to open it up. It should appear empty, but that is because the system objects that are automatically defined when you create a queue manager are hidden from view.

Right-click the **Queues** folder, select **View**, and check **Show System Objects**. Your Console Root panel should now look like the one shown in Figure B-5.

![Figure B-5  Queues view](image)

### B.2.2 Step 2: Using MQExplorer panels to define the first local queue

We will now define the dead letter queue (DLQ) we specified in creating the queue manager. The dead letter queue is used by WebSphere MQ whenever a message is undeliverable to a target queue for some reason. ‘Missing’ messages can often be found in a DLQ when messages flow across a channel to a queue manager and the target queue is unavailable, either due to not being defined or not currently allowing updates to take place.

Right-click **Queues**, select **New -> Local Queue**
In the Queue Name box enter **COPPER.DLQ**, as shown in Figure B-6 and click **OK** to define the local queue.

![Create Local Queue](image)

*Figure B-6  Create the local queue*

Leave the Put Messages and Get Messages options as allowed, the default. This means that application programs can both insert and select messages from the queue.

For the purposes of our tests we elected to make the default message persistence for these queues non-persistent. Messages that are not persistent are not logged and are not recovered after a queue manager is restarted. An application can, and should, override this setting on a per-message basis. This gives the application control over whether the message should be logged based on its relative importance, the ease of recreation, and the type of function. Request/reply scenarios often use non-persistent messages because of the much lower overhead. Messages that initiate transactions, messages that contain financial changes, are often set to persistent.

The scope should be queue manager, the default.
The Usage option will be left as normal for most queues. One queue, COPPER.TO.MQV1 will hold messages destined for our z/OS queue manager, will have the usage option set to Transmission. Transmission queues are special queues used by WebSphere MQ to provide local storage for messages destined for queues owned by other queue managers.

Defining the rest of the local queues with the MQExplorer GUI is exactly the same, with one exception. We have found it a very good practice to add a backout threshold and queue to most application queues. We do this because each time an application issues a rollback, WebSphere MQ will place the messages currently being processed back at the beginning of queue they came from. These messages, which may contain data that cannot be processed or have been incorrectly routed to this application, are often referred to as poisoned messages. If the message continues to get rolled back to the originating queue, no messages that are ‘behind’ it in the queue will ever get processed.

If the WebSphere MQ administrator has supplied a backout threshold and queue, the application can divert a message that has exceeded the threshold onto the backout queue. Most WebSphere MQ applications, including WMQI and many other commercially available applications use this technique to prevent bottlenecks.

To define these attributes on a queue, after entering the queue name on the General tab of the queue definition panels, click the Storage tab. Enter the queue name you want the messages to roll to in the Backout Requeue Name box, your threshold value in the Backout Threshold box and make certain the ‘Hardened’ option is selected. The storage tab should look something like that shown in Figure B-7.
B.2.3 Step 3: Define remaining local queues

The queues listed in Figure B-3 should be defined as described in Step 2. The MQExplorer window should contain the entries shown in Figure B-8 after step 3 is completed.

Table B-3 Remaining local queues

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackOut</td>
<td>Do not define a backout queue or threshold</td>
</tr>
<tr>
<td>CUSTOMER.INFO.REPLY</td>
<td>Use BackOut as the backout queue</td>
</tr>
<tr>
<td>COPPER.TEST</td>
<td>Use BackOut as the backout queue</td>
</tr>
<tr>
<td>COPPER.TO.MQV1</td>
<td>The Usage option should be set to Transmission</td>
</tr>
</tbody>
</table>
B.2.4 Step 4: Using MQExplorer to define remote queues

Right-click Queues, select New -> Remote Queue Definition

Do not alter the default values. Enter the queue information as shown in Table B-4, changing the name of the z/OS queue manager and transmission queue where necessary for your environment.

Table B-4  Remote queue information

<table>
<thead>
<tr>
<th>Box Label</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue Name</td>
<td>CUSTOMER.INFO.REQUEST</td>
</tr>
<tr>
<td>Remote Queue Name</td>
<td>CUSTOMER.INFO.REQUEST</td>
</tr>
<tr>
<td>Remote Queue Manager Name</td>
<td>MQV1</td>
</tr>
<tr>
<td>Transmission Queue</td>
<td>COPPER.TO.MQV1</td>
</tr>
</tbody>
</table>
Also, for testing purposes a second remote queue will be defined, MQV1.TEST, as shown in Figure B-9.

**B.2.5 Step 5: MQExplorer panels to define channels**

This section describes defining the WebSphere MQ sender and receiver channels that provide communication between our distributed queue manager, QM_copper and our z/OS queue manager, MQV1.

**B.2.5.1 Define sender channel**

The next step is to define the communication channels between the queue manager on COPPER and the z/OS queue manager. The first thing to do is to find out the IP address of the z/OS system and the port the queue manager is listening on, the z/OS WebSphere MQ administrator should provide these.

1. From the Console Root panel, click the Advanced folder to open it up.
2. Right-click Channels, and select New->Sender Channel

![Remote queue definition](image)
3. Enter the channel name, the connection name for your environment (the IP address and port for the queue manager), and the transmission queue as shown in Figure B-10.

4. Click OK to save the channel definition.

![Create Sender Channel](image)

**Figure B-10  Create Sender Channel**

**B.2.5.2 Define receiver channel**

1. Right-click **Channels**, and select **New->Receiver Channel**.

2. Enter the receiver channel name as shown in Figure B-11.

3. Click OK to save the channel definition.
At this point all of the required resources are defined on the distributed queue manager.

**B.3 Alternative to MQExplorer: WebSphere MQ commands**

An alternative to using the MQExplorer GUI to create the queue manager and the WebSphere MQ objects needed is to use WebSphere MQ commands. To use the `RUNMQSC` commands:

Open a command prompt window on your Windows platform

Create the definitions required in notepad or a similar text editor. The ones used in our test are shown in Example B-1.

Enter the `RUNMQSC` command as shown in Figure B-12 on page 306. The results are shown as well.
Example: B-1  RUNMQSC commands

```sql
define qlocal(COPPER.DLQ) defpsist(YES) GET(ENABLED) PUT(ENABLED) USAGE(NORMAL) REPLACE
define qlocal(BackOut) defpsist(YES) GET(ENABLED) PUT(ENABLED) BOTHRESH(2) USAGE(NORMAL) REPLACE
define qremote(CUSTOMER.INFO.REPLY) defpsist(YES) GET(ENABLED) BOTHRESH(2) USAGE(NORMAL) REPLACE
define qlocal(COPPER.TEST) defpsist(YES) GET(ENABLED) PUT(ENABLED) BOTHRESH(2) USAGE(NORMAL) REPLACE
define qlocal(COPPER.TO.MQV1) defpsist(YES) GET(ENABLED) PUT(ENABLED) BOTHRESH(2) USAGE(XMITQ) REPLACE
define qremote(CUSTOMER.INFO.REQUEST) defpsist(YES) PUT(ENABLED) RNAME(CUSTOMER.INFO.REQUEST) RQMNAME(MQV1) XMITQ(COPPER.TO.MQV1) REPLACE
define qremote(MQV1.TEST) defpsist(YES) PUT(ENABLED) RNAME(MQV1.TEST) RQMNAME(MQV1) XMITQ(COPPER.TO.MQV1) REPLACE
DEFINE CHANNEL(COPPER.TO.MQV1) CHLTYPE(SDR) XMITQ(COPPER.TO.MQV1)
CONNAME('9.12.6.2(1500)') REPLACE
DEFINE CHANNEL(MQV1.TO.COPPER) CHLTYPE(RCVR) REPLACE
```

For a complete explanation of the MQSC commands, please see *WebSphere MQ Script (MQSC) Command Reference*, SC34-6055.
B.4 Define WebSphere MQ objects on MQV1

This section assumes that the WebSphere MQ object definitions are being performed by an WebSphere MQ administrator. The required definitions are shown in Table B-5.

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTOMER.INFO.REPLY</td>
<td>QREMOTE</td>
<td>Transmission queue is MQV1.TO.COPPER</td>
</tr>
<tr>
<td>CUSTOMER.INFO.REQUEST</td>
<td>QLOCAL</td>
<td>Triggered queue, Trigger Type = F, Trigger Set = Y, Initiation Queue = CICS.MQV1.INITQ01 (see note), Process Name = GCIN.PROCESS</td>
</tr>
<tr>
<td>COPPER.TEST</td>
<td>QREMOTE</td>
<td>Remote Name = COPPER.TEST, Remote Queue Mgr = QM_copper, Transmission Queue = MQV1.TO.COPPER</td>
</tr>
<tr>
<td>MQV1.TO.COPPER</td>
<td>QLOCAL</td>
<td>Usage is set to ‘X’ to indicate a transmission queue</td>
</tr>
<tr>
<td>MQV1.TO.COPPER</td>
<td>Sender Channel</td>
<td>Connection Name = combination IP and port, ours was 9.1.39.57 (1416), Transmission Queue = MQV1.TO.COPPER</td>
</tr>
<tr>
<td>COPPER.TO.MQV1</td>
<td>Receiver Channel</td>
<td></td>
</tr>
<tr>
<td>GCIN.PROCESS</td>
<td>Process</td>
<td>Application Type = CICS Application Type = GCIN START</td>
</tr>
</tbody>
</table>

Note: The initiation queue should be the initiation queue set up for the CICS region used for testing. If none has been established, please see WebSphere MQ for z/OS System Administration Guide, SC34-6053.
B.5 Test the communication channels

This test consists of the following steps:
1. Start the sender channel on QM_copper, COPPER.TO.MQV1.
2. Start the sender channel on MQV1.
3. Use the MQSeries API Exerciser to put a test message on MQV1.TEST.
4. Verify message on COPPER.TEST.

B.5.1 Step 1: Start COPPER.TO.MQV1

- In the MQSeries Con.sole Root panel, right-click COPPER.TO.MQV1.
- Select START.
- Right-click COPPER.TO.MQV1, again.
- Select status - which should now be “Running”, as shown in Figure B-13.

