

WebSphere MQ Solutions in a Microsoft .NET Environment

Invoking WebSphere MQ from a .NET
application

WebSphere MQ as a SOAP
transport mechanism

.NET and J2EE integration
using WebSphere MQ



Saida Davies
Michael Hamann
Sachin Kulkarni
Tony Shan
Andrew Sheppard
Ope-Oluwa Soyannwo
Jerry Stevens
Dong Kai Yu



International Technical Support Organization

**WebSphere MQ Solutions in a Microsoft .NET
Environment**

January 2004

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

First Edition (January 2004)

This edition applies to Version 5, Release 3, Modification 0 of WebSphere MQ; WebSphere MQ Customer Service Diskette (CSD) 05; Microsoft Visual Studio .NET Professional 2003 with Microsoft Development Environment 2003 Version 7.1.3088 and Microsoft .NET Framework 1.1 Version 1.1.4322; Internet Information Services (IIS) Version 5; WebSphere MQ Transport for SOAP, February 2003; and WebSphere MQ classes for .NET (amqmdnet.dll Version Resource 1.0.0.3)

© Copyright International Business Machines Corporation 2004. 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

Notices	xi
Trademarks	xii
Preface	xiii
The team that wrote this redbook	xiii
Become a published author	xvi
Comments welcome	xvii
Chapter 1. Introduction	1
Chapter 2. Overview	3
2.1 The aim	4
2.2 Technologies	5
2.2.1 .NET environment and C# programming language	5
2.2.2 J2EE	6
2.2.3 WebSphere MQ	6
2.2.4 WebSphere MQ classes for Microsoft .NET	6
2.2.5 What is SOAP?	6
2.2.6 WebSphere MQ Transport for SOAP	7
2.3 Usage scenarios	8
2.3.1 Usage scenarios: .NET application to .NET application	8
2.3.2 Usage scenarios: .NET application to J2EE application	8
2.3.3 Usage scenarios: .NET application to a .NET Web Service	9
2.3.4 Usage scenarios: .NET application to a J2EE Web Service	10
Chapter 3. WebSphere MQ Transport for SOAP	11
3.1 What is WebSphere MQ transport for SOAP?	12
3.2 WebSphere MQ transport for SOAP Installation	13
3.2.1 Downloading WebSphere MQ transport for SOAP	13
3.2.2 Prerequisite software	14
3.2.3 Pre-installation	15
3.2.4 Installation	17
3.2.5 Running the demonstration programs	17
3.2.6 Re-registration to the Global Assembly Cache	19
3.2.7 Checking the WebSphere MQ transport for SOAP release level ...	20
3.3 SOAP formatting	20
3.3.1 How to specify RPC or Document style encoding	22
3.4 WebSphere MQ transport for SOAP application development	23
3.4.1 Client environment	24

3.5	WebSphere MQ transport for SOAP .NET deployment	25
3.5.1	WebSphere MQ URI Syntax	29
3.5.2	WebSphere MQ client connection options	30
3.5.3	Calling deployWMQService	31
3.6	WebSphere MQ transport for SOAP listener for .NET	33
3.6.1	Executing MQSoapHost	35
3.7	A simple example with a Microsoft .NET Web Service	36
3.7.1	Write the Web Service	36
3.7.2	Write the .NET ASMX service directive file	38
3.7.3	Deploying the Microsoft .NET service	39
3.7.4	Write the client application	41
3.7.5	Define the WebSphere MQ response queue	43
3.7.6	Start the prepared Microsoft .NET listener	43
3.7.7	Test the service	44
3.7.8	Distributed test in WebSphere MQ client mode	45
3.7.9	Distributed test in WebSphere MQ server bindings mode	47
3.7.10	Distributed WebSphere MQ using MQ clustering	48
3.8	WebSphere MQ transport for SOAP with J2EE deployment	48
3.8.1	Deployment of J2EE Web Services	48
3.8.2	WebSphere MQ Transport for SOAP SimpleJMSListener	49
3.8.3	Executing SimpleJMSListener	52
3.9	A simple example with a J2EE Web Service	52
3.9.1	Write the Web Service	53
3.9.2	Deploy the service	54
3.9.3	Write the client application	55
3.9.4	Additional WebSphere MQ configuration	56
3.9.5	Start the prepared JMS listener	56
3.9.6	Test the service	57
3.9.7	Distributed test in WebSphere MQ server bindings mode	57
3.9.8	Distributed test in WebSphere MQ client mode	59
3.9.9	Distributed WebSphere MQ using MQ clustering	59
3.9.10	Service code use of external classes	59
3.10	Starting listeners with WebSphere MQ triggering	60
3.10.1	Using a different initiation queue	62
3.11	WebSphere MQ transport for SOAP and SSL	64
3.11.1	Simple demonstration with SSL	65
3.11.2	Use of SSLPeerName	66
3.12	Asynchronous invocation of Web Services	67
3.13	Current status and future plans	69
Chapter 4.	Business case scenario	73
4.1	Business domain	74
4.2	Business process	74

4.2.1 Use case 1: Account opening	75
4.2.2 Use case 2: Investment advisory	76
4.3 Non-functional requirements and assumptions	77
Chapter 5. Solution design	79
5.1 Message flow	80
5.1.1 Use case 1: Account opening message flow	80
5.1.2 Use case 2: Investment advisory message flow	81
5.2 Server configuration	82
5.3 WebSphere MQ configuration	83
Chapter 6. Environment setup	89
6.1 Software prerequisites	90
6.2 Installation	90
6.2.1 Installing WebSphere MQ	90
6.2.2 Installing WebSphere MQ classes for Microsoft .NET	90
6.2.3 Installing WebSphere MQ Transport for SOAP	90
6.2.4 Installing Internet Information Services (IIS)	90
6.2.5 Installing Microsoft Visual Studio .NET	93
6.3 Environment Setup	110
6.3.1 Core systems overview	110
Chapter 7. Messaging solution: .NET application to .NET application	115
7.1 Process overview	116
7.1.1 Scenario overview	117
7.2 System context	118
7.2.1 Interface definitions	118
7.3 Development	119
7.3.1 Adding the WebSphere MQ reference to the project	119
7.3.2 Bank service application (C#)	121
7.3.3 Credit check application	124
7.3.4 Credit check application C# snippet	125
7.3.5 Credit check application VB .NET snippet	129
7.4 Deployment	133
7.4.1 Deploying BSS	134
7.4.2 Deploying CCS	136
7.5 Testing	136
7.5.1 How to start BSS	136
7.5.2 How to start CCS	137
7.5.3 Test 1 pass known data	137
7.5.4 Test 2 pass unknown user	138
Chapter 8. Messaging solution: .NET application to J2EE application	139
8.1 Process overview	140

8.1.1	Account opening	140
8.1.2	Investment advisory	141
8.2	System context	142
8.2.1	Bank service application	143
8.2.2	Investment advisory application	144
8.2.3	Customer profile application	144
8.2.4	Database.	145
8.2.5	JMS administered objects.	146
8.3	Development	147
8.3.1	Bank service application	147
8.3.2	Investment advisory application	148
8.3.3	Customer profile application	150
8.4	Deployment	157
8.4.1	Deploying BSS	157
8.4.2	Deploying CPS	157
8.5	Testing.	159
8.6	Alternative solutions	161
8.6.1	WebSphere MQ classes for Microsoft .NET and WebSphere MQ classes for Java	161
8.6.2	Web Services	162
8.6.3	Bridge between WebSphere MQ and Microsoft Message Queuing (MSMQ)	162

Chapter 9. Messaging solution: .NET client to .NET Web Services using WebSphere MQ SOAP transport		163
9.1	Process overview	165
9.2	System context	166
9.3	Development	173
9.3.1	.NET Web Service development.	173
9.3.2	IAS Web Service solution	176
9.3.3	WebSphere MQ transport for SOAP deployment for IAS	185
9.3.4	BSS client	187
9.3.5	BSS Web Application solution.	192
9.4	Deployment	194
9.4.1	IAS Web Service deployment	194
9.4.2	WebSphere MQ queue setup and WebSphere MQ transport for SOAP deployment	196
9.4.3	BSS Web Application deployment	197
9.4.4	Securing the IAS Web Service	197
9.5	Testing.	199
9.5.1	IAS Web Service testing using Microsoft Visual Studio .NET	199
9.5.2	BSS user interface testing.	203

Chapter 10. Messaging solution: .NET client to J2EE Web Services using WebSphere MQ SOAP transport	207
10.1 Process overview	208
10.2 System context	208
10.2.1 Interface definition	208
10.2.2 Service operation definition	209
10.2.3 XML data format	209
10.3 Development	213
10.3.1 Business logic implementation	213
10.3.2 Persistent storage	215
10.3.3 WebSphere MQ definition	218
10.3.4 Adding external classes to the CLASSPATH	219
10.4 Deployment	219
10.4.1 Runtime environment	219
10.4.2 ShareQuote service deployment	220
10.5 Testing	222
10.5.1 Calling the service from the IAS client	222
10.5.2 Test result	224
10.6 Solution discussion	225
Chapter 11. System integration and functional test	227
11.1 Scope and objectives	228
11.2 System integration	228
11.2.1 Runtime environment	229
11.2.2 Test data	229
11.2.3 System build and deployment	229
11.2.4 System startup	230
11.3 Functional test	230
11.3.1 Entrance and exit criteria	230
11.3.2 Use case 1: Account opening	231
11.3.3 Use case 2: Investment advisory	241
11.4 Summary	248
Chapter 12. Security	249
12.1 Security concepts	250
12.1.1 Security services	250
12.1.2 Security mechanisms	251
12.2 Planning the security services in use cases	252
12.2.1 Application layer security services	252
12.2.2 Transmission layer security services	253
12.3 Cryptographic concepts	254
12.3.1 Cryptography	254
12.3.2 Message digest	256

12.3.3	Digital signature	257
12.3.4	Digital certificate	258
12.3.5	Public Key Infrastructure (PKI)	262
12.4	Secure Sockets Layer (SSL) introduction	262
12.4.1	Secure Sockets Layer(SSL) concepts	262
12.4.2	CipherSuites and CipherSpecs	263
12.5	WebSphere MQ SSL support	263
12.6	WebSphere MQ working with SSL on Windows	264
12.7	Deploy SSL support in use cases	265
12.7.1	Obtaining certificates	265
12.7.2	Deploying SSL support in CCS	271
12.7.3	Deploying SSL support in IAS	278
12.7.4	Deploying SSL support in BSS	278
Chapter 13.	Transactions	283
13.1	Local transactions	284
13.1.1	ACID properties of a transaction	284
13.1.2	Programming local transactions	285
13.2	Distributed transactions	290
13.2.1	Transaction support under Windows 2000	291
13.2.2	Programming distributed transactions: Credit Check Service	294
13.2.3	Microsoft Transaction Server: MTS and WebSphere MQ	300
13.3	Web Service transactions	300
13.3.1	.NET Web Services and transactions	301
13.3.2	Programming Web Services transaction in .NET environment	301
13.3.3	WS Transaction	303
Chapter 14.	Best practices	305
14.1	Coding standards	306
14.2	Hints and tips	306
14.2.1	XML style comments	306
14.2.2	XML processing in Java	307
14.2.3	SOAP processing in Java	307
14.2.4	XML element versus attribute	307
14.3	Common errors	307
14.4	Testing	313
14.4.1	Unit Testing with JUnit	313
14.4.2	Unit Testing with NUnit	313
14.4.3	Unit Testing with csUnit	314
14.5	Version management	314
14.5.1	ClearCase	314
14.5.2	Concurrent Versions System	315
14.5.3	Visual SourceSafe	315

Appendix A. Scripts, source code and test data for YuBank	317
WebSphere MQ Setup	318
Use case 1	321
Use case 2	322
Appendix B. Additional material	323
Locating the Web material	323
Using the Web material	323
System requirements for downloading the Web material	324
How to use the Web material	324
Glossary	329
Abbreviations and acronyms	331
Related publications	333
IBM Redbooks	333
Other publications	333
Online resources	334
How to get IBM Redbooks	335
Help from IBM	335
Index	337

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:

AIX®
ClearCase®
CICS®
DB2®
Everyplace®
IBM®
ibm.com®

Lotus Notes®
Lotus®
MQSeries®
Notes®
Parallel Sysplex®
Rational®
Redbooks™

Redbooks (logo) ™
RS/6000®
SupportPac™
TXSeries®
WebSphere®
z/OS®

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX, and Pentium 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.

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

This IBM® Redbook demonstrates the use of WebSphere MQ in a .NET environment as a reliable transport mechanism for the invocation of .NET applications and as a middleware product used in the implementation of Web Services.

WebSphere® MQ now offers the programmer more choices than ever in which to write new WebSphere MQ applications. With the advent of the Microsoft® .NET platform, two new programming interfaces are explored by this redbook team.

First we introduce the two mechanisms for programming WebSphere MQ in a Microsoft .NET environment. These are *WebSphere MQ classes for Microsoft .NET* and *WebSphere MQ Transport for SOAP* for interfacing to Web Services. Because J2EE interoperability is of such strategic importance, the team considers these four scenarios:

- ▶ Using WebSphere MQ classes for Microsoft .NET
 1. .NET application to .NET application
 2. .NET application to J2EE application
- ▶ Using WebSphere MQ Transport for SOAP
 3. .NET application to .NET Web Service
 4. .NET application to J2EE Web Service

Next, we introduce a banking example which is used as a business case scenario to provide a realistic view of organizations with requirements exemplified in this book. The business case scenario is specifically contrived to illustrate the above four scenarios with sample code, but at the same time, an attempt is made to avoid any fictitious solutions.

We conclude the book with best practices and an appendix detailing source code and scripts that were used to create our environments.

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, Hursley Center.



*The authors – Top row from left: Andrew, Jerry, Sachin, Michael
Bottom row from left: Tony, Ope, Saida, Dong Kai*

Saida Davies is a Project Leader for ITSO Hursley UK. Saida is a Certified Senior IT Specialist and has 14 years of experience in IT. She has a degree in Computer Science and her background includes z/OS® systems programming. Saida has extensive knowledge of the IBM z/OS operating system and a detailed working knowledge of both IBM and Independent Software Vendors' operating system software. In a customer facing role with IBM Global Services, she was engaged in the development of services for MQSeries® within the z/OS platform. This covers the architecture, scope, design, project management and implementation of the software on stand-alone systems or on systems in a Parallel Sysplex® environment. One of her major projects was WebSphere MQ security review and planning for a bank in Portugal in line with the British Standard code of practice for information security management. Additional experience includes project management, pre-sales support, training, system and subsystems migrations.

Michael Hamann is an IT Specialist at IBM ITS Education Services in Germany. He has four years of experience in the field of Information Technology. During this time with IBM he worked as an Instructor, Course Developer, and Consultant for WebSphere MQ and WebSphere MQ Everyplace®. Michael is a Certified WebSphere MQ Solution Developer and a Certified WebSphere MQ Solution Designer. He holds a degree in Geography from the University of Tübingen, Germany.

Sachin Kulkarni is a Research Staff Member at Distributed Systems Technology Centre – DSTC, Australia, where he is responsible for investigating next generation distributed technologies and e-business architectures. Sachin's areas of expertise are XML Web Service design, .NET solutions development, and business process choreography. He has a strong professional background in software design, development, and project management. He has published and presented his research at various Australian and international conferences including topics, such as electronic contracts framework for e-business federation, and solution patterns for IBM WebSphere with Microsoft .NET integration. Also, he authored a comprehensive analysis report on Microsoft BizTalk Server technologies. Sachin has master's degree in Information Technology from Queensland University of Technology, and was one of the early achievers of Microsoft .NET Certifications in Australia.

Tony Shan is a Lead Systems Architect in Wachovia Bank in the United States. He has over 18 years experience in Information Technology. Tony has led life cycle design and development of large scale distributed systems on diverse platforms using a variety of cutting-edge technologies and object-oriented methodologies. Tony holds three master's degrees in Engineering and Science majors, and is a Sun Certified Java™ 2 Programmer and an IBM Certified eBusiness Solution Technologist, as well as a Sun Certified Faculty Instructor.

Andrew Sheppard is a Middleware Developer for IBM Global Services Australia. He has 33 years of experience in software development, including 13 years of software development in Microsoft Windows®. He holds a degree in Electrical Engineering with second class Division A honours from the University of Queensland, St. Lucia Campus, and is a Microsoft Certified Solution Developer. Prior to joining IBM Global Services in 1997, he spent 26 years working for Telstra. His areas of expertise include graphical user interfacing and all facets of embedded systems.

Ope-Oluwa Soyannwo is a Developer and Tester in the WebSphere Platform System House, IBM Hursley. She chose to intercalate from her degree, Computer Systems Engineering at the University of Hull, to get some practical experience. Prior to joining IBM on an internship program, Ope worked on several projects involving end-to-end solutions design, application development and testing. She has experience with testing interoperability between Microsoft

.NET and IBM WebSphere products. Her areas of expertise include .NET and J2EE Web Services, WebSphere MQ and Lotus® Notes® application development.

Jerry Stevens graduated from Exeter University with a first class honours degree in Mathematics and has 24 years of IT experience. He currently works in the WebSphere MQ Technical Strategy and Planning group at IBM UK Hursley Labs, where he is helping to develop Web Services facilities for WebSphere MQ. Prior to joining Hursley Labs, Jerry worked in an RS/6000® and SP consultancy practice for IBM UK Global Services in a customer facing role as an AIX® Consultant. He is also one of the authors of the IBM Redbook *Sizing and Tuning GPFS*. Prior to joining IBM in 1997, Jerry worked for Shell as a Senior Systems Engineer, where he undertook a range of technical consultancy and development roles and worked with a variety of Open Systems platforms and architectures.

Dong Kai Yu is an IT Specialist of IBM China. He has five years experience in the IT industry and is an IBM Certified Specialist on MQSeries. He has been working as Technical Sales Support for IBM China on WebSphere MQ, WebSphere Business Integration, TXSeries®, and Visual Age C/C++ for two years, and is the skill owner of WebSphere Business Integration Adapter. Prior to joining IBM, Dong Kai worked as an Application Developer on Microsoft Visual Studio for more than three years. He obtained his master's degree in Computer Science from Beijing Institute of Technology, P.R.C.

This redbook team would like to thank the following people for their contributions to this project:

Stephen Todd, IBM Hursley, for his invaluable support and help with WebSphere MQ classes for Microsoft .NET and WebSphere MQ Transport for SOAP technology.

Mike Bailey, IBM Hursley, for his invaluable support and help with MA7P SupportPac™ and WebSphere MQ classes for Microsoft .NET technology.

Jerry Stevens, IBM Hursley, for his invaluable support and help with MA0R SupportPac and WebSphere MQ Transport for SOAP technology.

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

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. HZ8 Building 662
P.O. Box 12195
Research Triangle Park, NC 27709-2195



Introduction

IBM WebSphere MQ is market leading business integration software. It connects business software together by providing an open, scalable, and an industrial strength messaging backbone. WebSphere MQ provides core capabilities needed to interconnect client and/ or server applications and serves as critical middleware technology, linking multiple applications together regardless of the software platforms on which they reside.

Microsoft .NET is a software platform which allows developers to write, build, and test .NET applications and Web Services. It contains its own messaging service called Microsoft Message Queuing (MSMQ) for Windows developers.

Java 2 Platform, Enterprise Edition (J2EE) is an open standard Java platform which defines the infrastructure for enterprise applications and supports Web Services to enable development of secure, robust and interoperable business applications.

WebSphere MQ, IBM messaging service, can be incorporated into the .NET environment to ensure a reliable request and delivery mechanism between .NET applications, .NET applications and J2EE applications, .NET Web Services invoked by .NET or Java applications and also J2EE applications invoked by .NET applications or Java.

Web Services are becoming the platform for application integration. They can be referred to as fundamental building blocks in the move to distributed computing on the Internet. Applications are constructed using multiple Web Services from

various sources and are allowed to work together regardless of where they reside or how they are implemented. Interoperability of the Web Services tools on different platforms are demonstrated by the successful invocation of the Web Services among the different platforms.

Web Services use Simple Object Access Protocol (SOAP) to send messages to one another. SOAP is a mechanism for a program running in one kind of operating system to communicate with a program in the same or alternate operating system. SOAP is designed as a transport neutral protocol to be used in combination with a variety of transport protocols such as Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP) and so on to deliver structured and typed information between two participants. A great amount of work has gone into making Web Services support SOAP message generation in a way that promotes interoperability within heterogeneous environments (for example, .NET applications consuming IBM WebSphere Application Server hosted Web Services).

The default transport mechanism for the delivery of SOAP messages to an endpoint is over HTTP. However, a known weakness of HTTP is the lack of guaranteed and reliable (single message only) delivery within the protocol. HTTP also requires the back end service to be physically on line for any request to be received.

By substituting HTTP with another message delivery option, a layer of proven security that HTTP provides is absent, therefore other security issues have been addressed, for example, security provided within WebSphere MQ.

WebSphere MQ illustrates a store and forward capability for a secure, assured and reliable request delivery of messages within the .NET environment and from the .NET environment to other environments.



Overview

Prior to the Microsoft .NET environments, the normal method of programming WebSphere MQ was by using the C, C++, COM or Java API. With the advent of .NET, which is both a language and platform neutral environment, a different approach is needed.

2.1 The aim

The aim of this redbook is to demonstrate the use of WebSphere MQ as a middleware product for application to application transport in a .NET environment. It also demonstrates WebSphere MQ as a transport mechanism for the invocation of a Web Service by modifying the Simple Object Access Protocol (SOAP) wrapper for the Web Service thereby communicating via WebSphere MQ instead of Hypertext Transfer Protocol (HTTP). Another aim of this redbook is to highlight the ability to effectively switch transports underneath a .NET client between HTTP, Microsoft Message Queuing (MSMQ) and WebSphere MQ. It demonstrates how trivial it is to alter the “target” of a SOAP Web Service request from a standard Internet Information Services (IIS) hosted variant, to a listener waiting on a WebSphere MQ queue. Sample applications in a Microsoft .NET environment for interfacing to WebSphere MQ are provided.

Finally, security over WebSphere MQ using Secure Sockets Layer (SSL) is implemented.

The team demonstrates the utilization of WebSphere MQ Application Programming Interface (API), using standard types like strings, arrays, binary data and custom serializable types, and also shows how this support can be achieved in an asynchronous fashion.

The team also recommends that before undertaking any development the latest Customer Service Diskette (CSD) should be obtained.

The four scenarios considered are illustrated in Figure 2-1.

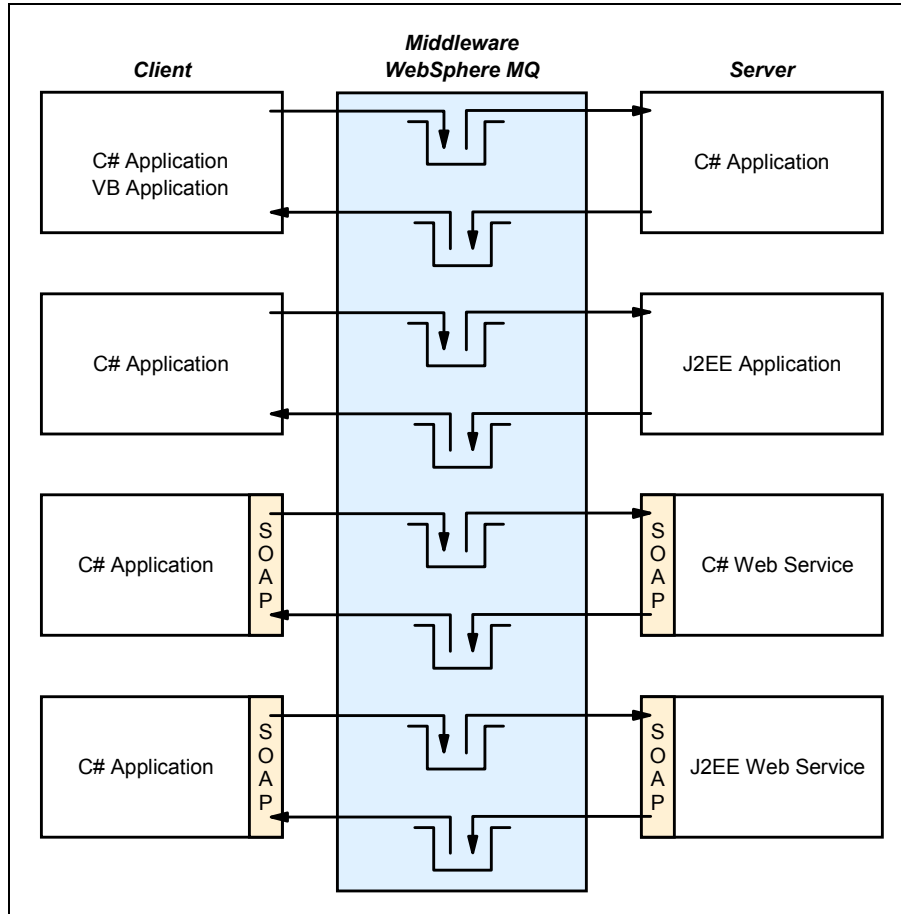


Figure 2-1 Scenario overview

2.2 Technologies

A number of technologies are involved in our business case scenario.

2.2.1 .NET environment and C# programming language

The sample codes for the four scenarios are written in C#. For readers familiar with Visual Basic (VB), there are significant changes when moving from VB to Visual Basic.NET that many consider changing directly to C#. However, the first scenario is coded in both C# and VB.NET for completeness. The other reason for

using C# is so that it requires only a minor adjustment for both Java and C++ programmers.

The following URL highlights the C# initiatives.

<http://www.ecma-international.org/>

Microsoft Visual Studio .NET is the chosen development environment.

2.2.2 J2EE

J2EE technology, and its component based model, simplifies enterprise development and deployment. The J2EE platform manages the infrastructure and supports the Web Services to enable development of secure, robust and interoperable business applications. J2EE interoperability with .NET is considered an important part of this redbook.

2.2.3 WebSphere MQ

WebSphere MQ is the market-leading business integration software which connects all your business software together to form one efficient enterprise by providing an open, scalable, industrial-strength messaging backbone.

WebSphere MQ minimizes time taken to integrate key resources and applications held in different systems, so your company can respond to the changing demands of e-business. By connecting business information with people and other applications, you can extract more value from existing investment, and quickly integrate new systems to support new market strategies.

2.2.4 WebSphere MQ classes for Microsoft .NET

To program WebSphere MQ from C# or VB .NET and so on, it is essential to install the .NET classes for Microsoft .NET. These are provided in WebSphere MQ classes for Microsoft .NET that allows WebSphere MQ to be invoked from Microsoft .NET applications. These classes were available as a category 2 (freeware) SupportPac, but with effect from WebSphere MQ V5.3 Customer Service Diskette (CSD05), the WebSphere MQ classes for Microsoft .NET are a fully incorporated in the WebSphere MQ product.

2.2.5 What is SOAP?

Simple Object Access Protocol (SOAP) is a lightweight format and protocol for exchange of information in a decentralized, distributed environment. It is an eXtensible Markup Language (XML) based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and

how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses

2.2.6 WebSphere MQ Transport for SOAP

Web Services have evolved as an increasingly important method for businesses to provide electronic services on demand. These are most commonly based on SOAP format messages and the Hypertext Transfer Protocol (HTTP) transport protocol. It is not however necessary to rely on HTTP as a transport, the limitations of which are already well documented. WebSphere MQ can be used as an alternative transport to HTTP in order to provide enhanced reliability and enable re-use of an existing WebSphere MQ infrastructure.

WebSphere MQ is supplied with a pluggable, reliable transport for SOAP messages that may be used in Web Service applications. At the time of writing this book, the WebSphere MQ Transport for SOAP is provided as a category 2 (freeware) SupportPac. We expect this to change as the intention is to incorporate this function in the WebSphere MQ product. Refer to 3.2, “WebSphere MQ transport for SOAP Installation” on page 13 for further detail.

The intention is to enable customers to make ready use of their existing investment in a WebSphere MQ infrastructure to provide Web Services. These are typically created using application development environments provided by IBM and others. In this redbook, we focus particularly on the use of Microsoft Visual Studio .NET.

WebSphere MQ does not undertake any SOAP formatting or parsing itself but instead relies on a host Web Services environment. This environment can be based either on the Apache Axis or the Microsoft .NET Framework. Although the use of SOAP formatting is not mandatory, it is the most commonly used format for transporting messages for Web Services. It is important that client applications and target Web Services understand the same versions and dialects of SOAP for which several different variants have already emerged. The use of WebSphere MQ transport for SOAP automatically enforces adherence to new SOAP standards as they are endorsed without the need for customers to have a low level understanding of the construction of individual SOAP messages.

For more detailed information about WebSphere MQ transport for SOAP, refer to:

- ▶ Chapter 3, “WebSphere MQ Transport for SOAP” on page 11
- ▶ Chapter 8, “Messaging solution: .NET application to J2EE application” on page 139

- Chapter 9, “Messaging solution: .NET client to .NET Web Services using WebSphere MQ SOAP transport” on page 163

2.3 Usage scenarios

There are four scenarios that are illustrated in Figure 2-1. Each one is now expounded as a separate scenario.

2.3.1 Usage scenarios: .NET application to .NET application

This is the first scenario because without the ability to write a .Net application it is not possible to proceed.

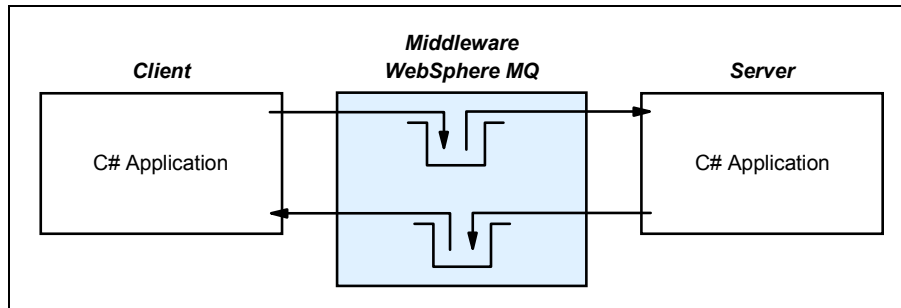


Figure 2-2 .NET to .NET

To write a C# application to interface to WebSphere MQ, it is necessary to use WebSphere MQ classes for Microsoft .NET.

2.3.2 Usage scenarios: .NET application to J2EE application

J2EE interoperability is of such strategic importance that this scenario is considered next.

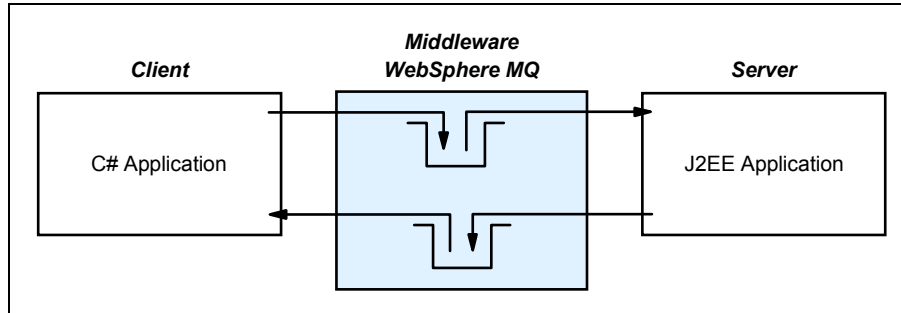


Figure 2-3 .NET to J2EE

J2EE application programming is exemplified here and a sample is provided in sufficient detail to illustrate the flexibility and interoperability that WebSphere MQ provides.

2.3.3 Usage scenarios: .NET application to a .NET Web Service

The World Wide Web (WWW) is used more and more for application to application communication. The programmatic interfaces made available for this are referred to as Web Services. The goal of this scenario is to highlight the use of this technology in a .NET environment and illustrate, by way of an example, a practical use of a Web Service.

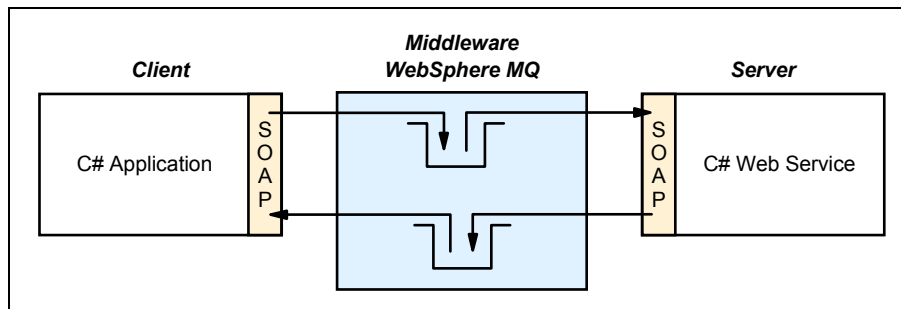


Figure 2-4 .NET to .NET Web Service

To write a C# application to interface to WebSphere MQ using the SOAP transport, it is necessary to use WebSphere MQ Transport for SOAP. This provides the ability to flow a SOAP message over a WebSphere MQ transport embedded in the Microsoft .NET Framework.

2.3.4 Usage scenarios: .NET application to a J2EE Web Service

The goal of this scenario is to highlight the use of WebSphere MQ technology in a .NET environment and illustrate, by way of an example, a practical use of how it interfaces to a J2EE Web Service.

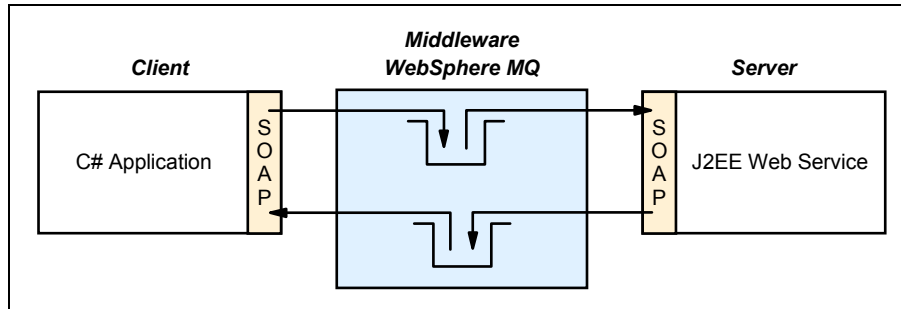


Figure 2-5 .NET to J2EE Web Service

Developing a J2EE Web Service is illustrated here and a sample is provided in sufficient detail to illustrate the flexibility and interoperability that WebSphere MQ provides.

To write a C# application to interface to WebSphere MQ using the SOAP transport, it is necessary to use WebSphere MQ Transport for SOAP. This provides the ability to flow a SOAP message over a WebSphere MQ transport embedded in the Microsoft .NET Framework.



WebSphere MQ Transport for SOAP

This chapter gives a detailed overview of WebSphere MQ transport for SOAP. The topics covered are:

- ▶ What is WebSphere MQ transport for SOAP?
- ▶ WebSphere MQ transport for SOAP installation
- ▶ SOAP formatting
- ▶ The WebSphere MQ URI syntax
- ▶ WebSphere MQ transport for SOAP development and deployment
- ▶ The WebSphere MQ transport for SOAP listener for Microsoft .NET Web Services
- ▶ Example use with a Microsoft .NET client and Web Service
- ▶ Use with Microsoft .NET clients and J2EE Web Services
- ▶ Example use with a Microsoft .NET client and a J2EE Web Service
- ▶ Use of WebSphere MQ triggering to start listeners
- ▶ WebSphere MQ transport for SOAP and SSL
- ▶ Current status and future plans

3.1 What is WebSphere MQ transport for SOAP?

WebSphere MQ transport for SOAP provides a transport for SOAP formatted messages that are used in conjunction with Web Services. It is implemented for either Apache Axis or Microsoft .NET host environments.

Figure 3-1 illustrates where WebSphere MQ transport for SOAP fits into an overall Web Service design. This illustrates a process where a client application calls a target Web Service with specific parameters and then obtains a response from the service.

First consider the case where HTTP is being used as a transport between the client application and target Web Service. The client passes the details of the required call to the SOAP layer, which prepares a request for invocation of the service as a SOAP formatted request. This request is dispatched to the server system, where an HTTP server receives the request and passes it through the particular SOAP layer for decoding and invocation of the service. The response message may be returned to the client either synchronously or asynchronously.

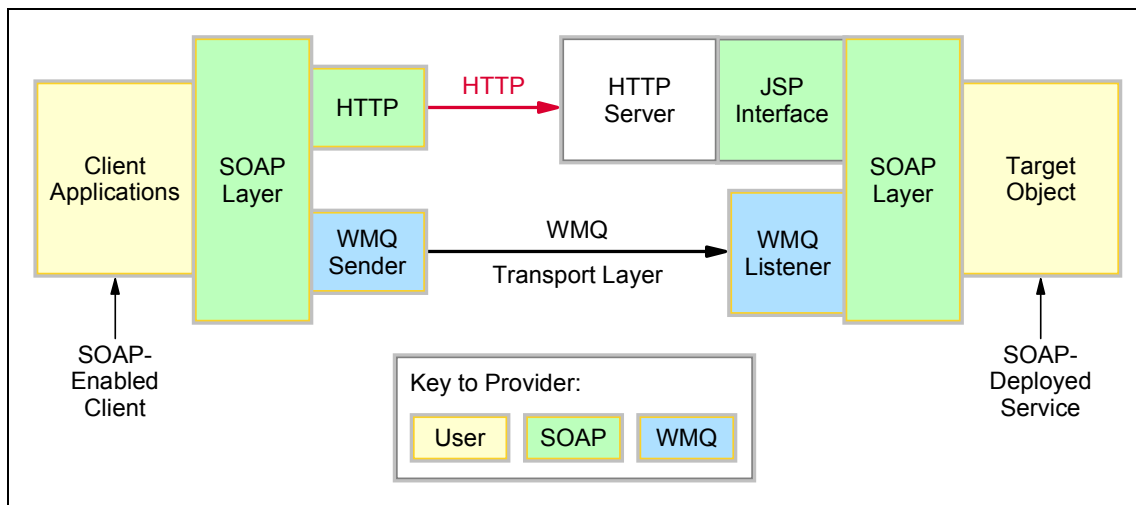


Figure 3-1 Overview of WebSphere MQ transport for SOAP

WebSphere MQ transport for SOAP provides an alternative transport to HTTP as shown in the diagram. This transport offers enhanced reliability compared to HTTP. It also permits solutions to be integrated readily within an existing WebSphere MQ infrastructure.

To prepare applications for this alternative transport, it is necessary to “plug-in” the transport to the clients environment via a special transport registration call. This transport registration will cause the invocation of sender software specific to

the WebSphere MQ transport when a Web Service is invoked with a “wmq:” prefix in the target URI. This sender software will write a SOAP request for invocation of the service to a WebSphere MQ request queue. A special service listener will retrieve messages from the request queue, invoke the required service and then pass any response back to the client via a WebSphere MQ response queue. Two service listeners are provided with WebSphere MQ transport for SOAP, one for Microsoft .NET services and one for J2EE services.

Target Web Services need to be processed through a series of deployment steps. These define the target service to the host infrastructure, generate proxy methods to simplify the process of invoking the service from the client, prepare a script file to start one of the service listeners and perform some queue and process configuration within WebSphere MQ.

As with the HTTP case, messages continue to be sent to and from target services in SOAP format. The WebSphere MQ transport does not undertake any of this formatting itself but instead relies on the Microsoft .NET or Apache Axis host Web Services environment.

The use of SOAP formatting for Web Services message is not mandatory, but it is the format assumed in WebSphere MQ transport for SOAP. More accurately, SOAP is the format used by the Microsoft .NET and Apache Axis SOAP engines that are employed by WebSphere MQ transport for SOAP. SOAP is the most common message format used in conjunction with Web Services.

3.2 WebSphere MQ transport for SOAP Installation

This section provides general information regarding the installation and use of WebSphere MQ transport for SOAP and supplements the documentation provided with the software.