![Channel Status Table]

Figure B-13 COPPER.TO.MQV1 status

If the channel does not start, check your definitions and make sure the queue manager MQV1 is active.
B.5.2 Step 2: Start MQV1.TO.COPPER

From the IBM WebSphere MQ for z/OS - Main Menu:
- Select Action = 1 (Display)
- Set Object Type to Sender
- Set Name = MQV1.TO.COPPER
- On the List Channels panel, enter ‘6’ (start) in the Action field.

If the channel does not start, check your definitions and make sure the listener for QM_copper is active.

B.5.3 Step 3: Put a test message

Start the MQSeries API Exerciser as described in Section A.4, “Step 4: Test the MQRECEIVE function” on page 287.

Connect to QM_copper as described.

Open the MQV1.TEST queue.

Select the MQPUT button. Under the Message tab, enter whatever text you want to the test message. Click the OK button to MQPUT the message.

Note: Opening a remote queue requires you to check to ‘Advanced Mode’ box, and de-select the input options. A remote queue can only be opened for output.

B.5.4 Step 4: Verify the message

Check the depth of the COPPER.TEST queue via the MQExplorer, it should have been increased by 1.

You may then use the MQSeries Explorer to browse the message on COPPER.TEST to make sure the contents are the same as those entered in Step 3: Put a test message (above).
z/OS VSAM and CICS definitions and samples

This appendix provides the following that were used in our scenarios:

- VSAM file definition
- Sample data for the customer information file
- COBOL copybooks defining the customer information request, reply and file layout
- COBOL source code for the CICS program
- CICS definitions.
C.1 VSAM File definitions

The VSAM file definition for the customer information file is shown in Example C-1. Please change the high level qualifier (‘DB75U’) to reflect your environment.

Example: C-1  VSAM definition

```
DELETE (DB75U.CUSTINFO)        ERASE CLUSTER
DEFINE CLUSTER                       -
   (NAME(DB75U.CUSTINFO)      -
    VOLUMES(TOTMQF)           -
    SHAREOPTIONS(2 3))     -
DATA                              -
   (NAME(DB75U.CUSTINFO.DATA)   -
    RECORDS(60 60)         -
    RECORDSIZE(261 261)   -
    CONTROLINTERVALSIZE(4096)
    FREESPACE(0 20)       -
    KEYS(10 0))          -
INDEX                             -
   (NAME(DB75U.CUSTINFO.INDEX)   -
    RECORDS(5 5)          -
    CONTROLINTERVALSIZE(1024))
```

C.2 VSAM File sample data

The sample data for the customer information file is too wide to be displayed (261 characters long) in one example box. We have put it in 79-80 byte sections as shown in Example C-2, Example C-3, Example C-4, and Example C-5.

Example: C-2  Sample data bytes 1-71

<table>
<thead>
<tr>
<th>0000000000FIRST NAME</th>
<th>MLAST NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001 FRED</td>
<td>MLO</td>
</tr>
<tr>
<td>1002 CAROLYN</td>
<td>TELKINS</td>
</tr>
<tr>
<td>1003 IAN</td>
<td>HARVEY</td>
</tr>
</tbody>
</table>

Example: C-3  Sample data bytes 72-143

<table>
<thead>
<tr>
<th>123456789 DRURY LANE</th>
<th>SECOND LINEXXXXXXXXXXXXXXXXXXXXXXXXX</th>
</tr>
</thead>
<tbody>
<tr>
<td>123 BAY STREET</td>
<td>DUNGEON LEVEL 3</td>
</tr>
<tr>
<td>301 OLD MILL COVE</td>
<td>PENTHOUSE SUITE</td>
</tr>
<tr>
<td>111 TORONTO WAY</td>
<td></td>
</tr>
</tbody>
</table>

Example: C-4  Sample data bytes 144-231

<table>
<thead>
<tr>
<th>XXXXXXXTHIRD LINEYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY</th>
</tr>
</thead>
<tbody>
<tr>
<td>VANCROUKER</td>
</tr>
<tr>
<td>CELL #255-D</td>
</tr>
</tbody>
</table>

Example: C-5  Sample data bytes 232-261

<table>
<thead>
<tr>
<th>STATETESTATESTATE</th>
<th>999999999999</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC CANADA</td>
<td>V7E2R9</td>
</tr>
<tr>
<td>GA USA</td>
<td>30018</td>
</tr>
<tr>
<td>ON CANADA</td>
<td>LIN5K1</td>
</tr>
</tbody>
</table>
C.3 COBOL copybooks

The following COBOL copybooks were created to define the request, the reply, and the file layout.

Customer information request copybook, CUSTIREQ, is shown in Example C-6.

Example: C-6  CUSTIREQ copybook

```
*****************************************************************
* FILE NAME:     CUSTIREQ                                       *
*                                                               *
* COBOL COPYFILE CONTAINING REQUEST       MESSAGE FOR CUSTOMER  *
* NAME/ADDRESS INQUIRY REQUEST.                                 *
*                                                               *
*****************************************************************

01 CUST-INFO-REQ.
*                                                                 *
  05 CUST-IREQ-ACTION        PIC X(20)
    VALUE 'GET CUST INFO       '.
  05 CUST-IREQ-KEY           PIC X(10) VALUE SPACES.
```

Customer information reply copybook, CUSTADRR, is shown in Example C-7.

Example: C-7  CUSTADRR copybook

```
*****************************************************************
* FILE NAME:     CUSTADRR                                       *
*                                                               *
* COBOL COPYFILE CONTAINING REQUEST/REPLY MESSAGE FOR CUSTOMER  *
* NAME/ADDRESS INQUIRY REQUEST.                                 *
*                                                               *
*****************************************************************

01 CUST-ADDR-REQ-REPLY.
*                                                                 *
  05 CUST-ADDR-ACTION        PIC X(20)
    VALUE 'GET CUST INFO       '.
  05 CUST-ADDR-RESPONSE-CD   PIC 9(04)  VALUE ZERO.
  05 CUST-ADDR-ERROR-MSG     PIC X(60)  VALUE SPACES.
  05 CUST-KEY                PIC X(10)  VALUE SPACES.
```
Customer information file definition copybook, CUSTINFO, is shown in Example C-8.

Example: C-8  CUSTINFO copybook

* ******************************************************
* IBM REDBOOK                                          *
* *                                               *
* FILE NAME:  CUSTINFO                                *
* *                                               *
* COBOL COPYFILE CONTAINING CUSTINFO FILE RECORD       *
* *                                               *
* ******************************************************

01 CUSTINFO-REC.
 *
05 CUSTINFO-KEY.
  10 CUSTINFO-ID          PIC X(10) VALUE SPACES.
05 CUSTINFO-DATA.
  10 CUSTINFO-NAME.
    15 CUSTINFO-FIRST-NAME  PIC X(30) VALUE SPACES.
    15 CUSTINFO-MI          PIC X(01) VALUE SPACES.
    15 CUSTINFO-LAST-NAME   PIC X(30) VALUE SPACES.
  10 CUSTINFO-ADDR.
    15 CUSTINFO-ADDR-LINE1 PIC X(40) VALUE SPACES.
    15 CUSTINFO-ADDR-LINE2 PIC X(40) VALUE SPACES.
    15 CUSTINFO-ADDR-LINE3 PIC X(40) VALUE SPACES.
    15 CUSTINFO-ADDR-CITY   PIC X(40) VALUE SPACES.
    15 CUSTINFO-ADDR-ST-PROV PIC X(20) VALUE SPACES.
    15 CUSTINFO-ADDR-CODE   PIC X(10) VALUE SPACES.
 *
---
C.4 COBOL source code

The COBOL program used to fulfill the customer information requests, READCINF, is shown in Example C-9.

Example: C-9  READCINF

CBL NODYNAM,LIB,OBJECT,RENT,RES,APOST
* --------------------------------------------------------------------- *
IDENTIFICATION DIVISION.
* --------------------------------------------------------------------- *
PROGRAM-ID. READCINF.
*REMARKS
******************************************************
******************************************************
* IBM WebSphere MQ for z/OS *
* *
* MODULE NAME : READCINF *
* *
* ENVIRONMENT : CICS; COBOL II *
* *
* CICS Transaction Name : GCIF *
* *
* DESCRIPTION : SAMPLE PROGRAM TO SHOW A MQ REQUEST FOR A VSAM *
* FILE READ BEING PROCESSED. *
* THE CUSTOMER INFORMATION FILE IS READ, AND *
* IF A MATCHING RECORD IS FOUND FORMAT THE *
* REPLY. IF NO MATCHING RECORD FOUND AN ERROR *
* MESSAGE IS RETURNED. *
* *
* THE QUEUE ON WHICH THE INPUT MESSAGE IS TO BE *
* FOUND IS IDENTIFIED IN THE TRIGGER INFORMATION *
* AVAILABLE TO THE PROGRAM. *
* *
* DEPENDENCY : USES VSAM FILE KNOWN TO CICS AS 'CUSTINFO' *
* *
* ************************************************************** *
EJECT
EJECT
* ------------------------------------------------------------- *
ENVIRONMENT DIVISION.
* ------------------------------------------------------------- *
* ------------------------------------------------------------- *
DATA DIVISION.
* ------------------------------------------------------------- *
* ------------------------------------------------------------- *
WORKING-STORAGE SECTION.
* ------------------------------------------------------------- *
* REQUEST MESSAGE
* COPY CUSTIREQ.
* REPLY MESSAGE
* COPY CUSTADDR.
* FILE RECORD
* COPY CUSTINFO.
* W00 - General work fields
* COPY CSQ4VBB.
  01 W05-TS-MESSAGE-LENGTH PIC S9(4) BINARY.
  01 W05-TD-MESSAGE-LENGTH PIC S9(4) BINARY.
  01 W00-MESSAGE PIC X(70).
  01 W00-STARTCODE PIC X(02).
  01 W00-WAIT-INTERVAL PIC S9(09) BINARY VALUE 30000.
  01 W00-RESPONSE PIC S9(09) BINARY.
* W01 - Queue names
* 01 W01-QUEUE-NAMES.
  05 W01-DEAD-QNAME PIC X(48) VALUE 'CSQ4SAMP.DEAD.QUEUE '.
* W03 - API fields
* 01 W03-HCONN PIC S9(9) BINARY VALUE ZERO.
  01 W03-OPTIONS PIC S9(9) BINARY.
  01 W03-HOBJ-CHECKQ PIC S9(9) BINARY.
  01 W03-COMPCODE PIC S9(9) BINARY.
  01 W03-REASON PIC S9(9) BINARY.
  01 W03-DATALEN PIC S9(9) BINARY.
  01 W03-BUFFLEN PIC S9(9) BINARY.
* W04 - READ MESSAGE LENGTH
  01 W04-READ-MESSAGE-LENGTH PIC S9(4) BINARY.
* API control blocks
* 01 MQM-OBJECT-DESCRIPTOR.
  COPY CMQODV.
  01 MQM-MESSAGE-DESCRIPTOR.
  COPY CMQMDV.
  01 MQM-PUT-MESSAGE-OPTIONS.
COPY CMQPMOV.
01 MQM-GET-MESSAGE-OPTIONS.
   COPY CMQGMOV.
01 MQM-TRIGGER-MESSAGE.
   COPY CMQTML.