The validity of the information in this section may change once WebSphere MQ transport for SOAP is supported. Only issues pertaining to the Windows operating system are identified here in accordance with the focus of this IBM Redbook.

3.2.1 Downloading WebSphere MQ transport for SOAP

The SupportPac is available for download from Internet from the following URL:

<http://www-3.ibm.com/software/integration/support/supportpacs/individual/ma0r.html>

The distribution is packaged as a zip file which is approximately 5 Mb in size.

3.2.2 Prerequisite software

At the time of writing this redbook, the prerequisites for Windows platforms are as follows:

- ▶ Windows 2000 + Service Pack 2, or Windows XP Professional or above
- ▶ WebSphere MQ Version 5.3 CSD05 or above
- ▶ Java™ 2 SDK and Runtime environment, Standard Edition (build 1.3.1 or above)
- ▶ Microsoft Internet Information Services (for running .NET services) is required on Windows 2000 platforms
- ▶ Microsoft .NET Framework redistributable V1.1 or above
- ▶ Microsoft .NET Framework SDK V1.1 or Microsoft Visual Studio .NET 2003 (for deploying Microsoft .NET services)
- ▶ Utility for extracting ZIP format files

The following sections expand more on some of these prerequisite requirements as well as providing other supplemental information concerning installation.

Note: The most up to date details on prerequisite software for WebSphere MQ transport for SOAP are detailed in the documentation. This should be checked carefully prior to installation in case of any changes to the list above.

Microsoft Internet Information Services (IIS)

For Windows 2000 systems it is necessary to have installed IIS to be able to deploy and run Microsoft .NET services. It is not necessary for it to remain installed though if target services will be using WebSphere MQ transport only. But it is necessary that IIS has at least been installed at some stage previously before making an actual service deployment.

If the Microsoft .NET framework has been installed before IIS, it will be necessary to use the “aspnet_regiis” utility to register IIS to the framework. Although the location of the aspnet_regiis.exe utility may vary with different versions of Microsoft .NET framework, it is typically located in:

%SystemRoot%\Microsoft.NET\Framework\<version number>\aspnet_regiis -i

If multiple versions are installed, use only the executable in the target version's directory.

Note: It is not necessary for IIS to have been installed in this way on Windows XP systems.

For more information about the system requirements for Microsoft .NET, refer to:
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconnetframeworksystemrequirements.asp>

Microsoft .NET Framework and SDK

On systems where it is desired to deploy and run Microsoft .NET services and clients, it is necessary to have installed either Microsoft Visual Studio .NET or the Microsoft .NET Framework together with the Framework SDK.

3.2.3 Pre-installation

The following section discusses some pre-installation issues.

Prerequisite software installation order

It is essential that prerequisite software for WebSphere MQ Transport for SOAP is installed in the following order:

1. Internet Information Server (IIS) (not mandatory on Windows XP)
2. Microsoft .NET Framework
3. Microsoft .NET Framework SDK or Microsoft Visual Studio .NET 2003.
4. WebSphere MQ CSD05 or later (if not already installed). This can be installed before the above items but if it has not been it should be installed before the following.
5. WebSphere MQ Transport for SOAP

The “ma0r_netdir” environment variable

WebSphere MQ transport for SOAP uses the Microsoft .NET gacutil utility to register the MQSOAP.dll and two of its own executable programs. It will also register the WebSphere MQ classes for Microsoft .NET provided with CSD05 (amqmdnet.dll) if it has not already been registered to the Global Assembly Cache.

The location of gacutil can vary with different versions of the Microsoft .NET redistributable framework, so (as at the time of writing this redbook) an environment variable called ma0r_netdir can be set to change the internal definition of the utility's location. This environment variable should be set if it is required to specify a location for gacutil other than the default location as used in the Microsoft .NET Framework SDK V1.1. This default location is %ProgramFiles%\Microsoft.NET\SDK\v1.1\Bin\gacutil. For example, users of Microsoft Visual Studio .NET 2003 would normally want to set ma0r_netdir as follows:

Example 3-1 Setting the “ma0r_netdir” environment variable

```
set ma0r_netdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin
where %ProgramFiles% is:
C:/Program Files
```

If the script regenDemos.bat is used to regenerate the supplied demonstration programs then re-registration of the DLLs will be forced. For this reason it is more reliable to place the definition of this environment variable permanently into the Windows environment. On Windows 2000, this can be accomplished by clicking on “My Computer”, then right-clicking on properties, selecting “environment variables”, and clicking “New” in the user variables pane and entering the variable definition.

The MQJAVA_INSTALL_PATH environment variable

WebSphere MQ transport for SOAP uses the environment variable MQJAVA_INSTALL_PATH to locate the WebSphere MQ JMS libraries. This is normally set permanently into the environment by the WebSphere MQ installation process. Before proceeding it should be verified that this variable is set and points to the correct location. If for some reason the variable is not set, the default install location for the libraries is assumed:
(%ProgramFiles%\IBM\WebSphere MQ\Java)

Java SDK version

WebSphere MQ transport for SOAP requires the availability of a Java SDK, even where it is only required to work with Microsoft .NET Web Services and from non Java clients. This is primarily because both C# and Java proxy code is generated by the deployment process, even when it is only required to build C# clients. The Java compiler is also used by the script regenDemo.bat to regenerate Java sample programs. The location of the SDK’s bin directory must be added to the path. As a final check, it is worth ensuring you can access both the java and the javac commands:

Example 3-2 Verifying access to the java and javac commands

```
C:\>java -version
java version "1.3.1"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.3.1)
Classic VM (build 1.3.1, J2RE 1.3.1 IBM Windows 32 build cn131-20020710 (JIT enabled: jitc))

C:\>java
Usage: java [-options] class [args...]
           (to execute a class)
    or  java -jar [-options] jarfile [args...]
           (to execute a jar file)
```

where options include:

```
-cp -classpath <directories and zip/jar files separated by ;>  
    set search path for application classes and resources  
-D<name>=<value>  
    set a system property  
-verbose[:class|gc|jni]  
    enable verbose output  
-version    print product version  
-showversion    print product version and continue  
-? -help    print this help message  
-X          print help on non-standard options
```

C:\>

3.2.4 Installation

The zip file downloaded in 3.2.1, “Downloading WebSphere MQ transport for SOAP” on page 13 should be unzipped in any required directory.

Having unzipped the distribution it is now necessary to run the demonstration programs. This will prepare the WebSphere MQ environment needed for the programs the first time it is used.

Before proceeding to run the standard demonstrations, we recommend that the the documentation for the distribution is reviewed. This is located in the docs sub-directory underneath the installation directory.

3.2.5 Running the demonstration programs

An IVT (Independent Verification Test) system is provided with WebSphere MQ transport for SOAP. This is the easiest way to run through the set of supplied and pre-prepared set of demonstration programs. It will also ensure the environment is correctly set up for the demonstrations after installation, including the necessary registration to the Global Assembly Cache with the gacutil utility. It is therefore recommended that the IVT utility is used to confirm correct installation and environment configuration before developing any custom applications.

The IVT is executed by using the runivt.bat script that is located in the demos directory. To run through the full set of tests, enter the command “runivt” with no arguments. An example run is given below. Some of the tests have been removed from the log for brevity:

Example 3-3 Sample execution of the runivt utility

C:\ma0r\demos>runivt

```
define channel(TESTCHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP) REPLACE completed OK.
amqmdain start MQSOAP.DEMO.QM completed OK.
```

```

----- [AxisSimple] -----
WMQ transport test: Axis to Axis, using simple Axis calls
+++ server: helpers\listen_SOAP.javaDemos.server.StockQuoteAxis
--- client: java javaDemos.clients.SoapClient -u
wmq:SOAP.javaDemos.server.StockQuoteAxis?connectQueueManager=MQSOAP.DEMO.QM -n
javaDemos.server.StockQuoteAxis_wmq

```

```
start SoapClient demo to:
    wmq:SOAP.javaDemos.server.StockQuoteAxis?connectQueueManager=MQSOAP.DEMO.QM
This request is synchronous.
Response: 55.25
OK.
```

```
----- [AxisSimpleOneWay] -----
WMQ transport test: Axis to Axis, using simple Axis calls to a one way service method
--- client: java javaDemos.clients.SoapClientOneWay -u
wmq:SOAP.javaDemos.server.StockQuoteAxis?connectQueueManager=MQSOAP.DEMO.QM -n
javaDemos.server.StockQuoteAxis_Wmq
```

```
start SoapClient demo to:
  wmq:SOAP.javaDemos.server.StockQuoteAxis?connectQueueManager=MQSOAP.DEMO.QM
Sending one way
Waiting for a results file to be generated...
One way service result is: 55.25
OK.
```

-
-
- <<< OUTPUT removed for brevity >>>
-
-
-

```

----- [Axis2DotNetWsd1] -----
WMQ transport test: Axis to .NET (Asmx), using WSDL Axis calls
--- client: java javaDemos.clients.Wsd1Client -D

start Wsd1Client demo, wsd1 port StockQuoteDotNetSoap resolving uri to ...
'wmq:SOAP.StockQuoteDotNet@MQSOAP.DEMO.QM?connectQueueManager=MQSOAP.DEMO.QM'
This request is synchronous.
Response: 88.88
OK.

```

```
----- [Axis2DotNetProxy] -----
```

```
WMQ transport test: Axis to .NET (Asmx), using WSDL Axis calls
--- client: java javaDemos.clients.SQAxis2DotNet
```

```
This request is synchronous.
```

```
Response: 88.88
```

```
OK.
```

```
-----
```

```
Service SimpleJMSListener: kill request accepted.
```

```
Service DotNet: kill request accepted.
```

```
endMQ.bat not implemented for Windows ... MQ left running.
```

```
=====
```

```
22 tests run, of which 0 failed.
```

By default, runivt starts the various listeners it requires during the tests and then closes them down at the end of its execution. If it is required to leave the listeners running after runivt has completed, the “hold” option should be specified on the runivt command line.

The IVT utility uses a configuration file called ivttests.txt that details the various tests to be performed. If required, it is possible to create different configuration files detailing different sets of tests. This can be accomplished by use of the “-c <filename>” option to the runivt script. We recommend that different configuration files are created for customized tests in preference to editing the default file.

For more details on the IVT, refer to the documentation provided with WebSphere MQ Transport for SOAP.

3.2.6 Re-registration to the Global Assembly Cache

As already mentioned, it is necessary for certain DLLs and EXEs used by WebSphere MQ transport for SOAP to be registered to the Global Assembly Cache (GAC) prior to deployment and execution of Microsoft .NET Web Services. This registration is performed automatically when the IVT is first run. The IVT calls the script setcp.bat, which sets up the correct CLASSPATH for the Java components and that in turn, calls the script registerDotNet.bat located in the bin directory. It uses the Microsoft .NET utility gacutil to perform the registration. Once made, this registration survives system reboots and subsequently re-registration is not necessary. If the IVT is not used, then setcp.bat makes the registration when it is first invoked.

Occasionally, you may need to force re-registration of these components, for example, if there are potential version mis-matches or some other suspected problem. The registerDotNet.bat script is invoked directly to do this. It uses gacutil to explicitly remove the existing references and then re-register them. For example:

Example 3-4 Registering required libraries to Microsoft .NET

```
C:\RandomService>\ma0r\bin\registerDotNet
```

Registering dotNet classes. This should only need to be done once.

```
C:\RandomService>
```

The registerDotNet.bat script also registers the WebSphere MQ classes for Microsoft .NET library (amqmdnet.dll) if this has not already been registered. Unlike the WebSphere MQ transport for SOAP components however, it will not refresh any existing registration. In the event that these classes are being updated to a later level it may be therefore necessary to use gacutil manually to remove the existing registration before calling registerDotNet.bat. These classes are supported and integrated into the WebSphere MQ distribution with effect from CSD05.

3.2.7 Checking the WebSphere MQ transport for SOAP release level

The release level of an installed copy of WebSphere MQ transport for SOAP SupportPac can be identified by typing the command “ma0rver” in a command prompt. For example:

Example 3-5 Checking the release of a WebSphere MQ transport for SOAP SupportPac

```
c:\ma0r>ma0rver
```

This is the February 2003 Cat2 (unsupported) SupportPac release of
WebSphere MQ Transport for SOAP
Press any key to continue . . .

3.3 SOAP formatting

In this section the Simple Object Access Protocol (SOAP) and the various SOAP encoding options that can be used in WebSphere MQ transport for SOAP applications are reviewed. The mechanisms to declare these different options in service code source are also illustrated.

SOAP has become an industry wide specification for describing XML messages and their attachments for delivery across a network. These XML messages are used in the context of Web Services to invoke remote methods. There are options other than SOAP for remote method invocation, but in the Web Services context, SOAP is far more reliable than alternatives such as HTTP's post and get methods. SOAP messages are written in XML as this permits data to be efficiently structured, unlike alternatives such as plain HTML.

In itself, SOAP places no constraints on the structure of the XML message. There are however several different SOAP style and encoding variants. These include:

- ▶ Remote Procedure Call (RPC) encoding
- ▶ Remote Procedure Call (RPC) Literal encoding
- ▶ Document Style encoding (also known as messaging style)
- ▶ Direct Internet Message Exchange (DIME)
- ▶ SOAP with attachments

The first three of these encoding options are the ones most commonly used today. From a developer's point of view, RPC and Document style encoding differ in a number of respects. RPC services are usually accessed by local proxy objects at the client side with invocation data being passed from client to service via parameters that generally follow the technical implementation of the service. The RPC message essentially describes the details of a procedure call with its name and parameters values or procedure returns. Document style services publish their data to services in a more generic XML form. Services can parse such messages from any client that follows a set XML schema. This is therefore a less rigid implementation than RPC.

RPC Literal encoding is a format for use where a single XML object is pre-prepared at the application level and passed complete to the SOAP stack. This has the advantage that the SOAP stack can serialize the data faster, but the disadvantage is that additional XML parsing will usually necessary in the application. DIME and SOAP with attachments are two different methods for encoding binary data.

RPC encoding is generally the easiest to implement within a SOAP engine and Document style encoding generally the hardest. RPC Literal is generally faster than RPC but slower than Document style. The most appropriate encoding option depends on the details of the specific application.

In addition to these different styles, there may also be differences in SOAP formatting from different vendors of Web Service infrastructures according to their different interpretations of the specification. Standards for the SOAP specification are still in their infancy. There are also currently several different versions of the SOAP specification in use, such as versions 1.1 and 1.2.

One of the design aims in WebSphere MQ transport for SOAP was to decouple its implementation from the specifics of the SOAP version or format options used in transported messages. The transport level is responsible for posting and receiving messages from the a services host and should not need to have knowledge of the details of the actual SOAP formatting.

Although the transport is theoretically independent of the SOAP format, care must be taken to ensure that the client and server Web Services infrastructure are compatible so that they are able to understand and respond to each others SOAP messages.

WebSphere MQ transport for SOAP can process RPC and Document style messages. However, there are some limitations:

- ▶ An Axis client cannot call a .Microsoft .NET service using RPC style encoding.
- ▶ RPC Literal encoding is not currently implemented.
- ▶ SOAP complex type support is not currently implemented.
- ▶ Neither SOAP with attachments nor DIME encoding are currently implemented.

Note: There is one area where WebSphere MQ transport for SOAP makes syntactic assumptions about the format of SOAP messages. This is in the Microsoft .NET listener MQSoapHost, where a request SOAP message is manually parsed to extract the details of a service that is to be invoked. The Java equivalent of this listener, SimpleJMSListener does not make these assumptions as it does not have to manually parse messages to extract the service details.

3.3.1 How to specify RPC or Document style encoding

Microsoft .NET, by default, creates services using Document style encoding. Services can however use RPC encoding by including the “SoapRpcMethod” attribute in the method declaration. Alternatively, the attribute [SoapRpcService] can be used on the class definition. The following extract, from the WebSphere MQ transport for SOAP samples, illustrates this in practice with the method getQuote being declared to use RPC style encoding and the method getQuoteDOC being declared to use the default Document style method encoding.

Example 3-6 Setting RPC or Document style encoding in service code

```
//RPC method
[WebMethod] [SoapRpcMethod]
    public float getQuote(String symbol) {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
        return 88.88F;
    }
```

```
//Document style method
[WebMethod]
public float getQuoteDOC(String symbol) {
    return 77.77F;
}
```

WebSphere MQ transport for SOAP does not permit target Microsoft .NET services to specify the SOAPAction HTTP header field. This is because the MQSoapHost listener itself reconstructs the SOAPAction field based on the default rules for the namespace and method.

For more details on SOAP formatting, refer to the article “Introduction for Web Services and the WSDK” which is available at the following URL:

<http://www.ibm.com/developerWorks>

3.4 WebSphere MQ transport for SOAP application development

A Web Services client developed to send SOAP messages over HTTP transport can normally be reused with WebSphere MQ transport for SOAP with only one mandatory change. This is the addition of a method call that registers WebSphere MQ as a recognized transport.

A Microsoft .NET service will need some further simple modifications that are described in this chapter. These are to declare the particular Web Service methods and SOAP parameter formatting options.

Java clients also need a method call to enable the WebSphere MQ transport, but after that is added services can generally be re-used with no need for further modification.

Because of this minimal coding impact, developers are able to continue to use favored development tools such as WebSphere Application Developer or Microsoft Visual Studio .NET for both their client and service applications. Use of the different transport mechanism does not directly affect the basic development environment. It is however necessary to deploy services outside the development environment before those services can be successfully invoked from clients.

Developers who are using WebSphere MQ transport for SOAP might, as a general guideline, adopt the following development sequence:

1. First, implement and test clients and services operating directly together within a single process/application development environment.

2. Next, make a test deployment of the service as an HTTP WebService. As well as deployment of the service, minor changes to the client will also be required. Then retest and debug the client with the service.
3. Finally, make a test deployment of the service as a WebSphere MQ transport for SOAP service. As described above, this process should be straightforward with little need for additional debugging.

This complete procedure is neither mandatory nor always appropriate. The first two steps would not be necessary, for example, if a target service has already been successfully implemented with HTTP.

3.4.1 Client environment

When considering the client environment, there are various different scenarios in which client applications can be prepared and made available. For example:

- ▶ By installing a pre-prepared Java or .NET binary application on the client system
- ▶ By developing and compiling an application directly on the client system that uses pre-prepared proxies
- ▶ By developing and compiling an application directly on a client system without pre-prepared proxies
- ▶ By using different systems for both the client and server

The first case is the simplest in terms of setting up the required client environment. For Microsoft .NET clients, it is necessary to have the Framework and (at the time of writing) the Framework SDK installed, together with WebSphere MQ transport for SOAP. It is also necessary to register the required DLLs, for example with the registerDotNet.bat script located in the bin directory. Java clients require WebSphere MQ transport for SOAP and access to a Java JDK. A default WebSphere MQ URI is set within the automatically generated proxy routines and care needs to be taken to ensure that this is either set correctly at deployment time for the eventual environment, or that the URL is over-ridden when executing the client.

In the second case, using Microsoft .NET services or using Java services that are being accessed by proxies, it is necessary to copy the proxy code from the server system on which the service is deployed back to the client. This can be extracted from the helpers directory created during deployment. The client is then built with the relevant proxy stubs that were generated on and copied from the deployment system.

The third case assumes the proxies are not available and it is therefore necessary to take the Java or Microsoft .NET source, together with the ASMX

file, and redeploy the service on the client system to regenerate the proxies. Although there is no reason why the deployment utility cannot be re-run on the client system, the result is an unnecessary overhead as the request queue is generated on the client and all the other intermediate deployment steps are executed. This does not cause any malfunction but it is superfluous, causes confusion and can give rise to security concerns. Some organizations may deem it unacceptable to publish the complete service code to clients.

In the fourth case, initial versions of a client and server are prepared on one system and then both transported to different target systems. Here, the complete contents of the deployment directory must be copied to the server system. The service must then be redeployed and the client changed as necessary, then rebuilt and distributed as per one of the methods above.

3.5 WebSphere MQ transport for SOAP .NET deployment

Deployment is essentially the process of configuring the host Web Services infrastructure to recognize the prepared Web Service. After deployment, clients are able to invoke the service with a special proxy class generated by the deployment process. Java clients can invoke the Web Service directly with low level calls or by the use of a Web Services Description Language (WSDL) configuration file. On the server side the infrastructure recognizes the calls to the service and are able to call it as specified.

The WebSphere MQ transport for SOAP deployment procedure is based around a provided Java deployment utility called `deployWmqService`. This is a simple character based utility and there are currently no graphical implementations.

Although it is not necessary to use the deployment utility, it is normally simpler to use this than undertake the various deployment steps manually. The utility is implemented by IBM in the Java language in order to minimize platform dependency issues. WebSphere MQ transport for SOAP is not only available for .NET Web Services hosted on Windows platforms, but also for Apache Axis Web Services hosted on other operating systems such as Linux and AIX.

The deployment utility operates “bottom up” in that it starts with an implemented class for the service:

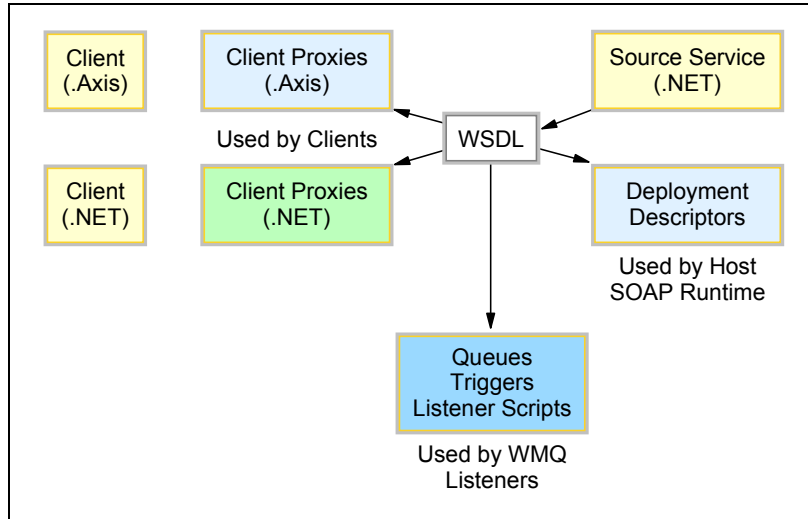


Figure 3-2 WebSphere MQ transport for SOAP deployment

The main activities deployment undertakes are:

1. To prepare the WSDL
2. To deploy the service from the WSDL
3. To generate client proxies from the WSDL. (The use of proxies simplifies the process of invoking the Web Service process).
4. To prepare a script for invoking a .NET listener on the Web Service platform
5. To configure MQ with the required queues and processes necessary to implement the service.

These separate steps can be run individually through the utility for alternative deployment scenarios.

The utility is called from the command script `deployWMQService.bat`. This in turn invokes the Java program `com.ibm.mq.ma0r.util.deployWMQService`. To use this utility the `CLASSPATH` environment variable must be correctly setup first. The easiest way to do this is to execute the “`setcp.bat`” utility provided in the “`bin`” sub-directory.

Details of how WebSphere MQ transport for SOAP services deployed in organizations are highly dependent on the target environment. The deployment utility provided acts as a useful guide. Developers with a more complex deployment scenario need to build their own deployment processes in order to match their requirements more closely. The source code for the Java

deployWmqService utility is therefore included with WebSphere MQ transport for SOAP (in the docs sub-directory) to help customers with this process.

The WebSphere MQ configuration activities perform only basic customization, specifically creating the request queue and setting up process definitions to enable the .NET or Java listeners to be started with trigger monitors. Further configuration is required in real situations, for example:

- ▶ To create channel definitions and enable communication between queue managers located on different machines. This is needed when the client is operating with server bindings to a service on a different machine.
- ▶ To create a server connection channel where the client application is to invoke the service with WebSphere MQ client bindings and there is no local queue manager at the client. It is necessary to create and configure a channel to enable the client and server to communicate.
- ▶ To configure SSL communications where required. Configuration requirements depend on whether the client is operating with server or client bindings. In general it is necessary to modify the WebSphere MQ channel definitions to use SSL and also to prepare a key repository file and import certificates into the queue manager(s). Refer to chapter “WebSphere MQ transport for SOAP and SSL” on page 64 for more details on SSL.

The deployment utility deployWmqService is called from the directory in which the source is located. This is also the directory from which the service is executed. The deployment procedure creates various directories and files within and under this directory. Most of the output is placed in the helpers subdirectory which is created automatically.

Deployment performs the following actions:

- ▶ For Java services, it compiles the source into the classes subdirectory. For example:

```
helpers\classes\javaDemos\server\StockQuoteAxis.class
```

- ▶ It generates the appropriate WSDL in the file helpers\<classname>_Wmq.wsdl. For example:

```
helpers\javaDemos.server.StockQuoteAxis_Wmq.wsdl
```

- ▶ For Java services, it prepares a deployment descriptor file (<classname>deploy.wsdd and <classname>undeploy.wsdd), and deploys into the execution directory to create or update server-config.wsdd. For example:

```
helpers\javaDemos\server\StockQuoteAxis_deploy.wsdd
helpers\javaDemos\server\StockQuoteAxis_undeploy.wsdd
helpers\server-config.wsdd
```

- ▶ It generates the appropriate proxies for Java, C# and Visual Basic from this WSDL. On Windows platforms, proxies are generated in Java, Visual Basic and C# regardless of the language in which the service was written. The package and file directories for Java proxies reflect the original package. The C# and VB proxies are placed directly into the helpers directory. For example:

Java proxies might be located in:

`helpers\javaDemos\server\StockQuoteAxisServiceLocator.java`

A C# proxy might be located in `helpers\StockQuoteAxisService.cs`

A Visual Basic proxy might be located in `StockQuoteAxisService.vb`

- ▶ It compiles the Java proxies into the appropriate directory beneath:

`helpers\classes` directory

For example:

`helpers\classes\javaDemos\server`

- ▶ It prepares the WebSphere MQ queue within which messages requesting invocation of the service are passed. This queue is named `SOAP.<classname>`, for example:

`SOAP.javaDemos.server.StockQuoteAxis`

- ▶ It prepares a file to start the listener that monitors this request queue and begin the process of service invocation. This script file is placed in `helpers\listener_<classname>.bat`, for example:

`helpers\listen_javaDemos.server.StockQuoteAxis.bat`

- ▶ It prepares WebSphere MQ definitions that permit a listener process to be automatically triggered. For example:

WebSphere MQ Process: `SOAP.javaDemos.server.StockQuoteAxis`

WebSphere MQ Trigger Initiation Queue: `SOAP.INITQ`

Although these definitions are automatically created, the use of triggered listeners are optional. Refer to 3.10, “Starting listeners with WebSphere MQ triggering” on page 60 for more information about triggered listeners.

The WSDL and the proxies generated from it have the appropriate WebSphere MQ URI set within it for the service, for example:

`wmq:SOAP.javaDemos.server.StockQuoteAxis@MQSOAP.DEMO.QM?connectQueueManager=MQSOAP.DEMO.QM`

3.5.1 WebSphere MQ URI Syntax

WebSphere MQ Transport for SOAP client applications make use of a specially formed URI that defines various WebSphere MQ attributes specific to the transport. This is formatted as follows:

wmq:<queueName>[@<queueManagerName>][?<options>], where options are an optional comma separated list of attribute=value pairs.

The proxy code generated by the deployment process sets a default URI based on the class name of the service and the target queue manager, but it needs to be changed to match precise requirements.

WebSphere MQ names within the URI are case sensitive, but the keywords themselves are case insensitive. The queueName and queueManagerName are used at the client to open the queue for sending the request message. If the queueManagerName is not specified, the queueName must be defined at the queue manager to which the client is connected.

The full list of the URI options is as follows.

- ▶ replyToQueue=<replyToQueue>
Specifies the queue at the client side to be used for the response message.
- ▶ timeout=<timeout>
Specifies the timeout in milliseconds for the client to wait for a response message. The default value is set for 10 seconds (10000).
- ▶ connectQueueManager=<connectQueueManager>
Specifies the queue manager to which the client connects.
- ▶ expiry=<expiry>
Specifies how long WebSphere MQ retains the message before discarding it. This is expressed in units of tenths of a second. By default the expiry time is unlimited (-1).
- ▶ priority=<priority>
Specifies the message priority as an integer value greater than or equal to minus one. The default value is -1 which indicates priority as per queue definitions.
- ▶ persistence=<persistence>
Specifies the message persistency:
0 = Not persistent
1 = Persistence
2 = Persistency as per queue definition

The default value is 2, persistency as defined by the queue definition.

3.5.2 WebSphere MQ client connection options

Here are WebSphere MQ client connection options:

- ▶ `clientChannel=<clientChannel>`
Specifies the channel to be used for a SOAP client to make a WebSphere MQ client connection.
- ▶ `clientTransport=<clientTransport>`
Specifies the transport to be used for a SOAP client to make a WebSphere MQ client connection (default tcp).
- ▶ `clientConnection=<clientConnection>`
Specifies the connection to be used for a SOAP client to make a WebSphere MQ client connection. This is the host name of the remote system.

SSL options

The following Secure Sockets Layer (SSL) options have been tested with Microsoft .NET clients only and will not function with Java clients.

- ▶ `sslKeyRepository=<sslKeyRepository>`
Specifies the location of the SSL repository
- ▶ `sslCipherSpec=<sslCipherSpec>`
Specifies the SSL Cipher specification to use
- ▶ `sslPeerName=<sspPeerName>`
Specifies any required SSL distinguished name authentication. This should be passed in parentheses as discussed in 3.11.2, “Use of SSLPeerName” on page 66.

Refer to 3.11, “WebSphere MQ transport for SOAP and SSL” on page 64 for more details on how these SSL options are used.

This URI needs to be changed to match the target WebSphere MQ topology. (When it is necessary to change the URI, we recommend that you make the change within the client code and not in the automatically generated proxy code).

Care must be taken with naming during the deployment process, particularly where “namespace” directives are used in the actual service source. The deployment utility creates the request queue using the name of the input ASMX file. For example, if the ASMX file is called test.asmx, the request queue is generated as SOAP.test. When using Microsoft Visual Studio .NET care should be taken to check that the ASMX filename should match that for the compiled

DLL for the service. Take care to check also that the name of the request queue that is generated by the deployment utility matches that set within the client URI.

Once a service has been deployed from a given directory client it can only be accessed from that directory. If the service directory has to be moved, or renamed, the service must be redeployed.

Note: .NET services that are not written using code embedded in the .ASMX file should be compiled prior to executing deployWMQService

3.5.3 Calling deployWMQService

In this section the calling options for the deployment utility deployWMQService are reviewed. The utility is invoked by executing a BAT or shell script called deployWMQService.bat/sh. This script is located in the bin sub-directory which will be placed into the path after setcp.bat is executed. The wrapper script invokes the Java program com.ibm.mq.ma0r.util.deployWMQService. The script passes in required environment variables as Java system properties which is why it is best to not invoke the Java program directly.

The script should be invoked as follows:

deployWMQService [options] -f className

- ▶ -f <input-file>

This argument is mandatory. For Microsoft .NET services <input-file> is the name of the assembly directive (ASMX) file that describes the service. For Java services, <input-file> is the fully qualified package name the source of the service. This path is denoted either with fileName or className notations.

Other optional arguments and parameters are:

- ▶ -m <queueManagerName>

Specifies the name of the queue manager on which the service is to be hosted. The default is the default queue manager ("").

- ▶ -q <queueName>

Specifies the name of the request queue to be used to carry messages to this service. For Java services this defaults to a name derived from the path name where the source is located. For .Net services it defaults to a name derived from the name of the ASMX file.

- ▶ -echo ON | OFF

Turns echo tracing on or off for execution of deployWMQService. Note this is currently a very simple tracing mechanism that is totally separate from WebSphere MQ tracing. The tracing is expected to be integrated into WebSphere MQ tracing in the future.

► -T num

Number of threads to use in listener. The default is one thread. (See below for more information about this parameter.)

► -v

Sets verbose output from commands

► -c <operation>

Specifies a specific part of the deployment process to be executed. Operation is one of these:

– allAxis

Perform all compile and setup steps for an Axis/Java service.

– compileJava

Compile the Java service (.java to .class).

– genAxisWsdI

Generate WsdI (.class to .wsdl).

– axisDeploy

Deploy the class file (.wsdl to .wsdd, apply .wsdd).

– genProxiestoAxis

Generate proxies (.wsdl to .java, .class, .cs and .vb).

– genAxisWMQBits

Setup WMQ queues, listeners and triggers for an Axis service.

– allAsmx

Perform all setup steps for an Asmx service.

– genAsmxWsdI

Generate WsdI (.asmx to .wsdl).

– genProxiesToDotNet

Generate proxies (.wsdl to .java, .class, .cs and .vb).

– genAsmxWMQBits

Setup WMQ queues, listeners and triggers.

- startWMQMonitor

Start the trigger monitor for WMQSOAP services.

It is not necessary to invoke the utility for each deployment step listed above. Where a filename is given with a .java or .asmx extension, the operation defaults to allAxis or allAsmx respectively. The name of this file is specified with the utility using the -f flag. The default operation then actions all the necessary deployment steps with the exception of the invocation of a trigger monitor.

The -T flag is used when invoking deployWMQService to implement a multi-threaded listener. The flag is followed with the number of target threads. If -T is not used, it defaults to a single threaded listener.

WebSphere MQ transport for SOAP includes a set of demonstrations which help to illustrate the complete deployment process further. In particular, the script demos\regenDemo.bat rebuilds all of the sample programs provided and includes calls to deployWMQService in order to illustrate the deployment procedure for both .NET and Axis Web Services.

3.6 WebSphere MQ transport for SOAP listener for .NET

This section gives more detail on the listener MQSOAPHost provided with WebSphere MQ transport for SOAP. This listener receives and actions requests for Microsoft .NET Web Services.

For Microsoft .NET Web Services, WebSphere MQ transport for SOAP requires a separate queue and listener for each deployed service. This is a key difference to the SimpleJMSListener where it is possible to use a single listener process to access services that have been deployed from the same directory.

MQSOAPHost listeners can either be started manually or alternatively WebSphere MQ triggering can be used to start them automatically on demand. For customers deploying a large number of services, triggering can prove more convenient. The use of triggering is discussed in more detail in 3.10, “Starting listeners with WebSphere MQ triggering” on page 60.

The service deployment process automatically generates the service request queues and prepares process definitions for the use of triggered listeners. It generates a script in the helpers directory, called listen_<classname>.bat that can be used to start a listener for the particular service. The deployment utility set the request queue name in the listener start up script to SOAP.<className>. This is changed if required, with the -q command line option of the listener. The default setting for the queue manager is MQSOAP.DEMO.QM. The queue

manager name can be changed at deployment with the -m option, or by specifically overriding it in the WebSphere MQ URL.

Note: This same script is used to start the listener whether it is invoked manually or by triggering.

SOAP messages from a client are encapsulated within a WebSphere MQ message and dispatched to the listener via the WebSphere MQ request queue. The MQSoapHost process monitors this queue for incoming messages. The method ProcessRequests() retrieves incoming messages from the request queue, extracts the text of the message and then uses the Microsoft .NET infrastructure to invoke the service. Both the SOAP processing and the actual invocation of the service itself are performed within the listener thread. The SOAP response message from is then returned to the client via the WebSphere MQ response queue.

Messages from clients are read into a byte array and then converted to text assuming UTF-8 encoding. Response messages are returned back transparently as a byte array.

MQSoapHost can operate in multi-threaded mode with a configurable number of worker threads each of which can read and action messages from request queue in parallel. This is required where service execution is comparatively slow so as to not compromise performance. The default number of threads is one, but this is changed with the -T command line option. This option is either given directly to the deployment utility, in which case it is placed into the listener start up script, or to the MQSoapHost command line if invoking the listener directly.

The listener sets a correlation id in the reply message which is the same as the originating message id, as per standard practice. The message is then dispatched over WebSphere MQ with the same persistence, priority and expiry options of the request message. These options are specified in the WebSphere MQ URI used in the client application.

The MQSoapHost listener does not currently handle pure JMS messages that include an MQRHF2 header. This is primarily because the C# classes for WebSphere MQ do not currently support JMS style messages. Client applications to Microsoft .NET services must therefore use native WebSphere MQ rather than pure JMS. Because of this restriction, it is necessary for the Java WMQSender code to prevent the generation of MQRHF2 header and ensure interoperability between Java clients and .NET services.

The listener currently only process request/response message pairs and does not correctly handle one way messages or report messages. As described in

“Current status and future plans” on page 69, facilities for one way messaging or report messages may be included in a future version.

For .NET services it is currently necessary to have a separate listener for each deployed service, even if they are deployed from the same directory.

3.6.1 Executing MQSoapHost

The Microsoft .NET listener is located in bin\MQSoapHost.exe under the installation directory. The options for calling this are:

```
MQSoapHost -m QueueManager -q QueueName -w directory [-v virtual dir] [-s web-service] [-T numServerThreads]
```

The mandatory parameters are:

- ▶ -m
 <Qmgr Name>
- ▶ -q
 <Queue Name>
- ▶ -w
 <directory>

The optional parameters are:

- ▶ -w
 Physical directory containing Web Service
 (default is 'c:\\inetpub\\wwwroot\\<Application>\\')
- ▶ -v
 <virtual directory>
 Application (extracted from queue if not specified)
- ▶ -s
 <Webservice>
 WebService name, for example. test.asmx

MQSoapHost is normally executed from the listener script generated at deployment so it is not necessary to run it directly. However, where the default options are not appropriate, it is necessary to either modify the generated script or use an alternative script. An example of how MQSoapHost might be called directly is provided below:

```
MQSoapHost -m MQSOAP.DEMO.QM -q SOAP.StockQuoteDotNet -w C:\ma0r\demos -v /vdir -s  
StockQuoteDotNet.asmx -T 10
```

3.7 A simple example with a Microsoft .NET Web Service

An example of how to use WebSphere MQ transport for SOAP is shown here. A Microsoft .NET client is used to access a very simple Web Service.

The following steps are required to develop this example:

- ▶ Write the Web Service
- ▶ Write the Web Service directive file
- ▶ Deploy the service using WebSphere MQ transport for SOAP
- ▶ Write the client application using generated Proxy code
- ▶ Define the WebSphere MQ response queue
- ▶ Define WebSphere MQ communication channels
- ▶ Start the prepared Microsoft .NET listener
- ▶ Test the application

In this section the service is developed on the same local machine as the client. The subsequent sections demonstrate how the service is invoked across the network, first with client bindings and then with server bindings.

3.7.1 Write the Web Service

Our Web Service is a simple method called `getRandom` that accepts a string as an input parameter which can be set to either “int” or “double”. The service then returns a string representation of either an integer or double precision random number. The service is first developed as a purely local standalone application, that is, it is not implemented as a Web Service. Once this standalone application is working properly then it is converted it into a Web Service.

When turning methods into Web Services it is necessary to check that any arguments to methods of the Web Services are compatible with the host .NET (or Axis) environment. Refer to 3.4, “WebSphere MQ transport for SOAP application development” on page 23. In this case, simple data types are used to prevent any issues.

When adopting code from a local application, the service method must be modified to declare it as a Web Service and the method by which each method's parameters are to be formatted are also identified. (These steps are not necessary when implementing Java services). When adopting service code that has already been prepared as an HTTP WebService, existing definitions do not further modification for it to be used as a WebSphere MQ WebService. (However, it must still be deployed through the WebSphere MQ transport for SOAP deployment mechanism).

The example below shows the Web Service declaration and method formatting option (RPC) in bold.

Example 3-8 Our getRandom Microsoft .NET Web Service

```
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;

[WebService (Namespace="http://dotnet.server")]
public class RandomNumberNET
{
    public Random r;

    public RandomNumberNET()
    {
        r = new Random();
    }

    [WebMethod] [SoapRpcMethod]
    public String getRandom(String varType)
    {
        if (varType.Equals("int"))
        {
            int anInt = r.Next(1000000);
            return System.Convert.ToString(anInt);
        }

        if (varType.Equals("double"))
        {
            double aDouble = r.NextDouble();
            return System.Convert.ToString(aDouble);
        }

        return("Error");
    }
}
```

The Web Service is placed into a new sub-directory called RandomServiceNET and is saved in a file called RandomNumberNET.cs.