*    CICS ts queue fields

*    CMQV contains constants (for filling in the control blocks)
*    and return codes (for testing the result of a call)

*    DFHAID contains the constants used for checking for
*    attention identifiers

COPY DFHAID SUPPRESS.

*    -------------------------------
*    LINKAGE SECTION.
*    -------------------------------

EJECT

*    -------------------------------
*    PROCEDURE DIVISION.
*    -------------------------------

A-MAIN SECTION.

*    -------------------------------
*    *    THIS SECTION Initializes and controls the program flow
*    *    *    After opening the input queue, the program enters a loop
*    *    *    getting and processing messages. Once no more messages
*    *    *    are available, shown by the program timing out, control is
*    *    *    returned to CICS
*    *    *    -------------------------------
*    *    *    Check that the program has been started with data
EXEC CICS ASSIGN
      STARTCODE(W00-STARTCODE)
END-EXEC.

*    RETRIEVE THE TRIGGER DATA FOR THIS TRANSACTION
EXEC CICS RETRIEVE
      INTO(MQTM)
END-EXEC.

*    AT THIS POINT THE DATA RETRIEVED HAS NAME OF THE QUEUE
*    WHICH HAS CAUSED THIS PROGRAM TO BE TRIGGERED
*    Open the queue
MOVE MQOT-Q      TO MQOD-OBJECTTYPE.
MOVE MQTM-QNAME  TO MQOD-OBJECTNAME.

*    INITIALIZE OPTIONS AND OPEN THE QUEUE FOR INPUT
*    COMPUTE W03-OPTIONS = MQOO-INPUT-SHARED +
               MQOO-SAVE-ALL-CONTEXT.
*    CALL 'MQOPEN' USING W03-HCONN
      MQOD
      W03-OPTIONS
      W03-HOBJ-CHECKQ
      W03-COMPCODE
      W03-REASON.

*    TEST THE OUTPUT FROM THE OPEN, IF
*    NOT OK THEN EXIT PROGRAM
*    IF W03-COMPCODE NOT = MQCC-OK THEN
       GO TO A-MAIN-EXIT.

*    Now get and process messages
*    COMPUTE MQGMO-OPTIONS = MQGMO-WAIT +
             MQGMO-ACCEPT-TRUNCATED-MSG +
             MQGMO-CONVERT +
             MQGMO-SYNCPPOINT.
MOVE LENGTH OF CUST-ADDR-REQ-REPLY
      TO W03-BUFFLEN.
MOVE W00-WAIT-INTERVAL        TO MQGMO-WAITINTERVAL.
MOVE MQMI-NONE                TO MQMD-MSGID.
MOVE MQCI-NONE                TO MQMD-CORRELID
*    MAKE THE FIRST MQGET CALL OUTSIDE THE LOOP
CALL 'MQGET' USING W03-HCONN
W03-HOBJ-CHECKQ
MQMD
MQGMO
W03-BUFFLEN
CUST-INFO-REQ
W03-DATALEN
W03-COMPCODE
W03-REASON.

TEST THE OUTPUT OF THE MQGET CALL USING THE PERFORM LOOP
THAT FOLLOWS

LOOP FROM HERE TO END-PERFORM UNTIL THE MQGET CALL FAILS

PERFORM WITH TEST BEFORE
UNTIL W03-COMPCODE = MQCC-FAILED

PERFORM THE MESSAGE RECEIVED
PERFORM PROCESS-QUERY
EXEC CICS SYNCPOINT END-EXEC

RESET PARAMETERS FOR THE NEXT CALL

MOVE LENGTH OF CUST-ADDR-REQ-REPLY
TO W03-BUFFLEN
MOVE MQMI-NONE TO MQMD-MSGID
MOVE MQCI-NONE TO MQMD-CORRELID
COMPUTE MQGMO-OPTIONS = MQGMO-WAIT +
MQGMO-ACCEPT-TRUNCATED-MSG +
MQGMO-CONVERT +
MQGMO-SYNCPOINT

GET THE NEXT MESSAGE

CALL 'MQGET' USING W03-HCONN
W03-HOBJ-CHECKQ
MQMD
MQGMO
W03-BUFFLEN
CUST-INFO-REQ
W03-DATALEN
W03-COMPCODE
W03-REASON
* TEST THE OUTPUT OF THE MQGET CALL AT THE TOP OF THE LOOP.
* EXIT THE LOOP IF AN ERROR OCCURS
* END-PERFORM.
* PERFORM CLOSE-QUEUES.
* A-MAIN-EXIT.
* Return to CICS
EXEC CICS RETURN
END-EXEC.
* GOBACK.
EJECT

* ------------------------------------------------------------ *
PROCESS-QUERY SECTION.
* ------------------------------------------------------------ *
* THIS SECTION DEVELOPS A REPLY MESSAGE AND PUTS IT ONTO THE REPLY QUEUE OF THE INPUT MESSAGE. IF THE PUT TO THE REPLY QUEUE FAILS, THIS IS RECORDED AND THE MESSAGE IS FORWARDED TO THE DEAD QUEUE.
* TO DEVELOP THE REPLY MESSAGE THE SECTION USES THE DATA IN THE INPUT MESSAGE TO OBTAIN INFORMATION FROM A FILE. IF NO RELEVANT RECORD IS FOUND IN THE FILE AN 'UNKNOWN' MESSAGE IS RETURNED.
* ------------------------------------------------------------ *
* INITIALIZE THE REPLY MESSAGE WITH DETAILS FROM THE QUERY MESSAGE
* MOVE CUST-IREQ-KEY TO CUSTINFO-ID.
MOVE CUST-IREQ-KEY TO CUST-KEY.
* LOOK FOR THE ACCOUNT NUMBER IN THE FILE
* MOVE LENGTH OF CUSTINFO-REC
TO W04-READ-MESSAGE-LENGTH.
* EXEC CICS READ
FILE('CUSTINFO')
INTO(CUSTINFO-REC)
LENGTH(W04-READ-MESSAGE-LENGTH)
RIDFLD(CUSTINFO-ID)
KEYLENGTH(10)
RESP(W00-RESPONSE)
END-EXEC.

* EXAMINE THE RESPONSE TO THE FILE READ
* EVALUATE TRUE
 WHEN (W00-RESPONSE = DFHRESP(NORMAL))
  * Account number found
    MOVE CUSTINFO-NAME TO CUST-NAME
    MOVE CUSTINFO-ADDR TO CUST-ADDR
  *
 WHEN (W00-RESPONSE = DFHRESP(NOTFND))
  * NO RECORD OF CUSTOMER ID FOUND
    MOVE 13 TO CUST-ADDR-RESPONSE-CD
    MOVE 'CUSTOMER INFO RECORD NOT FOUND' TO CUST-ADDR-ERROR-MSG
  *
 WHEN OTHER
  * ERROR READING FILE - RECORD AND FEEDBACK ERROR
    MOVE 99 TO CUST-ADDR-RESPONSE-CD
    MOVE 'FILE READ ERROR' TO CUST-ADDR-ERROR-MSG
END-EVALUATE.

* SET THE OBJECT DESCRIPTOR, MESSAGE DESCRIPTOR AND PUT
* MESSAGE OPTIONS TO THE VALUES REQUIRED TO CREATE THE
* MESSAGE.
* SET THE LENGTH OF THE MESSAGE

MOVE MQMD-REPLYTOQMGR TO MQOD-OBJECTQMGRNAME.
MOVE 'CUSTOMER.INFO.REPLY' TO MQOD-OBJECTNAME.
MOVE MQMT-REPLY TO MQMD-MSGTYPE.
MOVE MQRO-NONE TO MQMD-REPORT.
MOVE SPACES TO MQMD-REPLYTOQ.
MOVE SPACES TO MQMD-REPLYTOQMGR.
MOVE MQFMT-STRING TO MQMD-FORMAT.
MOVE MQMI-NONE TO MQMD-MSGID.
COMPUTE MQPMO-OPTIONS = MQPMO-SYNCPPOINT + MQPMO-PASS-IDENTITY-CONTEXT.
MOVE W03-HOBJ-CHECKQ TO MQPMO-CONTEXT.

* CALL 'MQPUT1' USING W03-HCONN
  MQOD
  MQMD
  MQPMO
  W03-BUFFLEN
  CUST-ADDR-REQ-REPLY
WO3-COMPCODE
WO3-REASON.

* PROCESS-QUERY-EXIT.
* *
*    RETURN TO PERFORMING SECTION

*    EXIT.
  EJECT
*

* ------------------------------------------------------------- *
CLOSE-QUEUES SECTION.                          *
* ------------------------------------------------------------- *
* THIS SECTION CLOSES THE INPUT QUEUE. ALL OUTPUT FROM THIS *
* PROGRAM USES MQPUT1, SO NO OUTPUT QUEUES ARE OPEN         *
* ------------------------------------------------------------- *

  CALL 'MQCLOSE' USING W03-HCONN
    W03-HOBJ-CHECKQ
    MQCO-NONE
    W03-COMPCODE
    W03-REASON.