The service code must be compiled. In this simple example the command line tools from the Microsoft .NET framework SDK is used.

Example 3-9 Compiling the Microsoft .NET getRandom service

```
C:\RandomServiceNET>csc /t:library RandomNumberNET.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.
```

The service code is compiled into a DLL file, RandomNumberNET.dll. When the Framework SDK's command line tools are used, it is important that the DLL is manually placed into a sub-directory called "bin" under the directory, where the service code and ASMX file are located. It is important that this is done before running the deployment utility otherwise the deployment process will fail. (This does not apply when using the Microsoft Visual Studio .NET IDE, as described in 9.3.3, "WebSphere MQ transport for SOAP deployment for IAS" on page 185

Note: Pre-compilation is not required if the service source is placed "in-line" within the assembly directive file.

For more details on the process of customizing existing service code, refer to the Microsoft .NET documentation.

3.7.2 Write the .NET ASMX service directive file

Having prepared the actual service code, it is now necessary to create an ASMX file which is the addressable entry point for the service used by Microsoft .NET. This file normally contains a simple one line definition, in this case as follows:

Example 3-10 Our assembly directive file for our getRandom service

```
<%@ WebService Language="C#" Codebehind="RandomNumberNET.cs" Class="RandomNumberNET" %>
```

This directive specifies that the source for the service is located in RandomNumberNET.cs and that its classname is RandomNumberNET. In our testing it is noted that the actual source name given with the "Codebehind" parameter is not significant. Deployment is successful even if this name is incorrect. The nominated classname is significant and needs to be correct.

It is possible to enter the service method directly in the ASMX file as "inline code". Refer to the directive file StockQuoteDotNet.asmx in the WebSphere MQ

transport for SOAP demos directory for an example on how to achieve this. In the business case scenario, having in-line source was not the most practical route, especially as it was required to use existing code for a new Web Service. The ASMX file was therefore separated from the service code.

Instead of creating this ASMX file manually, an alternative is to create a new C# project in Microsoft Visual Studio .NET. The IDE automatically generates the ASMX file with the correct references. This is done in Chapter 9., “Messaging solution: .NET client to .NET Web Services using WebSphere MQ SOAP transport” on page 163 in our business scenario and the reader is referred there for the detail.

3.7.3 Deploying the Microsoft .NET service

The next step in the development process is to run the WebSphere MQ transport for SOAP deployment utility.

It is first necessary to set the CLASSPATH environment variable so this utility can be found by the JVM. This is achieved with the setcp.bat script:

Example 3-11 Setting up the CLASSPATH environment variable

```
C:\RandomServiceNET>\ma0r\bin\setcp
```

```
C:\RandomServiceNET>
```

This script sets all required classes for WebSphere MQ transport for SOAP correctly into the CLASSPATH. This script is located in the bin sub-directory under the installation directory (note that its location may change in future releases). The setcp.bat script is safe to be executed from directories external to the actual installation directory so services can be located in any required directory on the development system.

Care is needed when executing other Java programs independently of WebSphere MQ transport for SOAP within the same command window as setcp.bat completely resets the CLASSPATH.

Before running the deployment utility it is necessary to create and start the target queue manager. This is done in the usual way with runmqsc or WebSphere MQ Explorer. In our example the queue manager is named RANDOMQM.

Once these issues are addressed, the deployment utility can now be executed as follows:

Example 3-12 Deployment of our getRandom service

```
C:\RandomServiceNET>deployMQService -m RANDOMQM -f RandomNumberNET.asmx
Package name: DefaultNamespace
Generating WSDL...
mqsoapwsdl wmq:SOAP.RandomNumberNET@RANDOMQM?connectQueueManager=RANDOMQM Random
NumberNET.asmx helpers\RandomNumberNET_Wmq.wsdl
Preparing listener...
Configuring MQ...
Generating and compiling proxy code...
java com.ibm.mq.ma0r.tools.RunWSDL2Java --output helpers -p dotNetService helper
s\RandomNumberNET_Wmq.wsdl

C:\RandomServiceNET>
```

It is only necessary to provide the deployment utility with two options in this case. These are to specify the queue manager and the name of the assembly directive file respectively.

The deployment utility performs the following actions:

- ▶ It generates WSDL that defines the implementation of the service.
- ▶ It configures the request queue SOAP.RandomNumberNET.
- ▶ It configures a trigger monitor for automated listener start-up (use is optional).
- ▶ It creates a script to start the MQSoapHost listener when required. The listener is configured to read messages from the request queue SOAP.RandomNumberNET, then invoke the getRandom() Web Service through the .NET Web Services framework and finally return the response, encapsulated into a SOAP string, back to the client. This response is returned to the client via the response queue SOAP.RESPONSE.RandomNumberNET.
- ▶ Generates the proxy code that can be used by a client application to invoke the service.

The proxy code is located in the helpers sub-directory. C#, Visual Basic and Java proxy classes are all generated automatically. These have the same filename prefix as the ASMX service file and care must be taken when referring to these files across directories as there are normally two files with the same name. One is the actual service implementation source in the main deployment directory, and the other is the proxy source located in the “helpers” sub-directory. This is illustrated in the following directory listing:

Example 3-13 Take care with service and proxy file name references

```
C:\RandomServiceNET>dir /s *.cs *.vb
Volume in drive C is C: Windows 2000
```

Volume Serial Number is 98E7-6D3C

Directory of C:\RandomServiceNET

```
15/07/2003  15:51                748 RandomNumberNET.cs
15/07/2003  15:52            1,285 RandomNumberNetClient.cs
                2 File(s)            2,033 bytes
```

Directory of C:\RandomServiceNET\helpers

```
15/07/2003  16:55                2,001 RandomNumberNET.cs
```

Directory of C:\RandomServiceNET\helpers

```
15/07/2003  16:55                2,138 RandomNumberNET.vb
                2 File(s)            4,139 bytes
```

```
Total Files Listed:
        4 File(s)            6,172 bytes
        0 Dir(s) 10,027,823,104 bytes free
```

C:\RandomServiceNET>

3.7.4 Write the client application

The client machine requires a simple application to invoke the service. In this case a Microsoft .NET client is written and this application is most easily written either in C# or Visual Basic because proxy classes are automatically generated by the deployment utility for both of these languages.

The sample client programs that are supplied with WebSphere MQ transport for SOAP provide a useful template for writing a client. These are located under the WebSphere MQ transport for SOAP installation directory in the sub-directory demos\dotnetDemos\clients. The file SQCS2DotNet.cs and SQVB2DotNet.vb are the C# and Visual Basic examples respectively. In this case the C# example is amended to invoke the service as follows:

Example 3-14 Our simple client application

using System;

```
class RandomNumberNetClient
{
    [STAThread]
    static void Main(string[] args)
    {
        String randomType="int";
```

```

try
{
    MQSOAP.MQWebRequest.Register();

    RandomNumberNET rsobj = new RandomNumberNET();

    // Any first argument is used as the target Url
    if (args.GetLength(0) >= 1) rsobj.Url = args[0];

    // Any second argument is the type of the random number required
    if (args.GetLength(0) >= 2) randomType = args[1];

    String res = rsobj.getRandom(randomType);

    Console.WriteLine("Random service returned: " + res);

}
catch (System.Exception e)
{
    Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING RandomNumberOnNet DEMO <<<\n" +
e.ToString());
}

}
}

```

The steps taken in this example are as follows:

- ▶ The sample source is taken and saved as RandomNumberNetClient.cs in the service deployment directory.
- ▶ The name of the method is changed to invoke to getRandom as defined in the service and hence in our proxy.
- ▶ The returned object is specified as a string and passed in an argument to the program representing the amount it was required to invest.
- ▶ The optional target URI now becomes the second argument. In this example, the proxy code sets this by default to:

wmq:SOAP.RandomNumberNET@WSServer?connectQueueManager=WSServer

In this case, the default is overridden in order to define the correct response queue SOAP.RESPONSE.RandomNumberNET and the appropriate client connection attributes. By default the response queue is set to a local queue MQSOAP.DEMO.RESPONSE. This is the queue used by the WebSphere MQ transport for SOAP sample applications. The deployment process assumes a local queue manager connection at the client. In this instance a local queue manager is to be used but with a different name appropriate for the service.

The correct response queue and client connection attributes need to be specified by setting the URI as follows:

`wmq:SOAP.RandomNumberNET@RANDOMMQM?replyToQueue=SOAP.RESPONSE.RandomNumberNET,connectQueueManager=RANDOMMQM`

The URI may also be used to set other attributes such as timeout, persistence, expiry or SSL options. (Refer to 3.4, “WebSphere MQ transport for SOAP application development” on page 23).

The call `MQSOAP.MQWebRequest.Register()` registers the URI prefix `wmq:` as a pluggable transport as an alternative to `http:`. This currently has to be specifically called from every client process. The call has been highlighted in bold in the above example.

Our client application is compiled as follows:

Example 3-15 Compilation of the client code

```
C:\RandomServiceNET>csc /lib:c:\ma0r\bin /r:MQSOAP.dll RandomNumberNetClient.cs
helpers\RandomNumberNET.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.
```

3.7.5 Define the WebSphere MQ response queue

The deployment process does not create a service specific queue for response messages. By default WebSphere MQ transport for SOAP assumes the use of `MQSOAP.DEMO.RESPONSE` which is really only intended for the supplied demonstrations. A response queue is therefore created directly by executing the `runmqsc` or WebSphere MQ Explorer utility. In this test a local queue is created and called `SOAP.RESPONSE.RandomNumberNET`.

Tip: Care should be taken not to use the WebSphere MQ Explorer graphical user interface at the same time as the `runmqsc` command line utility as problems can be experienced with the operation of the graphical interface.

3.7.6 Start the prepared Microsoft .NET listener

The listener is manually started as follows:

Example 3-16

```
C:\RandomServiceNET\helpers>listen_RandomNumberNET
```

```
C:\RandomServiceNET\helpers>rem - generated by deployWMQService.java at 15-Jul-0
```

3 16:55:23

```
C:\RandomServiceNET\helpers>call C:\ma0r\bin\setcp
```

```
C:\RandomServiceNET\helpers>MQSoapHost -m RANDOMQM -q SOAP.RandomNumberNET -w C:\RandomServiceNET -v /vdir -s RandomNumberNET.asmx -T 1
```

```
*****
*
* WebSphere MQ ASP.NET Webservices server
*
* Parns used are
*
* Qmgr      : RANDOMQM
* Queue     : SOAP.RandomNumberNET
* Physical dir : C:\RandomServiceNET
* Application : /vdir
* Webservice : RandomNumberNET.asmx
* Threads   : 1
*****
```

3.7.7 Test the service

The newly deployed service complete with WebSphere MQ transport is run. The client application is executed passing the target URI and type of floating number (int or double) as command line options.

Example 3-17 Execution of client

```
C:\RandomServiceNET>RandomNumberNetClient wmq:SOAP.RandomNumberNET@RANDOMQM?rep1
yToQueue=SOAP.RESPONSE.RandomNumberNET,connectQueueManager=RANDOMQM int
Using server bindings.
Random service returned: 734510
```

```
C:\RandomServiceNET>RandomNumberNetClient wmq:SOAP.RandomNumberNET@RANDOMQM?rep1
yToQueue=SOAP.RESPONSE.RandomNumberNET,connectQueueManager=RANDOMQM double
Using server bindings.
Random service returned: 0.646812647882296
```

```
C:\RandomServiceNET>RandomNumberNetClient wmq:SOAP.RandomNumberNET@RANDOMQM?rep1
yToQueue=SOAP.RESPONSE.RandomNumberNET,connectQueueManager=RANDOMQM wrong
Using server bindings.
Random service returned: Error
```

```
C:\RandomServiceNET>
```

This is a very simple example. A typical application does not pass the URI into the application via a command line argument but typically forms it internally.

Note: If you are using an ASPX application as your client you will need to add the user ASPNET to the WebSphere MQ group. Otherwise problems will be encountered with access to WebSphere MQ services. Refer to 7.3.2, “Bank service application (C#)” on page 121 for further details.

3.7.8 Distributed test in WebSphere MQ client mode

Having run the simple demonstration locally, it is next tested across the network. This section demonstrates how to do this in WebSphere MQ client mode, where there is no local queue manager on the client. A WebSphere MQ client connection is used from one machine to invoke the service on another.

Rather than use our initial test queue manager of RANDOMQM, a queue manager is created on the server machine called WSServer. A runmqsc script file called WSServer.mqsc is prepared to create required queues and channels. This script is documented in the Appendix A, “Scripts, source code and test data for YuBank” on page 317 and can be downloaded as per the instructions in Appendix B, “Additional material” on page 323.

The next step is to run the WebSphere MQ listener runmqslr on the service system. In this case, it was run on port 1415 so as not to interfere with the operation of the business case application used in the other chapters. The business application used the default port of 1414. For this client connection test, it meant running runmqslr with the port of 1415 explicitly specified with the -p command line option. The port number was then set in the URI as shown below.

The machine originally used for developing the service is targeted as the client machine. The contents of the deployment directory are therefore copied to the server system and then redeployed on that system. (It is only necessary to copy the service source and the assembly directive file from the deployment directory).

To recreate the test with a different queue manager on the server it is necessary to re-run the deployment procedure as follows:

Example 3-18 Redeployment for distributed tests

```
C:\RandomServiceNET>deployWMQService -m WSServer -f RandomNumberNET.asmx
Package name: DefaultNamespace
Generating WSDL...
mqsoapwsdl1 wmq:SOAP.RandomNumberNET@WSServer?connectQueueManager=WSServer Random
NumberNET.asmx helpers\RandomNumberNET_1wmq.wsdl
Preparing listener...
Configuring MQ...
Generating and compiling proxy code...
java com.ibm.mq.ma0r.tools.RunWSDL2Java --output helpers -p dotNetService helper
```

s\RandomNumberNET_wmq.wsd1

C:\RandomServiceNET>

The proxy code in the helpers directory is already on the client system. In our specific example, the existing client application can be used without the need for recompilation. The target URI needs to change, but the client is deliberately programmed to overwrite this on the command line.

Define WebSphere MQ channel definitions

The channel is created for communication between the WebSphere MQ client and the service host. This can either be performed within the runmqsc utility or by use of the WebSphere MQ Explorer Graphical User Interface (GUI). In this case runmqsc is used to define the channel TO.WSServer as follows:

Example 3-19 Defining the WebSphere MQ server connection channel

```
DEFINE CHANNEL(TO.WSServer) CHLTYPE(SVRCONN) TRPTYPE(TCP) +  
MCAUSER('stevens') REPLACE
```

This command is provided in the script WSClient.mqsc.

The listener is started on the server machine using the script file just generated by the deployment procedure:

Example 3-20 Starting the MQSOAPHost listener on the server machine

C:\RandomServiceNET\helpers>listen_RandomNumberNET

C:\RandomServiceNET\helpers>rem - generated by deployWMQService.java at 15-Jul-03 18:25:53

C:\RandomServiceNET\helpers>call C:\ma0r\bin\setcp

C:\RandomServiceNET\helpers>MQSoapHost -m WSServer -q SOAP.RandomNumberNET -w C:\RandomServiceNET -v /vdir -s RandomNumberNET.asmx -T 1

*

* WebSphere MQ ASP.NET Webservices server

*

* Parms used are

*

* Qmgr : WSServer

* Queue : SOAP.RandomNumberNET

* Physical dir : C:\RandomServiceNET

* Application : /vdir

```
* Webservice      : RandomNumberNET.asmx
* Threads         : 1
*
*****
```

As explained above, in our example case there is no need to recompile the client code. However the proxy code still contained the URI of the original deployment and is no longer appropriate for the new configuration as the various client connection attributes need to be set. The target URI is set to runtime with the client's command line option as follows:

Example 3-21 Execution of distributed test with client bindings

```
C:\RandomServiceNET>RandomNumberNetClient wmq:SOAP.RandomNumberNET?clientConnect
ion=ITSOI.hursley.ibm.com(1415),clientChannel=T0.WSSERVER,replyToQueue=SOAP.RESPONSE.R
andomNumberNET int
Using client bindings.
Random service returned: 5810

C:\RandomServiceNET>
```

Note the non default runmqslr port number 1415 is now carried in the URI according to our requirements described above.

3.7.9 Distributed test in WebSphere MQ server bindings mode

To run in bindings mode, the sender and receiver channels are set up on both, the client and service systems. The mqsc script WSServer.mqsc already created the required queues and channels on the server system, but it is necessary to create the additional queues and channels required on the client system. The script WSCient.mqsc is used to do this. This script is documented in the Appendix A, "Scripts, source code and test data for YuBank" on page 317 and can be downloaded as per the instructions in Appendix B, "Additional material" on page 323.

The runmqslr listener is executed on the client system. As in the case of client bindings, the runmqslr is executed with the -p 1415 option to use port 1415.

The service system listener continued to run as it was already configured.

It is necessary to set the CONNAME attribute for the sender channels to include the new port number. These were specified as myhost(1415) and ITS0I.hursley.ibm.com(1415), respectively, and re-execute the client with the appropriate URI for the bindings connection:

```
C:\RandomServiceNET>RandomNumberNetClient wmq:SOAP.RandomNumberNET@WSServer?replyToQueue=SOAP.RESPONSE.RandomNumberNET,connectQueueManager=WSClient int
Using server bindings.
Random service returned: 326397
```

```
C:\RandomServiceNET>
```

3.7.10 Distributed WebSphere MQ using MQ clustering

The simplest way to set up server to server communications in WebSphere MQ is to use clustering.

- ▶ Define all relevant queue managers (in this case SOAP client and SOAP servers) to be in the same cluster.
- ▶ Use an existing cluster, or define a new one, SOAP.CLUSTER for example.

For deployment of a service on a given queue manager, deploy in the standard way as local deployment, then alter the service queue definition to define the queue as a cluster queue. The queue is then visible on all queue managers in the cluster. All the necessary channel definitions will be created automatically by the WebSphere MQ clustering infrastructure.

3.8 WebSphere MQ transport for SOAP with J2EE deployment

This section shows how WebSphere MQ transport for SOAP is used to implement WebSphere MQ transport with J2EE Web Services. The deployment of J2EE Web Services is discussed and then the JMS listener that is provided for J2EE Web Services. A simple J2EE Web Service is developed and deployed and then invoked from a Microsoft .NET client application.

3.8.1 Deployment of J2EE Web Services

The process of invoking `deployWMQService` for J2EE Web Services is identical to that for Microsoft .NET Web Services except that the utility takes the source file for the J2EE service as an argument rather than the .NET assembly directive file (.asmx file). This file is also specified with the `-f` command line option. The utility detects whether it is invoking .NET or J2EE services by inspecting the filename attribute and checking for the “.java” or “.asmx” file. It prepares a script for invoking a JMS listener (see 3.8.2, “WebSphere MQ Transport for SOAP

SimpleJMSListener” on page 49) analogous to the scripts generated for the Microsoft .NET case.

WebSphere MQ transport for SOAP currently requires that J2EE Web Services are implemented through the Apache Axis framework. As with Microsoft .NET, Axis uses SOAP for formatting messages between client and Web Services. Target Web Services must currently be coded directly in Java. The use of Java Web Service (JWS) files are not implemented in the current edition of WebSphere MQ transport for SOAP. The advantage of JWS files are that they do not need compilation at deployment time. Instead, the services are compiled only when the service is accessed. This means that deployment is faster which is useful when initially developing and testing services. There are disadvantages of using JWS files for production environments. The source of the service must be available at the server side which is a security issue for some users. Also, options for configuring access to the service are limited. It is anticipated that the options to use JWS files are integrated into the WebSphere MQ transport for SOAP deployment process in future releases.

In the J2EE case, target services must be specified to deployWMQService from a Java source file. The file is named either in filename only form or in directory path name convention. A simple filename specification for example is, “myService.java”. A relative path name, for example, is: “com\ibm\test\myService.java”. Where a relative path name is used, the utility transforms the service from directory name into Java classname notation. In this case the resultant service would be identified as com.ibm.test.myService.