CLOSE-QUEUES-EXIT.
* *
*    RETURN TO PERFORMING SECTION

*    EXIT.
  EJECT
*

* ------------------------------------------------------------- *
* ------------------------------------------------------------- *
RECORD-CALL-ERROR SECTION.                    *
* ------------------------------------------------------------- *
* THIS SECTION WRITES AN ERROR MESSAGE TO THE CICS TD QUEUE *
* 'CSML' AND THE CICS TS QUEUE 'CSQ4SAMP'.                *
* THE FAILING OPERATION AND OBJECT NAME FIELDS ARE COMPLETED *
* BY THE CALLING APPLICATION. THE REMAINING FIELDS OF THE  *
* MESSAGE ARE COMPLETED BY THIS ROUTINE                   *
* ------------------------------------------------------------- *

  EXEC CICS ASKTIME
    ABSTIME(W05-ABSTIME)
END-EXEC.

Appendix C. z/OS VSAM and CICS definitions and samples  323
EXEC CICS FORMATTIME
  ABSTIME(W05-ABSTIME)
  DATEFORM(W05-DATETIME)
  DATE(W05-DATE)
  DAYOFMONTH(W05-DD)
  MONTHOFYEAR(W05-MM)
  YEAR(W05-YYYY)
  TIME(M02-TIME) TIMESEP
END-EXEC.

* MOVE W05-YYYY TO M02-YYYY.
  MOVE W05-MM TO M02-MM.
  MOVE W05-DD TO M02-DD.
  MOVE EIBTRNID TO M02-TRANSACTION
      M03-TRANSACTION.
  MOVE EIBTASKN TO M02-TASK-NUMBER
      M03-TASK-NUMBER.
  MOVE W03-COMPCODE TO M02-COMPCODE
  MOVE W03-REASON TO M02-REASON
  MOVE M02-DATE TO M03-DATE.
  MOVE M02-TIME TO M03-TIME.
  MOVE LENGTH OF M02-CALL-ERROR-MSG
      TO W05-TS-MESSAGE-LENGTH
  MOVE LENGTH OF M03-CSML-ERROR-MSG
      TO W05-TD-MESSAGE-LENGTH.

EXEC CICS WRITEQ TS
  QUEUE('CSQ4SAMP')
  FROM (M02-CALL-ERROR-MSG)
  LENGTH(W05-TS-MESSAGE-LENGTH)
END-EXEC.

EXEC CICS WRITEQ TD
  QUEUE('CSML')
  FROM (M03-CSML-ERROR-MSG)
  LENGTH(W05-TD-MESSAGE-LENGTH)
END-EXEC.

RECORD-CALL-ERROR-EXIT.

* Return to performing section
* EXIT.
  EJECT
* ------------------------------------------------------------- *
* ------------------------------------------------------------- *
* END OF PROGRAM *
C.4.1 Compiling and linking

A sample procedure to compile and link the COBOL program READCINF is in Example C-10.

Example: C-10  Sample CICS/MQ/COBOL compile proc

```
//CICMQCOB PROC SUFFIX=1$,
餍
//* This procedure has been changed since CICS/ESA Version 3
//* Parameter INDEX2 has been removed
//* INDEX='CICSTS22.CICS', Qualifier(s) for CICS libraries
//* COMPHLQ='IGY.V2R2MO', Qualifier(s) for COBOL compiler
//* CEEHLO='CEE', Qualifier(s) for LE370
//* MQMHLQ='MQ530A', Qualifier(s) for MQSeries
//* OUTC=A, Class for print output
//* REG=2M, Region size for all steps
//* LNKPARM='LIST,XREF', Link edit parameters
//* WORK=SYSDA Unit for work datasets
//* This procedure contains 4 steps
//* 1. Exec the COBOL translator
//*    (using the supplied suffix 1$)
//* 2. Exec the vs COBOL II compiler
//* 3. Reblock &LIB(&STUB) for use by the linkedit step
//* 4. Linkedit the output into dataset &PROGLIB
//* The following JCL should be used
to execute this procedure
//APPLPROG EXEC DFHEITVL
//TRN.SYSIN DD *

/* Application program */
/* */
/* */
/* /* */
/* /* */
/* /* */
/* /* */
/* /* */
/* /* */
/* Where anyname is the name of your application program. */
(Refer to the system definition guide for full details, including what to do if your program contains calls to
```
Getting Started on Integrating Your Information

```c
/* the common programming interface. */
//TRN EXEC PGM=DFHECP&SUFFIX,
//      PARM='COBOL2',
//      REGION=&REG
//STEPLIB DD DSN=&INDEX..SDFHLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSPUNCH DD DSN=&SYSCIN,
//      DISP=(,PASS),UNIT=&WORK,
//      DCB=BLKSIZE=400,
//      SPACE=(400,(400,100))
/*
//COB EXEC PGM=IGYCRCTL,REGION=&REG,
//      PARM='NODYNAM,LIB,OBJECT,RENT,RES,APOST,MAP,XREF'
//STEPLIB DD DSN=&COMPHLQ..SIGYCOMP,DISP=SHR
//SYSLIB DD DSN=&INDEX..SDFHCOB,DISP=SHR
//      DSN=&INDEX..SDFHMAC,DISP=SHR
//      DSN=&INDEX..SDFHSAMP,DISP=SHR
//      DSN=&MQMHLQ..SCSQCOBC,DISP=SHR
//      DSN=DB75U.COBOL,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSIN DD DSN=&&SYSCIN,DISP=(OLD,DELETE)
//SYSLIN DD DSN=&&LOADSET,DISP=(MOD,PASS),
//      UNIT=&WORK,SPACE=(80,(250,100))
//SYSUT1 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT2 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT3 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT4 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT5 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT6 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT7 DD UNIT=&WORK,SPACE=(460,(350,100))
/*
//COPYLINK EXEC PGM=IEBGENER,COND=(7,LT,COB)
//SYSUT1 DD DSN=&INDEX..SDFHCOB(DFHEILIC),DISP=SHR
//SYSUT2 DD DSN=&COPYLINK,DISP=(NEW,PASS),
//      DCB=(LRECL=80,BLKSIZE=400,RECFM=FB),
//      UNIT=&WORK,SPACE=(400,(20,20))
//SYSPRINT DD SYSOUT=&OUTC
//SYSIN DD DUMMY
/*
//LKED EXEC PGM=IEWL,REGION=&REG,
//      PARM='&LNKPARM',COND=(5,LT,COB)
//SYSLIB DD DSN=&INDEX..SDFHLOAD,DISP=SHR
//      DSN=&CEEHLQ..SCEECICS,DISP=SHR
//      DSN=&CEEHLQ..SCEELKED,DISP=SHR
//MQMLIB DD DSN=&MQMHLQ..SCSQLOAD,DISP=SHR
//SYSLMOD DD DSN=DB75U.LOAD,DISP=SHR
//SYSLMOD DD DISP=SHR,DSN=CICSTS22.CICS.SDFHLOAD
//SYSUT1 DD UNIT=&WORK,DCB=BLKSIZE=1024,
//      SPACE=(1024,(200,20))
```
Example JCL to compile the program is in Example C-11.

Example: C-11  Compile JCL sample

//CBARAR5C JOB MSGCLASS=O,CLASS=A,NOTIFY=&SYSUID
// JCLLIB ORDER=(DB75U.PROCLIB)
//APPLPROG EXEC CICMQCOB
//TRN.SYSIN DD DISP=SHR,DSN=DB75U.COBOL(READCINF)
//*LKED.MQLIB DD DISP=SHR,DSN=MQ530A.SCSQLOAD
//LKED.SYSIN DD *
  INCLUDE MQMLIB(CSQCSTUB)
  NAME READCINF(R)
/*
//
C.5 CICS definitions

The CICS administrator, or WebSphere MQ administrator for z/OS should set up the CICS definitions needed for integration. For our scenarios, we created a new CICS group called IIREDBOK for this document. It includes the three resources as shown in Table C-1.

Table C-1  CICS Resources List

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUSTINFO</td>
<td>FILE</td>
</tr>
<tr>
<td>READCINF</td>
<td>PROGRAM</td>
</tr>
<tr>
<td>GCIN</td>
<td>TRANSACTION</td>
</tr>
</tbody>
</table>

We used CICS Resource Definition Online (RDO) to create the group and define all the resources. For additional information on defining resources to CICS please see, CICS Resource Definition Guide, SC34-5990.

Once the resources are defined, the IIREDBOK group should be installed.

C.5.1 CUSTINFO Definition

The CUSTINFO file definition is shown in three panels, Figure C-1 on page 329, Figure C-2 on page 330, and Figure C-3 on page 331. In the first, the DSName should be set to the name of the VSAM file defined in Example C-1 on page 312, the recordsize should be set to 261.

Set DSName and recordsize
Set keylength

In the second, the keylength should be set to 10 as shown in Figure C-2.
**Set recordformat**

In the third, the record format is set to fixed ('F'), as shown in Figure C-3.
C.5.2  READCINF Definition

The READCINF program definition is illustrated on the CICS program definition RDO panel shown in Figure C-4. The two input fields are the program name (required) and the language.
C.5.3 GCIN Definition

The GCIN transaction definition is illustrated on the CICS transaction definition RDO panel shown in Figure C-5. The two input fields are the transaction and program names.
C.5.4 IIREDBOK group installation

Use the command in Example C-12 to install the resources defined above. At this point the CICS resources are available for use.

Example: C-12  IIREDBOK installation
CEDA INSTALL GROUP(IIREDBOK)

C.6 WebSphere MQ/CICS definitions

If not already done, the MQ adapter and automatic initiation must be set up in the CICS region selected for the test. Instructions for this can be found in
For our test, we used the initiation queue already defined in our CICS region, MQV1.CICS01.INITQ.
This appendix includes the complete DADX syntax used in the Web Services scenario.

The DADX file is an XML document. The elements of the DADX are described here.

1. Root element: `<DADX>`
   Attributes:
   - `xmlns:dadx`: The namespace of the DADX.
   - `xmlns:xsd`: The namespace of the W3C XML Schema specification
   - `xmlns:wsdl`: The namespace of the W3C Web services Definition Language specification

   Children:
   - 1.1 `<wsdl:documentation>`
     Specifies a comment or statement about the purpose and content of the Web service. You can use XHTML tags.
1.2 <implements>

Specifies the namespace and location of the Web service description files. It allows the service implementor to declare that the DADX Web service implements a standard Web service described by a reusable WSDL document defined elsewhere; for example, in a UDDI registry.

1.3 <result_set_metadata>

Stored procedures can return one or more result sets which can be included in the output message. Metadata for a stored procedure result set must be defined explicitly in the DADX using the <result_set_metadata> element. At run-time, the metadata of the result set is obtained and it must match the definition contained in the DADX file.

Note: Therefore, only stored procedures that have result sets with fixed metadata can be invoked.

This restriction is necessary in order to have a well-defined WSDL file for the Web service. A single result set metadata definition can be referenced by several <call> operations, using the <result_set> element. The result set metadata definitions are global to the DADX and must precede all of the operation definition elements.

Attributes:

name: Identifies the root element for the result set.
rowname: Used as the element name for each row of the result set.

Children:

1.3.1 <column>

Defines the column. The order of the columns must match that of the result set returned by the stored procedure. Each column has a name, type and nullability, which must match the result set.

Attributes:

name: Required. Specifies the name of the column.
type: Required if element is not specified. Specifies the type column.
 element: Required if type is not specified. Specifies the element of column
as: Optional. Provides a name for a columns.
nullable: Optional. Nullable is either true or false. It indicates whether column values can be null.
1.4 <operation>

Specifies a Web service operation. The operation element and its children specify the name of the operation, and the type of operation the Web service will perform, such as compose an XML document, query the database, or call a stored procedure. A single DADX file can contain multiple operations on a single database or location. The following list describes these elements.

Attribute:

- **name**: A unique string that identifies the operation; the string must be unique within the DADX file. For example: “findByCustomerID”

Children:

Document the operation with the following element:

1.4.1 <wsdl:documentation>

Specifies a comment or statement about the purpose and content of the operation. You can use XHTML tags.

Specify the type of operation using one of the following child elements:

1.4.2 <retrieveXML>

Specifies to generate zero or one XML documents from a set of relational tables using the XML collection access method. Depending on whether a DAD file or an XML collection name is specified, the operation will call the appropriate XML Extender composition stored procedure.

Children:

Specify which of these stored procedures is used by passing either the name of a DAD file, or the name of the collection using one of the following elements:

1.4.2.1 <DAD_ref>

The content of this element is the name and path of a DAD file. If a relative path is specified for the DAD file, the current working directory is assumed to be the group directory.

1.4.2.2 <collection_name>

The content of this element is the name of the XML collection. Collections are defined using the XML Extender administration interfaces, as described in DB2 XML Extender Administration and Programming.

Specify override values with one of the following elements:
1.4.2.3 `<no_override/>`

Specifies that the values in the DAD file are not overridden. Required if neither `<SQL_override>` nor `<XML_override>` element. Specifies a parameter for an operation. Use a separate parameter element for each parameter referenced in the operation. Each parameter name must be unique within the operation.

Attributes:

- **name**: The unique name of the parameter. A parameter must have its contents defined by either an XML Schema element (a complex type) or a simple type.
- **element**: Use the “element” attribute to specify an XML Schema element.
- **type**: Use the “type” attribute to specify a simple type.
- **kind**: Specifies whether a parameter passes input data, returns output data, or does both. The valid values for this attribute are:
  - in

1.4.3 `<storeXML>`

Specifies to store (decompose) an XML document in a set of relational tables using the XML collection access method. Depending on whether a DAD file or an XML collection name is specified, the operation will call the appropriate XML Extender decomposition stored procedure.

Children:

Specify which of these stored procedures is used by passing either the name of a DAD file, or the name of the collection using one of the following elements:

1.4.3.1 `<DAD_ref>`

The content of this element is the name and path of a DAD file. If a relative path is specified for the DAD file, the current working directory is assumed to be the group directory.

1.4.3.2 `<collection name>`

The content of this element is the name of an XML collection. Collections are defined using the XML Extender administration interfaces, as described in DB2 XML Extender Administration and programming.
1.4.4 <query>

Specifies a query operation. The operation is defined by an SQL SELECT statement in the SQL_select element. The statement can have zero or more named input parameters. If the statement has input parameters then each parameter is described by a <parameter> element.

This operation maps each database column from result set to a corresponding XML element. You can specify XML Extender user-defined types (UDTs) in the <query> operation, however this requires an <XML_result> element and a supporting DTD that defines the type of the XML column queried.

Children:

1.4.4.1 <SQL_query>

Specifies an SQL SELECT statement.

1.4.4.2 <XML_result>

Optional. Defines a named column that contains XML documents. The document type must be defined by the XML Schema element of its root.

Attributes:

name Specifies the root element of the XML document stored in the column.

element Specifies the particular element within the column.

1.4.4.3 <parameter>

Required when referencing a parameter in the <SQL_query> element. Specifies a parameter for an operation. Use a separate parameter element for each parameter referenced in the operation. Each parameter name must be unique within the operation.

Attributes:

name The unique name of the parameter. A parameter must have it contents defined by one of the following: an XML Schema element (a complex type) or a simple type.

element Use the “element” attribute to specify an XML Schema element.

type Use the “type” attribute to specify a simple type.
kind Specifies whether a parameter passes input data, returns output data, or does both. The valid values for this attribute are:

- in

1.4.5 <update>

The operation is defined by an SQL INSERT, DELETE, or UPDATE statement in the <SQL_update> element. The statement can have zero or more named input parameters. If the statement has input parameters then each parameter is described by a <parameter> element.

Children:

1.4.5.1 <SQL_update>

Specifies an SQL INSERT, UPDATE, or DELETE statement.

1.4.5.2 <parameter>

Required when referencing a parameter in the <SQL_update> element. Specifies a parameter for an operation. Use a separate parameter element for each parameter referenced in the operation. Each parameter name must be unique with the operation.

Attributes:

name The unique name of the parameter. A parameter must have its contents defined by one of the following: an XML Schema element (a complex type) or a simple type.

element Use the “element” attribute to specify an XML Schema element.

type Use the “type” attribute to specify a simple type.

kind Specifies whether a parameter passes input data, returns output data, or does both. The valid values for this attributes are:

- in

1.4.6 <call>

Specifies a call to a stored procedure. The processing is similar to the update operation, but the parameters for the call operation can be defined as ‘in’, ‘out’, or ‘in/out’. The defualt parameter kind is ‘in’. The ‘out’ and ‘in/out’ parameters appear in the output message.
1.4.6.1 <SQL_call>
Specifies a stored procedure call.

1.4.6.2 <parameter>
Required when referencing a parameter in the <SQL_call> element. Specifies a parameter for an operation. Use a separate parameter element for each parameter referenced in the operation. Each parameter name must be unique with the operation.

Attributes:
- **name** The unique name of the parameter. A parameter must have its contents defined by one of the following: an XML Schema element (a complex type) or a simple type.
- **element** Use the "element" attribute to specify an XML Schema element.
- **type** Use the "type" attribute to specify a simple type.
- **kind** Specifies whether a parameter passes input data, returns output data, or does both. The valid values for this attribute are:
  - in
  - out
  - in/out

1.4.6.3 <result_set>
Defines a result set and must follow any <parameter> elements. The result set element has a name which must be unique among all the parameters and result sets of the operation, and must refer to a <result_set_metadata> element. One <result_set> element must be defined for each result set returned from the stored procedure.

Attributes:
- **name** A unique identifier for the result sets in the SOAP response.
- **metadata** A result set meta data definition in the DADX file. The identifier must refer to the name of an element.
Web Services XML and DADX files

This appendix presents the XML and DADX files used in Chapter 4, “Business case II: Accessing your information through Web Services” on page 119:

- Addition stock and mutual fund information XML files.
- The entire stockInfo.dadx file

We also include additional Web material that accompanies this redbook.
E.1 Additional XML files

In our Web services scenario, we need stock and mutual fund information. This information includes the current price, description, type, and so on of certain stock/mutual fund. To complete our scenario, we need the following stock and mutual fund information to be inputted into the DB2 XML collection. Table specifies the list of XML files:

Table E-1 List of stock/mutual fund information XML files

<table>
<thead>
<tr>
<th>Symbol</th>
<th>XML file name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALEX</td>
<td>alex.xml</td>
</tr>
<tr>
<td>CNB</td>
<td>cnb.xml</td>
</tr>
<tr>
<td>FGFND</td>
<td>fgfnd_mutualfund.xml</td>
</tr>
<tr>
<td>IBM</td>
<td>ibm.xml</td>
</tr>
<tr>
<td>LMMKT</td>
<td>lmmkt_mutualfund.xml</td>
</tr>
<tr>
<td>MSFT</td>
<td>microsoft.xml</td>
</tr>
<tr>
<td>QVC</td>
<td>qvc.xml</td>
</tr>
<tr>
<td>RVG</td>
<td>rvg.xml</td>
</tr>
<tr>
<td>SCBFD</td>
<td>scbdf_mutualfund.xml</td>
</tr>
<tr>
<td>SCGFD</td>
<td>scgfd_mutualfund.xml</td>
</tr>
<tr>
<td>TBFND</td>
<td>tbfnd_mutualfund.xml</td>
</tr>
</tbody>
</table>

This XML is defined by the stock_mutualFund_info.dtd. We show a couple of examples here, the rest of the XML files you can download from the Internet following the instructions in (cross reference).

Example E-1 shows the information associated with Sandwich Club Growth Fund.

Example: E-1 Content of the scgfd_mutualfund.xml file

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE stock_mutualFund_info SYSTEM "stock_mutualFund_info.dtd">

<stock_mutualFund_info key="10">
  <symbol>SCGFD</symbol>
  <description>Sandwich Club Growth Fund</description>
  <type>Mutual Fund</type>
  <sector>Food and entertainment</sector>
</stock_mutualFund_info>
```
Example E-2 shows the information associated with IBM:

Example: E-2 Content of the ibm.xml file

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE stock_mutualFund_info SYSTEM "stock_mutualFund_info.dtd">

<stock_mutualFund_info key="4">
  <symbol>IBM</symbol>
  <description>Internation Business Machines</description>
  <type>Equity</type>
  <sector>Technology Hardware</sector>
  <price>62.84</price>
  <market>NYSE</market>
</stock_mutualFund_info>
```

You can build the rest of the XML files by following the examples.