When using Axis clients with J2EE Web Services, it is not mandatory to use the automatically generated proxy methods generated at deployment. Instead, lower level Axis calls can be used to prepare and then invoke the Web Service directly. Examples of such lower level Axis clients are included in the Independent Verification Test (IVT) utility located in the demos sub-directory under the SupportPac installation directory. The demonstration programs SoapClient.java and WsdlClient.java are used to demonstrate ‘raw’ Axis calls and WSDL based calls respectively. In general it will be easier and simpler to use the proxy method, but the direct option may be useful if it is required to operate at a lower level than the proxy interface permits.

In the example given in this chapter, from a .NET client to a J2EE Web Service, it is necessary to build a client with the automatically generated proxy code in order to access the service specifically from a Microsoft .NET client application.

3.8.2 WebSphere MQ Transport for SOAP SimpleJMSListener

For J2EE Web Services, WebSphere MQ transport for SOAP provides a JMS listener called simpleJMSListener which is used instead of the Microsoft .NET

listener MQSOAPHost. It is the job of the listener to receive messages from the request queue, invoke the target service and then return the result message back to the client via the response queue. It performs the same function for J2EE services that the MQSoapHost listener provides for Microsoft .NET services.

The SimpleJMSListener is implemented in the class `com.ibm.axis.transport.wmq.SimpleJMSListener`. The deployment process automatically creates wrapper scripts in the helpers directory that set up and invoke the listener. As with the Microsoft .NET listener, it is not essential to use these wrapper scripts, they are just there to make the process of starting the listener a little easier. The script is named after the input service class in the form `listen_<classname>.bat`. If, for example, a service called `javaDemos.server.StockQuoteAxis` is deployed, the script is called `listen_javaDemos.server.StockQuoteAxis.bat`.

SimpleJMSListener processes can either be started manually, or, like the MQSOAPHost Microsoft .NET listener, can be started by triggering. Refer to 3.10, “Starting listeners with WebSphere MQ triggering” on page 60 for more details.

SOAP messages from a client are encapsulated within a WebSphere MQ message and dispatched to the listener via the WebSphere MQ request queue. The SimpleJMSListener process monitors this queue for incoming messages. The listener is derived from the Java class `javax.jms.MessageListener` and the method `onMessage()` from this base class is overridden and is automatically called by the base class when a message arrives. This method picks up the incoming JMS message, extracts the text of the message and then feeds it to the Axis engine for processing. Both the SOAP processing and the actual invocation of the service itself are performed within the listener thread. The SOAP response message from Axis is then returned to the client via the WebSphere MQ response queue.

Messages from clients are constructed in a general WebSphere MQ format that is equivalent to a JMS `TextMessage`. The SimpleJMSListener will however accept either incoming `TextMessage` or `BytesMessage` format messages and return response messages in the same format. `BytesMessages` are assumed to be encoded in UTF-8 format and will be returned in UTF-8 format, the ‘encoding’ of the SOAP header itself is ignored. If messages are received in any formats other than `BytesMessage` or `TextMessage`, the `onMessage()` method will throw an exception.

SimpleJMSListener can operate in multi-threaded mode with a configurable number of worker threads each of which can read and action messages from request queue in parallel. This is required where service execution may be comparatively slow so as to not compromise performance. The default number of threads is one, but this may be changed with the `-T` command line option. This

option can be given either directly to the deployment utility, in which case it is placed into the listener start up script, or to the SimpleJMSListener command line if invoking the listener directly.

The deployment utility defaults the request queue name in the listener start up script to SOAP.<className>. This can be changed if required with the -q command line option of SimpleJMSListener. The default settings for the queue manager is MQSOAP.DEMO.QM. The queue manager name can be changed with the -m option of the deployment utility. The deployment process also automatically sets this value up in the BAT file. This behavior is consistent with the Microsoft .NET listener MQSoapHost.

The listener currently only processes request/response message pairs and does not correctly handle one way messages or report messages. If the reply destination in the message is null the onMessage method exits with an error message.

The listener sets a correlation id in the reply message which is the same as the originating message id, as per standard practice. The message is then dispatched over WebSphere MQ with the same persistence and priority options. The expiry interval from the original request message is converted to a time to live interval on the response. These options can all be specified in the WebSphere MQ URI used in the client application.

The JMS listener uses the getJMSDestination() method to determine whether the request message was sent via JMS or raw MQ. This is so that it can process both messages from the Java JMS sender WMQSender and from the Microsoft .NET MQWebRequest sender. The response message is then set to use the same form. The main difference between the two forms is that messages returned over JMS include an RFH2 header, while the native form does not. The client applications will clearly be expecting the right form for their method of invocation. That fact that the listener has to code this logic explicitly is a limitation in the current version of the Java classes for WebSphere MQ. A future update of the Java classes for WebSphere MQ may automatically detect native WebSphere MQ or JMS type messages and correctly pass the response message back in the same form without such direct intervention.

One key difference between the Microsoft .NET listener and the SimpleJMSListener is that with the latter it is possible to use a single listener process to access services that have been deployed from the same directory.

3.8.3 Executing SimpleJMSListener

The general listener calling syntax is:

```
java com.ibm.axis.transport.wmq.SimpleJMSListener [-u wmqUri] [-c  
configDirectory] [-f configFileName] [-a] [-T numServerThreads]
```

Where:

- ▶ -u specifies that target URI
- ▶ -c specifies the Axis configuration directory
The Axis configuration directory defaults to the current working directory
- ▶ -f specifies the Axis configuration file name
The Axis configuration file name defaults to server-config.wsdd.
- ▶ -s specifies the target service
- ▶ -a is used to initiate a thread for an automated stop
- ▶ -T specifies the number of listener threads required (defaults to 1)

For example:

Example 3-23 Illustrating the use of SimpleJMSListener

```
java com.ibm.axis.transport.wmq.SimpleJMSListener -u  
"wmq:SOAP.javaDemos.server.StockQuoteAxis@MQSOAP.DEMO.QM?connectQueueManager=MQSOAP.DEMO.QM" -T  
10
```

In this case two options are used. The -u option specifies the target URI of the service and -T options specifies to activate 10 listener threads.

3.9 A simple example with a J2EE Web Service

This section provides a very simple illustration of how WebSphere MQ transport for SOAP is used with a J2EE Web Service. To do this a very basic J2EE service is written and it is then shown how this can be invoked using WebSphere MQ transport from a client Microsoft .NET application.

The service is built locally, that is, with both client application and server environment on the same machine. All the files for this service are created in a new directory, in this case c:\RandomService.

The following steps were required in this process:

- ▶ Write the Web Service.

- ▶ Deploy the service using WebSphere MQ transport for SOAP.
- ▶ Write the client application using generated Proxy code.
- ▶ Define the WebSphere MQ response queue.
- ▶ Define WebSphere MQ communication channels.
- ▶ Start the prepared JMS listener.
- ▶ Test the application.

Having developed and tested the service locally in this section, in the subsequent sections the service is redeployed and used across the network, first with server bindings and then with client bindings.

3.9.1 Write the Web Service

The service previously demonstrated in 3.7, “A simple example with a Microsoft .NET Web Service” on page 36 was converted to the Java programming language. The basic logic was otherwise unaltered. The service accepts a String input parameter that can be set to either “int” or “double” and returns a random integer or double precision number represented as a string.

Example 3-24 A sample J2EE Web Service

```
import java.io.*;
import java.util.*;
import java.util.Random;

public class RandomNumberAXIS
{
    public Random r;

    public RandomNumberAXIS()
    {
        r = new Random();
    }

    public String getRandom(String varType)
    {
        if (varType.equals("int"))
        {
            Integer anInt = new Integer(r.nextInt(1000000));
            return anInt.toString();
        }

        if (varType.equals("double"))
        {
            Double aDouble = new Double(r.nextDouble());
```

```

        return aDouble.toString();
    }

    return("Error");
}
}

```

Note: Unlike the Microsoft .NET case, no directives are needed inside the Java code in order to configure methods as Web Service methods or to set SOAP formatting options for parameters.

3.9.2 Deploy the service

To use the WebSphere MQ transport for SOAP deployment utility the required classpath must be setup. This is the same procedure as covered earlier in 3.7.3, “Deploying the Microsoft .NET service” on page 39.

As with the Microsoft .NET example, it is necessary to ensure that the queue manager RANDOMQM is available and started before deployment.

The service can now be deployed:

Example 3-25 Deployment of the RandomService Web Service

```

C:\RandomService>deployWMQService -m RANDOMQM -f RandomNumberAXIS.java
Package name: DefaultNamespace
Compiling service code...
Generating WSDL...
Serviceport: RandomNumberAXIS_Wmq
java org.apache.axis.wsdl.Java2WSDL --input helpers\RandomNumberAXIS_Wmq.wsdl --
output helpers\RandomNumberAXIS_Wmq.wsdl --namespace RandomNumberAXIS_Wmq --loca
tion wmq:SOAP.RandomNumberAXIS@RANDOMQM?connectQueueManager=RANDOMQM --bindingNa
me RandomNumberAXISBindingSoap --servicePortName RandomNumberAXIS_Wmq RandomNumb
erAXIS
Generating and deploying server wsdd file...
Target dir: C:\RandomService\helpers\
Patching deploy.wsdd...
Patching undeploy.wsdd...
Removing temp.server directory...
Preparing listener...
Configuring MQ...
Generate and compile proxy code...
java com.ibm.mq.ma0r.tools.RunWSDL2Java --timeout -1 --output helpers -p Default
Namespace helpers\RandomNumberAXIS_Wmq.wsdl

C:\RandomService>

```

3.9.3 Write the client application

The sample application SQCS2Axis.cs from WebSphere MQ transport for SOAP is used and changed appropriately for our demonstration service. The code ends up virtually identical to our test client in 3.7, “A simple example with a Microsoft .NET Web Service” on page 36. Aside from the different classname and proxy object type the code is identical.

Example 3-26 Our simple client application

```
using System;

class RandomNumberAxisClient{
    [STAThread]
    static void Main(string[] args)
    {
        String randomType="int";
        try
        {
            MQSOAP.MQWebRequest.Register();

            RandomNumberAXISService rsobj = new RandomNumberAXISService();

            // Any first argument is used as the target Url
            if (args.GetLength(0) >= 1) rsobj.Url = args[0];

            // Any second argument is the type of the random number required
            if (args.GetLength(0) >= 2) randomType = args[1];

            String res = rsobj.getRandom(randomType);

            Console.WriteLine("Random service returned: " + res);
        }
        catch (System.Exception e)
        {
            Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING RandomNumberOnAxis DEMO <<<\n" +
e.ToString());
        }
    }
}
```

The client application takes optional arguments for the URI and the type of the random number object to return. Note again the requirement to register the WebSphere MQ transport with the call to `MQSOAP.MQWebRequest.Register()`. This is highlighted in bold.

The client application is compiled. In this case the command line is used:

Example 3-27 Compilation of the client code

```
\RandomService>csc /lib:c:\ma0r\bin /r:MQSOAP.dll RandomNumberAxisClient.cs
helpers\RandomNumberAXISService.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.
```

```
C:\RandomService>
```

3.9.4 Additional WebSphere MQ configuration

The service is first tested on a local machine. A response queue called SOAP.RESPONSE.RandomNumberAXIS is manually created as the deployment process does not create a service specific queue for response messages. This can be done by using runmqsc or the WebSphere MQ Explorer utility.

3.9.5 Start the prepared JMS listener

The script for the JMS listener is prepared by the deployment utility in the helpers sub-directory. It is started manually in a new window. The CLASSPATH must be correctly set up first for WebSphere MQ transport for SOAP in this window.

Example 3-28 Starting the prepared JMS listener

```
C:\RandomService\helpers>listen_SOAP.RandomNumberAXIS
```

```
C:\RandomService\helpers>rem - generated by deployWMQService.java at 15-Jul-03 1
6:29:15
```

```
C:\RandomService\helpers>call C:\ma0r\bin\setcp.bat
```

```
C:\RandomService\helpers>cd /d C:\RandomService
```

```
C:\RandomService>java com.ibm.axis.transport.wmq.SimpleJMSListener -u "wmq:SOAP.
RandomNumberAXIS@RANDOMQM?connectQueueManager=RANDOMQM" -T 1
```

Starting Axis JMS listener.

listeners initialised.

```
Parameters: -u wmq:SOAP.RandomNumberAXIS@RANDOMQM?connectQueueManager=RANDOMQM -
c . -f null -s null -a false -T 1
```

```
*****
* Hit Enter to stop the listener and close this window *
*****
```

3.9.6 Test the service

The prepared client can now be executed. This is executed passing the target URI and the type of floating number (int or double) as command line options:

Example 3-29 Execution of our client application

```
C:\RandomService>RandomNumberAxisClient wmq:SOAP.RandomNumberAXIS@RANDOMQM?reply
ToQueue=SOAP.RESPONSE.RandomNumberAXIS,connectQueueManager=RANDOMQM int
Using server bindings.
Random service returned: 690665
```

```
C:\RandomService>RandomNumberAxisClient wmq:SOAP.RandomNumberAXIS@RANDOMQM?reply
ToQueue=SOAP.RESPONSE.RandomNumberAXIS,connectQueueManager=RANDOMQM double
Using server bindings.
Random service returned: 0.4638977929993522
```

```
C:\RandomService>RandomNumberAxisClient wmq:SOAP.RandomNumberAXIS@RANDOMQM?reply
ToQueue=SOAP.RESPONSE.RandomNumberAXIS,connectQueueManager=RANDOMQM wrong
Using server bindings.
Random service returned: Error
```

```
C:\RandomService
```

The service is called three times, once to request an integer random number, once to request a floating point random number and once with an illegal type option.

3.9.7 Distributed test in WebSphere MQ server bindings mode

The technique to make the service distributed, using WebSphere MQ distributed communications are exactly the same as for the Microsoft .NET case. This is covered in “Distributed test in WebSphere MQ server bindings mode” on page 47 and the reader is referred to that section for more detail. The redeployment process is executed as follows:

Example 3-30 Redeployment for distributed test

```
C:\RandomService>\ma0r\bin\setcp
```

```
C:\RandomService>deployWMQService -m WSServer -f RandomNumberAXIS.java
Package name: DefaultNamespace
```

```

Compiling service code...
Generating WSDL...
Serviceport: RandomNumberAXIS_Wmq
java org.apache.axis.wsdl.Java2WSDL --input helpers\RandomNumberAXIS_Wmq.wsdl --
output helpers\RandomNumberAXIS_Wmq.wsdl --namespace RandomNumberAXIS_Wmq --loca
tion wmq:SOAP.RandomNumberAXIS@WSServer?connectQueueManager=WSServer --bindingNa
me RandomNumberAXISBindingSoap --servicePortName RandomNumberAXIS_Wmq RandomNumb
erAXIS
Generating and deploying server wsdd file...
Target dir: C:\RandomService\helpers\
Patching deploy.wsdd...
Patching undeploy.wsdd...
Removing temp.server directory...
Preparing listener...
Configuring MQ...
Generate and compile proxy code...
java com.ibm.mq.maOr.tools.RunWSDL2Java --timeout -1 --output helpers -p Default
Namespace helpers\RandomNumberAXIS_Wmq.wsdl

C:\RandomService>

```

The listener is started on the server machine using the script file just generated by the deployment procedure:

Example 3-31 Starting the JMS Listener on the server machine

```

C:\RandomService\helpers>listen_SOAP.RandomNumberAXIS

C:\RandomService\helpers>rem - generated by deployWMQService.java at 16-Jul-03 0
9:53:48

C:\RandomService\helpers>call C:\maOr\bin\setcp.bat

C:\RandomService\helpers>cd /d C:\RandomService

C:\RandomService>java com.ibm.axis.transport.wmq.SimpleJMSListener -u "wmq:SOAP.
RandomNumberAXIS@WSServer?connectQueueManager=WSServer" -T 1

Starting Axis JMS listener.

listeners initialised.
Parameters: -u wmq:SOAP.RandomNumberAXIS@WSServer?connectQueueManager=WSServer -
c . -f null -s null -a false -T 1

*****
* Hit Enter to stop the listener and close this window *
*****

```

On the client machine there is no need to recompile the client code. However the proxy code still contained the URI of the original deployment. This needs to be changed to specify the new WebSphere MQ attributes for the server bindings connection. It is good practice to copy the updated proxy code from the deployment process back to the client, or edit the existing proxy to change the default URI setting. For simplicity the URI is specified at runtime with the command line option as follows:

Example 3-32 Running the demo with server bindings

```
C:\RandomService>RandomNumberAxisClient -wmq:SOAP.RandomNumberAXIS@WSServer?reply
ToQueue=SOAP.RESPONSE.RandomNumberAXIS,connectQueueManager=WSClient -int
Using server bindings.
Random service returned: 617338

C:\RandomService>
```

3.9.8 Distributed test in WebSphere MQ client mode

The technique to configure the simple test to use a client mode connection is exactly the same as for the Microsoft .NET case. This is covered in “Distributed test in WebSphere MQ client mode” on page 45 and the reader is referred to that section for more detail. To run the test in client mode, the URL on the client machine as follows needs to be modified:

Example 3-33 Running the demo with client bindings

```
C:\RandomService>RandomNumberAxisClient -wmq:SOAP.RandomNumberAXIS?clientConnecti
on=ITSOI.hursley.ibm.com(1415),clientChannel=T0.WSSERVER,replyToQueue=SOAP.RESPON
SE.RandomNumberAXIS -int
Using client bindings.
Random service returned: 826765

C:\RandomService>
```

3.9.9 Distributed WebSphere MQ using MQ clustering

This technique is the same as described in 3.7.10, “Distributed WebSphere MQ using MQ clustering” on page 48.

3.9.10 Service code use of external classes

In any non trivial Web Service it is likely that additional classes will be used in addition to those defined by WebSphere MQ transport for SOAP. In the case of the ShareQuote service in our business scenario for example, some classes are

needed to help format XML statements. (These are jdom.jar, jaxen-core.jar, jaxen-jdom.jar and saxpath.jar). For convenience these are installed into a directory called “lib” within the service directory. It is convenient to contain all the required classes within the same directory structure.

Because these classes were referenced in the service code, they needed to be in the CLASSPATH from the process where the JMSListener was invoked. This is most easily facilitated by creating an additional script containing the extra definitions and then amending the listener bat file to make a call to it after the normal WebSphere MQ transport for SOAP setcp.bat call:

Example 3-34 Amending a listener script to define external classes

```
rem - generated by deployWMQService.java at 04-Jul-03 12:16:17
call C:\ma0r\bin\setcp.bat
call ..\yubank_setcp.bat
cd /d C:\ma0r\tony_040603
java com.ibm.axis.transport.wmq.SimpleJMSListener -u
"wmq:SOAP.ShareQuote@DOTIP?connectQueueManager=DOTIP" -T 1
```

In this example we named the additional script yubank_setcp.bat. Inserting the call to our own specific CLASSPATH script in this way is convenient when starting the JMS listener, but every time the service is redeployed the listener start-up script has to be reedited to add it in again. Some care may be needed if using relative pathnames to refer to a directory containing external jar files as the deployment directory is generally one level higher than the directory from where the listener is invoked.

Note: It is also necessary for the additional service classes to be set into the CLASSPATH at the time of service deployment.

3.10 Starting listeners with WebSphere MQ triggering

Whilst writing this redbook, the Microsoft .NET and JMS listeners were started manually during testing. However it is also possible to use WebSphere MQ triggering to start these listeners automatically when they are required. The deployment procedure automatically creates an initiation queue called SOAP.INITQ and creates process definitions that can start the listeners when messages first arrive on a request queue. Triggering is more likely to be appropriate for a production environment where there is a potential scope for starting many listeners and can be a waste of resource to keep them all running permanently.