**E.2 The complete stockInfo.dadx file**

You will find the stockInfo.dadx in its entirety in Example E-3. Included in the file are some additional viewMyPortfolio implementation. One of which does not make call to MQ so you can test viewMyPortfolio with the MQ connectivity. The file is also included as part of the additional material you can download from the Internet.

Example: E-3 File stockInfo.dadx in its entirety

```xml
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Working with your stock and mutual funds portfolio.</documentation>

  <operation name="storeStockMutualFundInfo">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Decomposing an XML document and storing the information into a DB2 collection table.</documentation>
    <storeXML>
      <DAD_ref>stock_mutualFund_info.dad</DAD_ref>
    </storeXML>
  </operation>
</DADX>
```
<operation name="makeMQRequest">
<documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Call a stored procedure to put in a MQ Request</documentation>
<call>
<SQL_call>CALL MQ_REQUEST(:custID, :rowsAffected, :errorCode, :errorLabel)</SQL_call>
<parameter name="custID" type="xsd:string" kind="in"/>
<parameter name="rowsAffected" type="xsd:int" kind="out"/>
<parameter name="errorCode" type="xsd:int" kind="out"/>
<parameter name="errorLabel" type="xsd:string" kind="out"/>
</call>
</operation>

<!--
<operation name="viewMyPortfolio">
<documentation xmlns="http://schemas.xmlsoap.org/wsdl/">View my current portfolio.</documentation>
<query>
<SQL_query>
select T.FIRST, T.LAST, T.ID
FROM
(SELECTsubstr(msg,1,20) as REQ,
 substr(msg,21,4) as RETCODE,
 substr(msg,25,60) as ERR_MSG,
 substr(msg,85,10) as ID,
 substr(msg,95,30) as FIRST,
 substr(msg,125,1) as MI,
 substr(msg,126,30) as LAST,
 substr(msg,156,40) as ADRLN1,
 substr(msg,196,40) as ADRLN2,
 substr(msg,236,40) as ADRLN3,
 substr(msg,276,40) as CITY,
 substr(msg,316,20) as STATE,
 substr(msg,336,10) as ZIPCODE
from
<table(db2mq.MQRECEIVEALL('IIR.BCIA.RECEIVER', 'IIR.TEST.POLICY', '1001', 1)) AS X ) as T
</SQL_query>
</query>
</operation>

<operation name="viewMyPortfolio">
<documentation xmlns="http://schemas.xmlsoap.org/wsdl/">View my current portfolio.</documentation>
<query>
<SQL_query>
SELECT
B.ID,
rtrim(T.firstname) || ', ' || rtrim(T.lastname) as Name,
B.SYMBOL,
A.DESCRIPTION,
B.QUANTITY,
A.PRICE,
B.QUANTITY * A.PRICE AS MARKET_VALUE,
T.CITY
FROM
  (select
    CUST_ID as ID,
    STOCK_SYMBOL as SYMBOL,
    TOTAL_SHARES as QUANTITY
  from ids_ifmx92nt.stock_portfolio_tab where CUST_ID = :custID1
  union
  select
    CUSTOMER_ID as ID,
    MUTUAL_FUND_SYMBOL as SYMBOL,
    QUANTITY as QUANTITY
  from acc_mf_portfolio where CUSTOMER_ID = :custID2 ) AS B,
STOCK_MUTUALFUND_INFO A,
ACC_CUST_INFO T
WHERE A.SYMBOL = B.SYMBOL AND B.ID = T.CUSTOMER_ID
</SQL_query>
<parameter name="custID1" type="xsd:string"/>
<parameter name="custID2" type="xsd:string"/>
</query>
</operation>

<operation name="viewMyPortfolio">
<documentation xmlns="http://schemas.xmlsoap.org/wsdl/">View my current portfolio.</documentation>
<query>
<SQL_query>
SELECT
  B.ID,
  rtrim(T.LOC) || ', ' || rtrim(T.FIRST) as Name,
B.SYMBOL,
A.DESCRIPTION,
B.QUANTITY,
A.PRICE,
B.QUANTITY * A.PRICE AS MARKET_VALUE,
T.CITY
FROM
  (select
    CUST_ID as ID,
    STOCK_SYMBOL as SYMBOL,
    TOTAL_SHARES as QUANTITY
  from ids_ifmx92nt.stock_portfolio_tab where CUST_ID = :custID1
  union
  select
    CUSTOMER_ID as ID,
    MUTUAL_FUND_SYMBOL as SYMBOL,
    QUANTITY as QUANTITY
  from acc_mf_portfolio where CUSTOMER_ID = :custID2 ) AS B,
STOCK_MUTUALFUND_INFO A,
ACC_CUST_INFO T
WHERE A.SYMBOL = B.SYMBOL AND B.ID = T.CUSTOMER_ID
</SQL_query>
<parameter name="custID1" type="xsd:string"/>
<parameter name="custID2" type="xsd:string"/>
</query>
</operation>
from ids_ifmx92nt.stock_portfolio_tab where CUST_ID = :custID1
union
select
    CUSTOMER_ID as ID,
    MUTUAL_FUND_SYMBOL as SYMBOL,
    QUANTITY as QUANTITY
from acc_mf_portfolio where CUSTOMER_ID = :custID2 ) AS B,
STOCK_MUTUALFUND_INFO A,
(SELECTsubstr(msg,1,20) as REQ,
    substr(msg,21,4) as RETCODE,
    substr(msg,25,60) as ERR_MSG,
    substr(msg,85,10) as ID,
    substr(msg,95,30) as FIRST,
    substr(msg,125,1) as MI,
    substr(msg,126,30) as LAST,
    substr(msg,156,40) as ADRln1,
    substr(msg,196,40) as ADRln2,
    substr(msg,236,40) as ADRln3,
    substr(msg,276,40) as CITY,
    substr(msg,316,20) as STATE,
    substr(msg,336,10) as ZIPCODE
FROM
    TABLE(db2mq.MQRECEIVEALL('IIR.BC1A.RECEIVER', 'IIR.TEST.POLICY',
        '1001      ' , 1)) AS X ) as T
WHERE A.SYMBOL=B.SYMBOL AND B.ID=T.ID
</SQL_query>
<parameter name="custID1" type="xsd:string"/>
<parameter name="custID2" type="xsd:string"/>
</operation>

<operation name="viewRecentTransactions">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">View a list of customers recent trading transactions.</documentation>
    <query>
        <SQL_query>
            select cust_id,
                order_number,
                transaction_date,
                action_indicator,
                symbol,
                quantity,
                price
            from orders
            where cust_id = :custID and
                current_date - transaction_date &lt; :daysOfHistory
        </SQL_query>
        <parameter name="custID" type="xsd:string"/>
        <parameter name="daysOfHistory" type="xsd:int"/>
    </query>
</operation>
<operation name="makeAnOrder">
  <documentation xmlns="http://schemas.xmlsoap.org/wsd1/">Make an order to buy or sell a stock or mutual fund.</documentation>
  <update>
    <SQL_update>
      insert into orders (cust_ID, symbol, action_indicator, transaction_date, quantity, price)
      values (:custID, :symbol, :action, current_date, :quantity, :price)
    </SQL_update>
    <parameter name="custID" type="xsd:string"/>
    <parameter name="symbol" type="xsd:string"/>
    <parameter name="action" type="xsd:string"/>
    <parameter name="quantity" type="xsd:decimal"/>
    <parameter name="price" type="xsd:decimal"/>
  </update>
</operation>

</DADX>
Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246892

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the Additional materials and open the directory that corresponds with the redbook form number, SG24-6892.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
</table>
Getting Started on Integrating Your Information

6892-add_material.zip All the files we used to complete the scenarios in Business Case I and II.

System requirements for downloading the Web material

The following system configuration is recommended:

- **Hard disk space:** 326 KB minimum
- **Operating System:** Windows

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder. The content is as follow:

```
\Business Case I
\Business Case I\COBOL
\Business Case I\COBOL\CUSTADDR.txt
\Business Case I\COBOL\CUSTINFO.txt
\Business Case I\COBOL\CUSTIREQ.txt
\Business Case I\COBOL\READCINF.txt
\Business Case I\COBOL\samplecompile.txt
\Business Case I\COBOL\sampleproc.txt
\Business Case I\Database
\Business Case I\Database\CreateInvestmentsView.txt
\Business Case I\Database\DB2V72_Defs.txt
\Business Case I\Database\GETPORTFOLIO.txt
\Business Case I\Database\IIRedBook_Database_Create.txt
\Business Case I\Database\INFORMIX_Defs.txt
\Business Case I\Database\MQReceiveExample.txt
\Business Case I\Database\MQSendExample.txt
\Database\MutualFundImportCMDSample.txt
\Business Case I\Database\MutualFundSampleData.txt
\Business Case I\Database\MutualFundTableDef.txt
\Business Case I\Database\StockTableDef.txt
\Business Case I\Database\StockTableSampleData.txt
\Business Case I\MQSeries
\Business Case I\MQSeries\MQDEFS_for_zos.txt
\Business Case I\MQSeries\RUNMQSC.cmd.txt
\Business Case I\VSAM
\Business Case I\VSAM\CustinfoSampleData.txt
\Business Case I\VSAM\VSAMDEFINE.txt
\Business Case II Web services
\Business Case II Web services\dad
\Business Case II Web services\dad\stock_mutualFund_info.dad
\Business Case II Web services\dadx
\Business Case II Web services\dadx\stockinfo.dadx
```
## Abbreviations and acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMI</td>
<td>Application Messaging Interface</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BI</td>
<td>Business Intelligence</td>
</tr>
<tr>
<td>BLOB</td>
<td>Binary Large Object</td>
</tr>
<tr>
<td>BO</td>
<td>Business Objects</td>
</tr>
<tr>
<td>CDR</td>
<td>Call Detail Record</td>
</tr>
<tr>
<td>CICS</td>
<td>Customer Information Control System</td>
</tr>
<tr>
<td>CLOB</td>
<td>Character Large Object</td>
</tr>
<tr>
<td>COBOL</td>
<td>Common Business Oriented Language</td>
</tr>
<tr>
<td>CRM</td>
<td>Customer Relationship Management</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma Separated Variables</td>
</tr>
<tr>
<td>DAD</td>
<td>Document Access Definition</td>
</tr>
<tr>
<td>DBA</td>
<td>Data Base Administrator</td>
</tr>
<tr>
<td>DTD</td>
<td>Data Type Definition</td>
</tr>
<tr>
<td>DWH</td>
<td>Data Warehouse</td>
</tr>
<tr>
<td>EAR</td>
<td>Enterprise Application Archive</td>
</tr>
<tr>
<td>ERP</td>
<td>Enterprise Resource Planning</td>
</tr>
<tr>
<td>ETL</td>
<td>Extract Transform Load</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transport Protocol</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines Corporation</td>
</tr>
<tr>
<td>IM</td>
<td>Intelligent Miner</td>
</tr>
<tr>
<td>IM4D</td>
<td>DB2 Intelligent Miner for Data</td>
</tr>
<tr>
<td>ITSO</td>
<td>International Technical Support Organization</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Enterprise Edition</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MIME</td>
<td>Multipurpose Internet Mail Extensions</td>
</tr>
<tr>
<td>MIS</td>
<td>Management Information System</td>
</tr>
<tr>
<td>MQI</td>
<td>MQSeries application programming interface</td>
</tr>
<tr>
<td>ODS</td>
<td>Operational Data Store</td>
</tr>
<tr>
<td>OLAP</td>
<td>OnLine Analytical Processing</td>
</tr>
<tr>
<td>OLTP</td>
<td>OnLine Transactional Processing</td>
</tr>
<tr>
<td>PMML</td>
<td>Predictive Model Markup Language</td>
</tr>
<tr>
<td>QMF</td>
<td>Query Management Facility</td>
</tr>
<tr>
<td>RBF</td>
<td>Radial Basis Function</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational DataBase Management System</td>
</tr>
<tr>
<td>ROI</td>
<td>Return On Investment</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedural Call</td>
</tr>
<tr>
<td>SMS</td>
<td>Short Message (Text)</td>
</tr>
<tr>
<td>SMTP</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery, and Integration</td>
</tr>
<tr>
<td>UDF</td>
<td>User Defined Function</td>
</tr>
<tr>
<td>UDT</td>
<td>User Defined Type</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>URN</td>
<td>Uniform Resource Names</td>
</tr>
<tr>
<td>VSAM</td>
<td>Virtual Storage Access Method</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>WMQI</td>
<td>WebSphere MQ Integrator</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schema Definition or XML Schema Language</td>
</tr>
</tbody>
</table>
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 359.

- Building the Operational Data Store on DB2 UDB, SG24-6513
- Integrating XML with DB2 XML Extender and DB2 Text Extender, SG24-6130-00
- The XML Files: Using XML for Business-to-Business and Business-to-Consumer Applications, SG24-6104-00
- The XML Files: Development of XML/XSL Applications Using WebSphere Studio Version 5, SG24-6586-00

Other resources

These publications are also relevant as further information sources:

- Cut Loose From Old Business Processes, John Hagel III and John Seely Brown, December 2001
- WebSphere MQ for Windows Quick Beginnings Version 5.3, GC34-6073
- WebSphere MQ for z/OS System Setup Guide, SC34-6052
- Application Development Guide: Building and Running Applications, SC09-4825
- IBM DB2 Universal Database SQL Reference Volume 1, SC09-4844
- IBM DB2 Universal Database SQL Reference Volume 2, SC09-4845
- Program Directory for WebSphere MQ for z/OS, GI10-2548
- WebSphere MQ for z/OS System Setup Guide, SC34-6052
- WebSphere MQ for z/OS System Administration Guide, SC34-6053
» WebSphere MQ Application Messaging Interface, SC34-6065
» IBM DB2 Universal Database Quick Beginnings for DB2 Servers, GC09-4836
» WebSphere MQ Application Programming Guide, SC34-6064
» WebSphere MQ Application Programming Reference, SC34-6062
» CICS Resource Definition Guide, SC34-5990
» DB2 XML Extender: Administration and Programming, SC27-1234-00
» An Introduction to Messaging and Queueing, GC33-0805
» WebSphere MQ Integrator Broker Introduction and Planning, GC34-6167
» WebSphere MQ Script (MQSC) Command Reference, SC34-6055

Referenced Web sites

These Web sites are also relevant as further information sources:

» IBM information integration Web site:

» MQSeries AMI SupportPac MA0F

» MQSeries Publish/Subscribe SupportPac MA0C

» Dynamic e-business with DB2 and Web Services

» Creating the Patient DADX Web service

» White paper on Web Services Object Runtime Framework: Implementing DB2 Web Services:
  http://www7b.software.ibm.com/dmdd/zones/webservices/worf/

» Running DB2 Web Services on WebSphere Application Server Advanced Edition 4.0

» Document Access Definition Extension (DADX) 1.0
The article “Cut Loose From Old Business Processes” can be found here
http://www.optimizemag.com/issue/002/pr_culture.fhtml

XML Protocol Working Group on W3C
http://www.w3.org/2000/xp/Group

Apache SOAP
http://www.apache.org/soap/

Web Services Description Language (WSDL) 1.1 on W3C
http://www.w3.org/TR/wsdl

UDDI.org home site
http://www.uddi.org

UDDI core information model specified in XML schema
http://www.uddi.org/schema/uddi_v3.xsd

Web services and UDDI
http://www-3.ibm.com/services/uddi/

Web services by IBM

WebSphere MQSeries family homepage

Getting started with MQSeries Pub/Sub SupportPac MA0D

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:
ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.
# Index

## Numerics

<table>
<thead>
<tr>
<th>Numeric</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>24x7</td>
<td>30</td>
</tr>
</tbody>
</table>

## A

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td>8, 14</td>
</tr>
<tr>
<td>acquisitions</td>
<td>3</td>
</tr>
<tr>
<td>additional data</td>
<td>33</td>
</tr>
<tr>
<td>administrator</td>
<td>82, 93, 100, 232</td>
</tr>
<tr>
<td>agent sites</td>
<td>249</td>
</tr>
<tr>
<td>aggregation</td>
<td>33, 108</td>
</tr>
<tr>
<td>AMI</td>
<td>32, 210, 256</td>
</tr>
<tr>
<td>AMI Administration Tool</td>
<td>246</td>
</tr>
<tr>
<td>analysis</td>
<td>6, 8</td>
</tr>
<tr>
<td>analytical</td>
<td>40</td>
</tr>
<tr>
<td>analytics</td>
<td>26–27</td>
</tr>
<tr>
<td>ANSI</td>
<td>36, 43</td>
</tr>
<tr>
<td>Apache SOAP</td>
<td>131</td>
</tr>
<tr>
<td>APIs</td>
<td>18, 24, 36, 58</td>
</tr>
<tr>
<td>application</td>
<td></td>
</tr>
<tr>
<td>connectivity</td>
<td>7, 14</td>
</tr>
<tr>
<td>development</td>
<td>16</td>
</tr>
<tr>
<td>external</td>
<td>9, 23</td>
</tr>
<tr>
<td>integration</td>
<td>3, 9, 27, 29–30</td>
</tr>
<tr>
<td>internal</td>
<td>9, 23</td>
</tr>
<tr>
<td>layer</td>
<td>18</td>
</tr>
<tr>
<td>server</td>
<td>21</td>
</tr>
<tr>
<td>Application Messaging Interface</td>
<td>32</td>
</tr>
<tr>
<td>application-to-application communication</td>
<td>48</td>
</tr>
<tr>
<td>architecture</td>
<td>49, 199, 221</td>
</tr>
<tr>
<td>ASCII</td>
<td>33, 95</td>
</tr>
<tr>
<td>assets</td>
<td>7, 14, 45</td>
</tr>
<tr>
<td>asynchronous</td>
<td>12, 23, 29–30, 199</td>
</tr>
<tr>
<td>attribute</td>
<td>35, 38</td>
</tr>
<tr>
<td>node</td>
<td>145</td>
</tr>
<tr>
<td>audit trail</td>
<td>109</td>
</tr>
<tr>
<td>automate</td>
<td>7, 63, 106, 225</td>
</tr>
</tbody>
</table>

## B

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2B</td>
<td>3, 9</td>
</tr>
<tr>
<td>B2C</td>
<td>3, 9</td>
</tr>
<tr>
<td>bandwidth</td>
<td>228</td>
</tr>
<tr>
<td>batch</td>
<td>78</td>
</tr>
<tr>
<td>bi-directional</td>
<td>13</td>
</tr>
<tr>
<td>bind</td>
<td>46, 234</td>
</tr>
<tr>
<td>binding</td>
<td>52, 57, 125</td>
</tr>
<tr>
<td>template</td>
<td>59</td>
</tr>
<tr>
<td>broker</td>
<td>13</td>
</tr>
<tr>
<td>registry</td>
<td>46</td>
</tr>
<tr>
<td>bus</td>
<td>11</td>
</tr>
<tr>
<td>business</td>
<td></td>
</tr>
<tr>
<td>analysis</td>
<td>5</td>
</tr>
<tr>
<td>channels</td>
<td>3</td>
</tr>
<tr>
<td>effectiveness</td>
<td>26</td>
</tr>
<tr>
<td>entity</td>
<td>59</td>
</tr>
<tr>
<td>events</td>
<td>4, 9, 14</td>
</tr>
<tr>
<td>integration</td>
<td>4, 7, 9, 48</td>
</tr>
<tr>
<td>logic</td>
<td>11, 31</td>
</tr>
<tr>
<td>models</td>
<td>3, 26</td>
</tr>
<tr>
<td>objectives</td>
<td>45</td>
</tr>
<tr>
<td>process</td>
<td>3, 7, 47</td>
</tr>
<tr>
<td>layer</td>
<td>18</td>
</tr>
<tr>
<td>services</td>
<td>23, 59</td>
</tr>
<tr>
<td>state</td>
<td>14</td>
</tr>
<tr>
<td>business-to-business</td>
<td>9</td>
</tr>
<tr>
<td>business-to-consumer</td>
<td>9</td>
</tr>
</tbody>
</table>