After creating the process definition, the deployment sets the necessary triggering attributes when it creates the request queue for a particular service. These attributes are shown in bold below:

Example 3-35 Illustrating the triggering attributes set on a request queue by the deployment utility

```
display qlocal('SOAP.RandomNumberNET')
  6 : display qlocal('SOAP.RandomNumberNET')
AMQ8409: Display Queue details.
  DESCR(WebSphere MQ Default Local Queue)
  PROCESS(RandomNumberNET.PROCESS)      BOQNAME( )
  INITQ(SOAP.INITQ)                     TRIGDATA( )
  CLUSTER( )                            CLUSNL( )
  QUEUE(SOAP.RandomNumberNET)           CRDATE(2003-07-15)
  CRTIME(18.25.53)                       ALTDATA(2003-07-15)
  ALTTIME(18.25.53)                       GET(ENABLED)
  PUT(ENABLED)                           DEFPRTY(0)
  DEFPSIST(NO)                           MAXDEPTH(5000)
  MAXMSGL(4194304)                       BOTHRESH(0)
  SHARE                                 DEFSOPT(SHARED)
  HARDENBO                               MSGDLVSQ(PRIORITY)
  RETINTVL(999999999)                   USAGE(NORMAL)
  TRIGGER                                TRIGTYPE(FIRST)
  TRIGDPTH(1)                           TRIGMPRI(0)
  QDEPTHHI(80)                           QDEPTHLO(20)
  QDPMAXEV(ENABLED)                       QDPHIEV(DISABLED)
  QDPLQEV(DISABLED)                       QSVCINT(999999999)
  QSVCIEV(NONE)                           DISTL(NO)
  DEFTYPE(PREDEFINED)                     TYPE(QLOCAL)
  SCOPE(QMGR)                             DEFBIND(OPEN)
  IPPROCS(0)                              OPPROCS(0)
  CURDEPTH(0)
```

These triggering settings cause a message to be written to the initiation queue SOAP.INITQ when the message depth on the request queue changes from zero to one. This message sent to SOAP.INITQ is a message requesting the runmqtrm utility to start the process named by the PROCESS parameter on the server system. For the RandomNumberNET service for example, this PROCESS parameter is defined by the deployment utility as follows:

Example 3-36 Illustrating the WebSphere MQ process definition for a service listener

```
4 : display process('RandomNumberNET.PROCESS')
AMQ8407: Display Process details.
  DESCR( )
  APPLICID(start "WMQAsmxListener - RandomNumberNET" /min C:\RandomServiceNET\h
elopers\listen_RandomNumberNET.bat)
  USERDATA( )                           ENVRDATA( )
  PROCESS(RandomNumberNET.PROCESS)      ALTDATA(2003-07-15)
```

To activate the triggering function it is necessary to start the trigger monitor on the server machine. This can either be accomplished by executing the runmqtrm command directly, or by running deployWMQservice with the “-c startWMQMonitor” option. This is an example of starting the trigger monitor command manually:

Example 3-37 Starting a trigger monitor on the service system

```
C:\Documents and Settings\stevens\Desktop>runmqtrm -m WSServer -q SOAP.INITQ
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
WebSphere MQ trigger monitor started.
```

Waiting for a trigger message

This is an example of starting the trigger monitor via deployWMQService:

Example 3-38 Starting a trigger monitor via the deployment utility

```
C:\RandomServiceNET>\ma0r\bin\setcp

C:\RandomServiceNET>deployWMQService -m WSServer -c startWMQMonitor
+++ server: runmqtrm -m WSServer -q SOAP.INITQ

C:\RandomServiceNET>
```

The deployment utility starts the trigger monitor minimized so as to minimize screen clutter. The process definitions for the listeners are also set to start the listeners in minimized windows.

3.10.1 Using a different initiation queue

The trigger monitor queue “SOAP.INITQ” is hard coded into the deployment process. To use other names for the trigger monitor queue it is necessary to define the queue first and then modify the process definition set up by the deployment utility to use the new queue as the initiation queue instead of SOAP.INITQ. This queue needs to be specified to the runmqtrm command with the -q parameter. This is illustrated in the following three steps:

Create the new initiation queue

This is done with the runmqsc command or the Explorer Graphical User Interface (GUI) in the normal way:

Example 3-39 Creating the new initiation queue

```
C:\RandomServiceNET> runmqsc WSServer
5724-B41 (C) Copyright IBM Corp. 1994, 2002.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager WSServer.
```

```
define qlocal('SOAP.RandomNumber.INITQ')
  1 : define qlocal('SOAP.RandomNumber.INITQ')
AMQ8006: WebSphere MQ queue created.
end
  2 : end
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.

C:\RandomServiceNET>
```

Change the request queue to use the new initiation queue

This step is also performed with runmqsc or the Explorer GUI:

Example 3-40 Changing the request queue to use the new initiation queue

```
C:\RandomServiceNET> runmqsc WSServer
5724-B41 (C) Copyright IBM Corp. 1994, 2002.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager WSServer.
```

```
alter qlocal('SOAP.RandomNumberNET') INITQ('SOAP.RandomNumber.INITQ')
  1 : alter qlocal('SOAP.RandomNumberNET') INITQ('SOAP.RandomNumber.INITQ')
AMQ8008: WebSphere MQ queue changed.
end
  2 : end
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

Run the trigger monitor with the new initiation queue

The trigger monitor is started with the runmqtrm command, specifying the new initiation queue with the -q argument:

Example 3-41 Illustrating how to change to another initiation queue (this is how the output looks)

```
C:\RandomServiceNET>runmqtrm -m WSServer -q SOAP.RandomNumber.INITQ
5724-B41 (C) Copyright IBM Corp. 1994, 2002.  ALL RIGHTS RESERVED.
WebSphere MQ trigger monitor started.
```

Waiting for a trigger message

```
start "WMQAsmxListener - RandomNumberNET" /min C:\RandomServiceNET\helpers\listen_RandomNumberNET.bat "TMC 2SOAP.RandomNumberNET
RandomNumberNET.PROCESS
start \"WMQAsmxListener - RandomNumberNET\" /
min C:\RandomServiceNET\helpers\listen_RandomNumberNET.bat
```

WSServer

"

End of application trigger.

Waiting for a trigger message

3.11 WebSphere MQ transport for SOAP and SSL

As described in “SSL options” on page 30, WebSphere MQ transport for SOAP provides three SSL options that may be specified in the WebSphere MQ URI for use with client connections over a channel configured to run in SSL mode. These three options are:

1. SSLKeyRepository - specifies the key repository in “stem” format
2. SSLCipherSpec - specifies the SSL cipher scheme used on the channel
3. SSLPeerName - specifies peer name for Distinguished Name filtering. (optional)

The key repository option is specified in “stem” format, this means that the full path is given to the key repository but without the extension. Therefore, if the full path name to the repository is “c:\ssl\key.sto”, the SSLKeyRepository attribute would be given as “c:\ssl\key”.

These SSL attributes only apply to use from WebSphere MQ clients where no use is made of a local queue manager. Where local queue managers are used on both client and host machines, SSL attributes would be set on the channel in the usual manner.

By default, the SSL option “Always authenticate parties initiating connections to this channel definition” is set when enabling SSL on the channel. This means that clients are required to authenticate themselves before SSL communication can commence. They do this by sending their certificate to the server system. If this

option is not set, then SSL communications are established without client authentication. If using client authentication, it is important that the client's keystore has assigned the same test certificate as is assigned on the channel.

The `SSLPeerName` is optional. For additional security, WebSphere MQ channels can be configured to only accept certificates whose Distinguished Names match a required set of values. If this requirement has been set on a channel, the client's certificate must then match the required Distinguished Name attributes.

Refer to Chapter 12, "Security" on page 249 for more information about SSL.

Note: The use of the above SSL options are only available with Microsoft .NET client applications and services.

3.11.1 Simple demonstration with SSL

To demonstrate the use of SSL the simple demonstration program in "A simple example with a Microsoft .NET Web Service" on page 36 is taken where a WebSphere MQ client connection to the `RandomNumber` service is tested and deployed as a Microsoft .NET Web Service.

To run this client in SSL mode the following configuration steps are taken:

- ▶ Import the appropriate certificate store into WebSphere MQ and assign it to the queue manager.
- ▶ Create a key repository for use from the client.
- ▶ Set the SSL attributes on the target channel.
- ▶ Add the appropriate SSL options to the WebSphere MQ URI. (the client command line option to do this is used in this case).

On the client machine the keystore is prepared. This is located in this particular instance in the file "`c:\$user\Certificates\key\key.sto`", but it is generated wherever it is required. The certificate is imported into the queue manager on the server system and assigned it as the queue manager's certificate. It is then necessary to set the Cipher specification attribute on the channel "`TO.WSSERVER`" to the required SSL cipher. In this case "`NULL_MD5`" is chosen. Once these steps are completed, then add the two SSL attributes to the WebSphere MQ URI in order to run the test in SSL mode:

Example 3-42 Demonstrating the use of SSL WebSphere MQ URI options

```
C:\RandomServiceNET>RandomNumberNetClient  
wmq:SOAP.RandomNumberNET?clientConnection=ITSOI.hursley.ibm.com(1415),clientChannel=TO.WSSERVER  
,replyToQueue=SOAP.RESPONSE.RandomNumberNET,sslcipher=spec=NULL_MD5,sslkeyrepository="c:\$user  
\Certificates\key\key" int
```

Using client bindings.
Random service returned: 825688

```
C:\RandomServiceNET>RandomNumberNetClient wmq:SOAP.RandomNumberNET?clientConnect
ion=ITSOI.hursley.ibm.com(1415),clientChannel=T0.WSSERVER,replyToQueue=SOAP.RESP
ONSE.RandomNumberNET,sslCipherspec=NULL_MD5",sslkeyrepository="c:\$user\Certifi
cates\key\key" double
Using client bindings.
Random service returned: 0.856901188780042
```

```
C:\RandomServiceNET>RandomNumberNetClient wmq:SOAP.RandomNumberNET?clientConnect
ion=ITSOI.hursley.ibm.com(1415),clientChannel=T0.WSSERVER,replyToQueue=SOAP.RESP
ONSE.RandomNumberNET,sslCipherspec=NULL_MD5",sslkeyrepository="c:\$user\Certifi
cates\key\key" wrong
Using client bindings.
Random service returned: Error
```

C:\RandomServiceNET>

3.11.2 Use of SSLPeerName

The SSLPeerName URL attribute is used to specify any required SSL distinguished name. This is specified in the usual form with equals (“=”) characters to separate attribute names from their values and comma (“,”) characters to separate different name and value pairs. The only difference is that entire distinguished name string has to be enclosed in parentheses. This is shown in the following example:

```
SSLPeerName="(CN=MQ Test 1,O=IBM,S=Hampshire,C=GB)"
```

The Common Name attribute does not need to be self-contained within embedded quotes if it contains spaces.

An example in practice with our Microsoft .NET RandomNumber client application is given below:

Example 3-43 Example use of SSLPeerName

```
C:\RandomServiceNET>RandomNumberNETClient wmq:SOAP.RandomNumberNET?clientConnect
ion=localhost,clientChannel=ANSSLCHANNEL,replyToQueue=SOAP.RESPONSE.RandomNumber
NET,sslCipherspec=NULL_MD5",sslkeyrepository="c:\$user\Certificates\key\key",SS
LPeerName="(CN=MQ Test 1,O=IBM,S=Hampshire,C=GB)" int
Connecting to queue manager
Connected to QM
Using client bindings.
Random service returned: 605520
```

3.12 Asynchronous invocation of Web Services

The current edition of the WebSphere MQ Transport for SOAP SupportPac provides prototype facilities for long term asynchronous invocation of Web Services. These facilities were not tested or used during the course of writing this redbook, but some brief notes are provided here for reference.

Long term asynchrony refers to the ability for a client to be able to invoke a service request from one process and then retrieve the response from the service in a completely separate process. Such a facility may be required where it is not practical or efficient to prolong the lifetime of a client until the response is received. Some services may not be designed to return a response immediately but may return a set of responses at some future point. Other services may be able to process the request immediately but might then be unable to return the response owing to unstable network connections. For both these situations, the ability to be able to decouple the processing of the client's request and response across separate processes would be a valuable option.

To use this long term asynchrony facility, the client becomes somewhat more complicated. It first registers its intent to invoke services asynchronously by calling, in the Microsoft .NET case, the method `MQSOAP.Async.Request()`. It is necessary to pass to this method a reference to an object that has been derived from the class `MQSOAP.Async.CallBack` and which overrides the function `CallBackFunction()`. For example:

Example 3-44 Long term asynchronous client invocation of a Web Service

```
// Create a context object
testStateClass requestContext = new testStateClass();

MQSOAP.Async.Request(requestContext);
```

An example class definition for the callback object might be as follows:

Example 3-45 Example callback class used in long term asynchronous invocation

```
[Serializable]
class testStateClass : MQSOAP.Async.CallBack
{
    public override void CallBackFunction()
    {
        StockQuoteDotNet stockobj = new StockQuoteDotNet();
```

```

MQSOAP.Async.Response(this);

System.Single res = stockobj.getQuote("ZZZ");
Console.WriteLine("ASYNC response is: " + res);

MQSOAP.MQResponseListener.stopListener();
    }
}

```

The function `CallBackfunction()` is called in the response process if and when a reply has been received from the service.

The process is more involved than it might first seem. The callback object must first be serialized and then written to a dedicated WebSphere MQ “side” queue called `MQSOAP.SIDE.QUEUE`. The response process then starts a listener process by creating an `MQSOAP.MQStartResponseListener` object. For example:

Example 3-46 Starting the response listener from the client's response process

```

// Create a proxy object so we know what URL to point the response listener to
StockQuoteDotNet stockobj = new StockQuoteDotNet();