## C

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>36</td>
</tr>
<tr>
<td>headers</td>
<td>31, 33</td>
</tr>
<tr>
<td>C++</td>
<td>36</td>
</tr>
<tr>
<td>call</td>
<td>128</td>
</tr>
<tr>
<td>call center</td>
<td>216</td>
</tr>
<tr>
<td>capacity planning</td>
<td>116</td>
</tr>
<tr>
<td>CASE</td>
<td>231</td>
</tr>
<tr>
<td>catalog</td>
<td>234, 236</td>
</tr>
<tr>
<td>remote database</td>
<td>84–85</td>
</tr>
</tbody>
</table>
DCOM 48
decoding 231, 271
decomposition 36, 40, 153
default values 33
demographic data 217, 273
deployment 47–48, 172, 189
design 18
destructive read 42
development 48
discovery 48, 58
distributed 48–49
computing 6
Distributed Common Object Model 48
Distributed Objects Middleware 10
Document Access Definition 38
Document Access Definition Extension 21, 62
document oriented communication 51
documents 26
DTD 33, 35, 40, 107, 140, 151
DTD_REF 38
extranets 28

F
federated
data source 251
object 85
server 5, 106
view 89
federated access 20, 68, 225–226, 229, 235, 238, 253, 277
federated database 252
federated server 20, 83, 124, 137
federation 3, 5, 7, 19, 24, 28, 77
fire and forget 13
flat files 28, 222, 228, 262, 268
flow 109–110, 155, 274
format 22–23, 30, 33, 50, 96
foundation 5
framework 18, 45–46
FTP 11

G
group.properties 151
groupNamespaceUri 152

H
heterogeneous 44, 110
high-speed searching 22
historical 191
HLED 67
homogenous 106
HTTP 6, 11, 45, 47, 49–50, 57
hub-and-spoke 11

I
IBM DiscoveryLink 28
IBM Enterprise Information Portal 28
IBM WebSphere MQ 23
IDCAMS 100
implementation 82
incremental update 63
indexing 40
information
assets 8
integration 5, 7–9, 14, 29
Informix 20, 28, 77, 82, 86, 131, 229, 235, 238
Informix Client SDK 86, 238
Informix Connect 238
INFORMIXDIR 240
INFORMIXSERVER 240
infrastructure 17, 26, 48, 50, 68, 74, 228, 232
initialContextFactory 151
instance 84, 233
integration 14
   framework 9
types 13
inter-enterprises 44, 47
interchange
   XML data 22
inter-connectivity 14
interface 15, 49, 52
Internet xxi, 27, 47, 50, 125, 188
interoperability 6, 13, 47–48, 188
intra-enterprise 44, 47
intranets 28
ISO 36
isolate changes 67

J
J2EE xxi, 3, 13–14, 173, 182
jar 133, 138
Java 36, 200
JavaBeans 122
Java RMI 48
JDBC xxi, 25, 36, 131, 139, 197
JSPs 126
just-in-time integration 48

L
legacy 3, 25, 48
listener.ora 111
load 221, 225, 227
load balancing 191
local agent 248
local queue 30

M
maintenance 116, 277
manipulate
   XML documents 22
many-to-one relationship 35
mapping 38, 40–41, 127, 151, 180
   XML 22
mergers 3
message 23, 31, 52
   body 33
   content 33–34
delivery 24, 29
descriptor 33
enrichment 33
format 257
notification 3
queues 36
queuing 3, 23, 25, 222
Message Broker Middleware 11
message brokers 12
Message Oriented Middleware 10
metadata 25, 59, 64
   interchange 64
meta-information 6
metamodel 64
method 158–159, 171–172, 184
Microsoft SQL Server 28
middleware 5, 27
MIME 58
mining 3, 7, 26
MOM 10–11, 13
monitoring 4, 63, 116, 191
move 221, 270
MQ 82
   wrapper 32
MQ message descriptor 102
MQ resources 242
MQMD 102
MQPublish 44
MQPUT 109
MQRead 44
MQReadAll 44
MQReadXML 42
MQReadXMLAll 42
MQReadXMLAllCLOB 42
MQReadXMLCLOB 42
MQReceive 44, 102
MQReceiveAll 44
MQReceiveXML 42
MQReceiveXMLAll 42
MQReceiveXMLAllCLOB 42
MQReceiveXMLCLOB 42
MQSend 44, 101–102, 162, 199, 208, 212
MQSendXML 42, 199
MQSeries AMI 32, 75, 92, 226–227, 242
MQSeries Integration 31–32, 68, 75, 77, 80, 92,
   99–100, 124, 193, 195, 227, 235

364 Getting Started on Integrating Your Information
MQSeries Publish/Subscribe 34, 41
MQSeries SupportPac MA0C 30
MQSeries text functions 43
MQSeries Workflow 23
MQSeries XML functions 42–43
MQSubscribe 44
MQXML 99
multihub 11

MQSeries Publish/Subscribe 34, 41
MQSeries SupportPac MA0C 30
MQSeries text functions 43
MQSeries Workflow 23
MQSeries XML functions 42–43
MQSubscribe 44
MQXML 99
multihub 11

N
namespace 58
namespaceTable 151
navigate 38
near real time 26, 29–30
Net.Search 27
Net8 111
nickname 85, 89, 114, 237, 241
node 109, 236
nondestructive read 42
non-relational sources 20
NST 151
nst.xsd 133

O
object/method approach 11
objects 50
ODBC xxi, 25, 36, 197
OLAP Servers 63, 222
OMG 64
online analytical processing 28
optimizer 29
optimizing
  data access 5, 20, 80
Oracle 20, 28, 110–111, 277
Oracle Universal Installer 112
outages 30

P
performance 5, 20, 29, 40, 77, 116, 191, 195, 277
Perl 36
persistent 24
pervasive 28
petabytes 28
point to point 11
point-of-time 25
policy 32, 101–102, 210, 227, 245, 256
populate 63
portals 3–4
  integration 9
portfolio 69, 104
ports 52–53
prioritizing 191
procedure
  remote 50
process 9, 225, 231, 247, 270, 274
  deployment 274
  integration 3, 7, 9, 27
Process Builder 64, 270
production 20
project 172
propagate 6
provider 30
publish 3, 13, 23, 29–31, 34, 46

Q
QLOCAL 91
QoS 190
QREMOTE 91
Quality of Service 190
query 128, 205
  models 25
  rewrite 29
queue 23, 101
  manager 91, 101
  name 96
  remote 30

R
RDB node mapping 41, 143, 200
read/write access 25
real time 4, 25, 27–28, 30, 191
  read and write access 5
receiver 23, 102
Redbooks Web site 359
  Contact us xxiv
redundancy 106, 279
reference 6, 50
refresh 63
  frequency 228, 278
registry 58
  relational database 28, 222
  reliability 30, 48
reload Interval 151
remapping 33
Remote Method Invocation   48
Remote Procedure Call   10, 48
replication   6, 8, 20, 23–25, 28–29, 222
reply   13, 23, 30, 102, 108
request   30, 102, 108, 127, 162
requestor   30, 127
response messages   51
response time   26
responsiveness   190
restructuring   3
retrieveXML   128
REXX   36
routing   33
RPC   10, 48, 50–51, 54, 197
runtime environment   70, 72–73

S
scalability   48
Scheduler   64
scheduling   274
search   8, 20, 22, 26–28, 35, 58
security   117, 189, 249, 277
sender   23
sending
application   23
sequence   274
server   237, 240
broker   46
service
broker   48
Service Level Agreement   191
Service Oriented Architecture   13
service point   32, 101–102, 210, 227, 243, 256
service provider   46–47
service registry   46
service requester   46–47
service-oriented architecture   4
services   47, 53, 108
consuming   47
providers   189
provisioning   47
servlet   36, 126, 133, 152, 196, 200, 202, 213
Setnet32   239
share
data   22
sharing information   7
shred   36, 40, 67
skills   115

SLA   191
SMTP   11, 45
SOA   13–14, 45
SOAP   xxi, 6, 21, 49, 188
messages
body   50
headers   50
request   51
specifications   52
SOAP-ENV   51
spatial   28
speed   30
spreadsheets   28
SQL   xxi, 20, 25, 28, 36, 41, 197
operations   128
query   188
SQL Assist   64
SQL statement   77, 103, 177
wizard   174
SQL wizard   171
SQL/XML   25, 36, 43, 68, 193, 195, 198, 205
query   212
SQLHOSTS   87
SQLJ   36, 197
SQLPlus   112
standards   13, 34, 36, 43, 48, 67, 188
statistical analysis   28
steps   274
stocks   78
storage
models   25
stored procedure   13, 21, 24, 36, 80, 104, 127–128, 160, 188, 196, 201, 211
stored procedures   36
storeXML   128
storing
XML documents   35
streaming video   26
Subject Area   249
subscribe   3, 13, 23, 30–31, 34
Sybase   28
synchronization   6, 29, 31, 106, 278
synchronous   12, 195, 199
System Monitor   64

tagg   35, 40, 154, 167, 207
target   20


Getting Started on Integrating Your Information
Getting Started on Integrating Your Information

Integration solutions incorporate one or more elements of process, application, user interface, and information integration. Coping with the information explosion now requires information integration, a class of middleware which lets applications access data as though it were in a single database, whether or not it is.

This IBM Redbook shows, through examples, the various ways in which information integration can be implemented as a “pure play” or in conjunction with the other approaches for integration.

Using IBM DB2 Universal Database V8.1, we describe how to start implementing several information integration scenarios based on DB2 UDB V8.1 to federate data from multiple sources such as message queues with DB2 and Informix databases using SQL; to access your information using a DB2 Web services solution; to store, compose, and manipulate XML data; and to populate a common data store.

This book will help architects and implementers to understand the integration technologies in DB2 Universal Database and DB2 Information Integrator and to start implementing information integration solutions.

For more information: ibm.com/redbooks