// Now start the response listener
MQSOAP.MQStartResponseListener sl = new MQSOAP.MQStartResponseListener(stockobj.Url);

```

This listener monitors the response queue for the service until the reply is received. It then retrieves the corresponding entry on the side queue, reconstitutes the callback object and then calls its `CallBackFunction()` so that this may then retrieve the response message. Before the `CallBackFunction` can do this it must first register that it wishes to receive an asynchronous response by calling the method `MQSOAP.Async.Response()`. It then makes a dummy invocation of the same target service. The transport code detects it is actually being requested to retrieve a waiting asynchronous response and then proceeds to retrieve the waiting response message without actually reinvoking the service.

Similar facilities for long term asynchrony have also been provided for the J2EE environment. However, it is not currently possible to make a long term asynchronous invocation between the Microsoft .NET and J2EE environments.

Note that the side queue is automatically created by the script `setupmq.bat` that is executed when the IVT is first run. The name of the side queue is currently fixed.

These facilities are intended for long term asynchrony. Where it is required to provide facilities for short term asynchrony (that is, asynchronous operation within

the context of a single client process) within the Microsoft .NET environment, more direct use can be made of the existing Microsoft .NET interfaces. This is illustrated in the client application WMQSoapGui provided with WebSphere MQ Transport for SOAP.

Refer to the documentation provided with WebSphere MQ Transport for SOAP for more details on long term asynchrony.

3.13 Current status and future plans

At the time of writing this redbook, WebSphere MQ transport for SOAP is available as a SupportPac and may be directly downloaded from the IBM WebSphere MQ SupportPacs home page. The current released version of this SupportPac (dated October 2003) is a “Category 2” SupportPac. It is unsupported by IBM. The October 2003 release offers the following enhancements over earlier versions:

- ▶ Installation options for Windows, Linux and AIX, HP-UX and Solaris.
- ▶ Options to use multithreaded Axis and Microsoft .NET listeners for improved performance.
- ▶ Compatibility with a supported release of WebSphere MQ classes for Microsoft .NET. Earlier versions of WebSphere MQ transport for SOAP used unsupported versions of these classes that were embedded directly within the SupportPac. They will be referenced as an externally supported component of WebSphere MQ.
- ▶ Compatibility with Microsoft .NET framework redistributable and SDK version 1.1.
- ▶ Embedded libraries for Apache Axis upgraded to V 1.1 and Xerces upgraded to V2.5.
- ▶ Inclusion of persistence, priority, expiry and SSL options in a WebSphere MQ URL.
- ▶ Availability of WebSphere MQ client connections.
- ▶ The ability to use a single JMS listener in conjunction with multiple services deployed from the same directory. This will avoid having to always activate one listener per service. This option will only be available for J2EE Web Services and not for Microsoft .NET services. This is because the Apache Axis infrastructure automatically passes the prerequisite information about the target service in the SOAP message, while the Microsoft .NET infrastructure does not.
- ▶ Resolution of triggering problems on Linux. Triggering of listeners on Linux did not work correctly in the earlier February 2003 release.

- ▶ Prototype facilities for long term asynchronous invocation of Web Services. Previously, asynchrony was only possible within a specific Microsoft .NET client instance and could not extend beyond the life of the process within which it was invoked. There were no options at all for asynchrony for Java based services. In the current release asynchronous options have been provided for both the environments that allow for asynchronous operation beyond the lifetime of a specific client instance. See 3.12, “Asynchronous invocation of Web Services” on page 67 for more details.
- ▶ Various performance and reliability enhancements
- ▶ Improved documentation

IBM is planning to move this technology into a supported state. It is intended to achieve this by including WebSphere MQ transport for SOAP into a future Customer Service Diskette (CSD) update for WebSphere MQ. At that point WebSphere MQ Transport for SOAP will be a fully supported component of WebSphere MQ.

Before this CSD version is released the October 2003 edition of the SupportPac may be updated. If so, the refreshed version most likely will also be a Category 2 unsupported SupportPac but will be closer to the first supported CSD release.

The following functional improvements may be included in any further category 2 SupportPac or in the first supported release:

- ▶ Provision for one way messages. WebSphere MQ transport for SOAP is currently fashioned on a request/response model only and there is no provision for one way. One way messages are invocations of a service where no SOAP response message is expected at a client.
- ▶ Provision for report messages. These might be used to perform activities such as status switching of a particular service.
- ▶ Installation options for an extended range of platforms.
- ▶ An option for a Java only Windows installation. The .NET Framework and the Framework SDK are currently a prerequisite for WebSphere MQ transport for SOAP, even if Java support only is required. This prerequisite needs to be removed.
- ▶ Options for transacted operations.
- ▶ The ability to use binary attachments with WebSphere MQ transport for SOAP messages.
- ▶ The ability to use Java Web Services coded as .jws files.
- ▶ Tighter specification of the WebSphere MQ transport for SOAP standard. This will include tighter definition of parameters permitted in the 'wmq:' URI, how they map onto details of the WebSphere MQ messages, and how the clients

and listeners use these details, especially in error situations. This will make it clearer to implement custom applications, and will improve interoperability.

Note: We recommend that the most current version of the SupportPac is installed before using the techniques described in this redbook. The CSD release should also be installed as soon as possible once it becomes available.



Business case scenario

This chapter provides an overview of a bank used as a business case scenario to illustrate the use of the latest technical features in the WebSphere MQ product. Due to time and resource constraints, this scenario is intentionally simplified and does not claim completeness or overall soundness. However, it gives an understanding of how to use WebSphere MQ as a transport mechanism for messaging and Web Services invocation from .NET applications.

4.1 Business domain

A bank called YuBank provides a variety of banking services to its customers and plans to expand its services to include brokerage accounts and investment advisory services. The new service allows a customer to open an investment account and create a customer profile. A questionnaire is provided to the customer to capture various customer's financial information, such as the investment amount, risk level, investment period, return expectation, interested industries, existing assets, current debts, family income and so on. Based on the investment criteria and intelligent analysis, the advisory system generates a portfolio recommendation. If the customer accepts the recommendation, the trade order is dispatched to a securities transaction system. The customer is notified automatically after the trade order is executed, together with the latest portfolio snapshot. A customer may also submit a request to the advisory system for a new portfolio recommendation later on.

4.2 Business process

The banking system contains several independent, but cooperating modular subsystems, a design that encourages using only those parts in the system that are needed for a particular business requirement, and facilitates distributing development to teams each responsible for a separate subsystem. The major subsystems include the Bank Service System, Credit Check System, Customer Profile System, Investment Advisory system, and Share Quote System. Figure 4-1 illustrates the basic flow context in the business process. The solution implementations in this book only include the primary subsystems described in the foregoing section. The use cases are defined in details in the following sections.

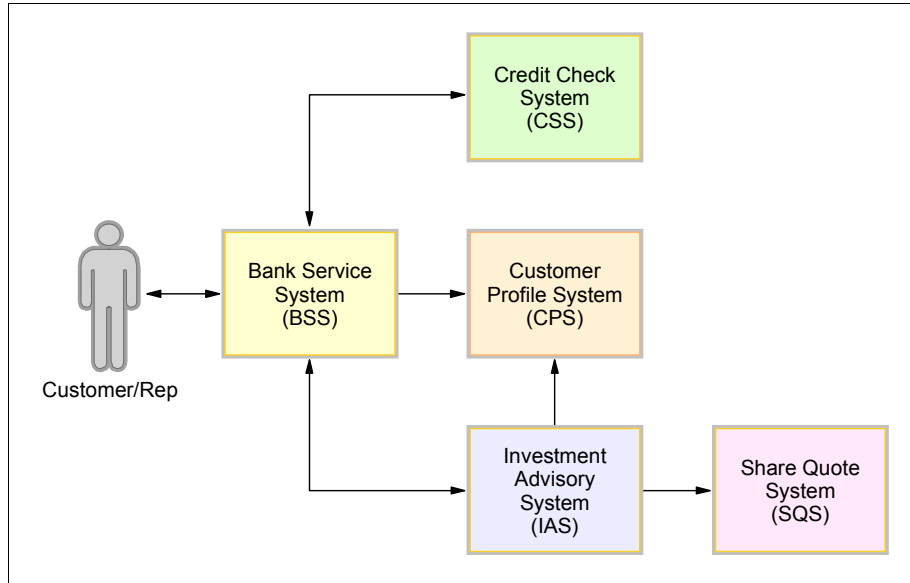


Figure 4-1 Business process diagram

4.2.1 Use case 1: Account opening

To open a new investment account, a customer or a service representative uses a Web browser to access the Bank Service System (BSS), a Web-based application, to enter the customer information. Multiple pages or screens are used to capture all financial data. All significant information is collected in the first page or screen, and the necessary processing starts in parallel at the back end, while the customer continues entering data in the subsequent pages or screens. A customer unique identifier and identifier type are required for credit check. For example, a social security number is needed for a customer in the United States. Additional information like date of birth, name and address are also needed for credit check. The application communicates with the Credit Check System (CCS), typically an external credit bureau system, to obtain the credit score(s) and credit history for the applicant. The data submitted to CCS is composed of the customer name, unique identifier, identifier type, date of birth, and address in eXtensible Markup Language (XML) format. After the credit result is retrieved, a set of business rules are applied in the application to analyze whether the applicant is qualified to open a new investment account with the bank. All information collected are used for detailed risk analysis. Even though a candidate may have a low credit score at present, if his or her credit history is satisfactory and his or her financial status (assets, debts and income) are good, the applicant may be accepted.

If the qualification of an applicant meets the bank requirements, a new investment account is created for the customer. And a new customer profile is generated in the Customer Profile System (CPS). All customer information such as the name, address, telephone number and so on is stored in his or her profile, together with the new account number as well as the investment information.

4.2.2 Use case 2: Investment advisory

After the investment account is set up, the customer or the service representative may submit a request to the Investment Advisory System (IAS) for portfolio recommendations, using BSS. IAS pulls the customer information from the CPS. IAS conducts an intelligent analysis, and communicates with the Share Quote System (SQS) to retrieve stock prices and historical data. Then IAS further analyzes the stock data to find a best fit to the customer's expectations. Finally, when the portfolio recommendation result is available, the outcome is dispatched to BSS in an asynchronous manner.

The intelligent analysis involves artificial intelligence, fuzzy logic and optimization search as well as human interactions such as analyst's recommendations and overriding. For simplicity in this demonstration system, simplified logic is used instead. The SQS system provides the stock performance history in periods of 1, 3, 6, 9, and 12 months in the unit of the return percentage. IAS uses a weighted formula to calculate a yearly average return percentage:

$$(1M \times 12 + 3M \times 4 + 6M \times 2 + 9M \times 4/3 + 12M \times 1) / 5.$$

Based on the average value, the biggest variance can be derived among these five weighted yearly returns. The amount of the biggest variance divided by the average reflects the fluctuation, and this value should fall into the range that matches the risk level the customer desires.

For example, a stock's historical performance is as follows:

1%, 2%, 2%, 6% and 8% for the period of 1, 3, 6, 9 and 12 months, respectively. The average yearly return is:

$$(1\% \times 12 + 2\% \times 4 + 2\% \times 2 + 6\% \times 4/3 + 8\% \times 1) / 5 = 8\%.$$

The biggest variance is:

$$1\% \times 12 - 8\% = 4\% \text{ (or } 8\% - 2\% \times 2 = 4\%).$$

Therefore, the fluctuation is:

$$4\% / 8\% = 50\%.$$

4.3 Non-functional requirements and assumptions

To simplify the design, facilitate delivery of a demonstration application and focus on the asynchronous transport feature of WebSphere MQ in .NET environment, the following requirements and assumptions are defined:

- ▶ The BBS is a Web-based application written in C#. A user accesses the BBS through a standard Web browser, such as Microsoft Internet Explorer or Netscape Navigator. The application is built for existing customer services. The new features of investment account opening and advisory are added to this application.
- ▶ The CCS is a .NET server application written in C#, there is also a VB.NET version.
- ▶ The CPS is a J2EE server application written in Java.
- ▶ The IAS is a .NET Web Services application written in C#.
- ▶ The SQS is a J2EE Web Services application written in Java.
- ▶ The trading system design is out of scope.
- ▶ All systems for Web Services in this context use the Basic Profile 1.0 defined in WS-I standard:
 - Extensible Markup Language (XML) Schema 1.0
 - Simple Object Access Protocol (SOAP) 1.1
 - Web Services Description Language (WSDL) 1.1
 - Universal Description, Discovery and Integration (UDDI) 2.0
- ▶ All communication transport mechanisms are asynchronous using WebSphere MQ.
- ▶ Necessary data protection is implemented in the inter-system communications. Secure Sockets Layer (SSL) is a viable option at the transport layer.
- ▶ Part of the process is implemented in a transaction as deemed necessary.
- ▶ No firewall is included in the cross-network communications.
- ▶ Scalability, availability and disaster recovery are not addressed.
- ▶ Minimum system management and monitoring is provided.
- ▶ Security such as user authentication, authorization, confidentiality, integrity and nonrepudiation is not included.
- ▶ Maintainability and extensibility of functionality are not covered.
- ▶ Reliability is not in the scope.
- ▶ Minimum exception handling is implemented to deal with errors resulted from incorrect data formats, insufficient data elements, and invalid data contents.



Solution design

This chapter provides information about the solution design of our business case scenario. It begins by describing the message flow for the two use cases. The design contains WebSphere MQ message types and message persistence. Next it shows the server configuration of YuBank and the frameworks used. Finally the chapter illustrates the WebSphere MQ network in detail.

5.1 Message flow

There are five applications involved in the business process. The process can be divided into two separated message flows, one for each use case. Some applications are involved in both flows.

Figure 5-1 shows where data has to be exchanged between applications. The numbers in the diagrams show the logical order of the data transfer within the two use cases.

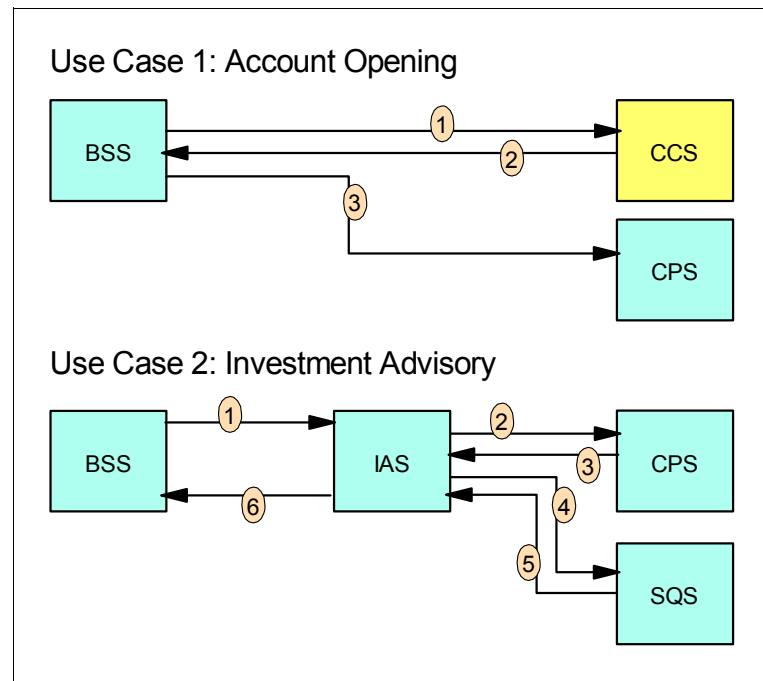


Figure 5-1 Message flows

5.1.1 Use case 1: Account opening message flow

The account opening message flow can be divided into three steps:

1. To open a new investment account, the Banking Service System (BSS) first has to send a request message to the external Credit Check System (CSS). The message has to contain information to identify the customer. This message does not necessarily have to be persistent because, it can be resent if the service is currently unavailable.

2. The CCS sends back a reply with the credit information pertaining to the customer. This reply has to be persistent, because the credit check is a paid external service.
3. After the BSS has received the reply with the credit score, it sends a datagram to the Customer Profile System (CPS) to store the customer information and the credit rating. The message also has to be persistent, because the data within it is time consuming and expensive to recreate. Since assured message delivery is provided by WebSphere MQ, no reply or positive action notification from the CPS is required.

5.1.2 Use case 2: Investment advisory message flow

The investment advisory message flow can be divided into six steps:

1. To receive a portfolio recommendation, the Banking Service System (BSS) invokes the Investment Advisory System (IAS) Web Service using WebSphere MQ transport for SOAP. The SOAP request contains account and investment information. This request message does not necessarily need to be persistent, but WebSphere MQ transport for SOAP uses the same persistence for the request and the corresponding response. The response contains valuable data and has to be persistent, so the request has to be persistent too.
2. Whenever the IAS is invoked it sends a request message to the Customer Profile Service (CPS) containing the customers account number. For performance reasons this request can be non-persistent. If the message gets lost, the IAS can resend the request.
3. The CPS sends back the stored profile information in a reply message. Like the request, the response can be non-persistent for performance reasons.
4. When the IAS receives this reply, it invokes the Share Quote System (SQS) Web Service using WebSphere MQ transport for SOAP to receive all relevant stock prices. The SOAP request contains all share names.
5. The SQS returns a SOAP response to the IAS with the prices of the requested shares. Both SOAP request and response can be non-persistent. If one of the messages get lost, the SQS is invoked again to receive the latest prices.
6. The IAS analyzes the data and returns a SOAP response to the BSS containing the investment advice information. WebSphere MQ transport for SOAP requires that this message has the same persistence as the corresponding request.

5.2 Server configuration

YuBank uses a three tier architecture. Tier one is located at the bank's branch offices, tiers two and three are located at the head office. Our business case also requires communication with an external credit bureau system.

On tier one the graphical client application Banking Service System (BSS) is running in a .NET environment. In our implementation it runs on ITSOL.

On tier two the Web Service Investment Advisory System (IAS) is running in a .NET environment. The IAS runs on the server ITSOL.

Tier three is composed of two servers, ITSOO and ITSOE. The Java application Customer Profile System (CPS) is running on ITSOO in a J2EE environment. The Web Service Share Quote System (SQS) also runs in a J2EE environment on ITSOE. Both services access a data source on their machine.

YuBank has business to business connection with a credit bureau. This bureau offers a Credit Check System (CCS). This service is a server application running in a .NET environment on ITSOD.

Figure 5-2 shows our configuration of all servers with the applications and the environment on which they are running.

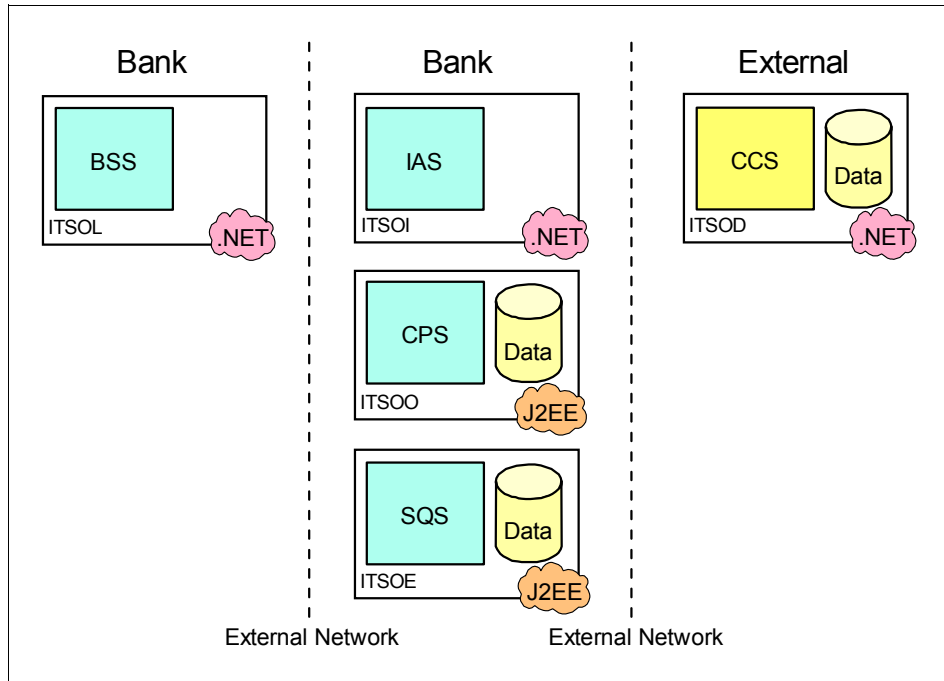


Figure 5-2 Server configuration

In our configuration we use five Intel® machines running Microsoft Windows 2000 with Service Pack 4. All machines are connected over a TCP/IP network.

5.3 WebSphere MQ configuration

The Banking Service System (BSS) currently runs a WebSphere MQ client for cost and administration reasons. All other systems have their own local queue manager.

The BSS on tier one is connected to the queue manager DOTIP on tier two using a WebSphere MQ client connection. The tier two queue manager DOTIP is connected to the tier three queue manager's DOTOP and DOTEP using sender-receiver channel pairs. Since there is no message transfer between the two, tier three queue managers, there is no need for a connection between them.

The connection between the BSS and the external queue manager DOTDP is a WebSphere MQ client connection. DOTDP is not connected to any queue manager within YuBank's head office.

Figure 5-3 shows all servers with their queue managers and the applications connected to them. It also illustrates the queue manager connections.

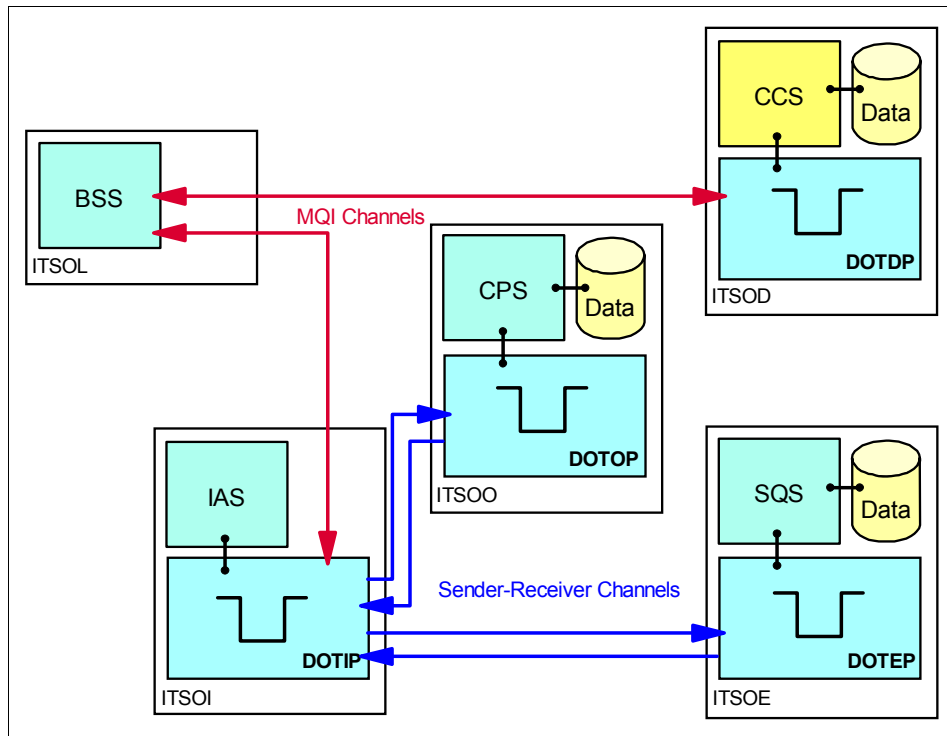


Figure 5-3 WebSphere MQ network

Figure 5-4 shows all WebSphere MQ objects and message channel agents defined in our queue manager network.

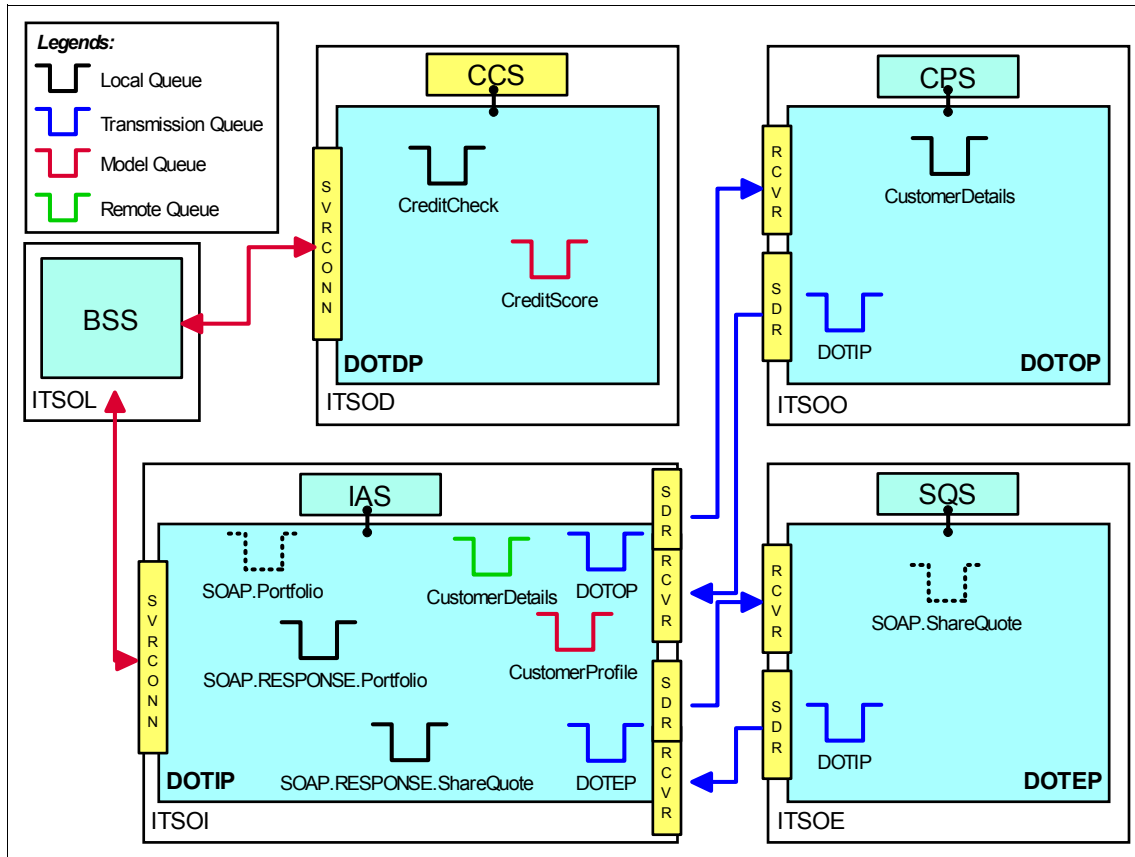


Figure 5-4 WebSphere MQ objects

The queue manager DOTDP is standalone. The following WebSphere MQ objects and message channel agents are defined:

- ▶ A server connection channel TO.DOTDP for the BSS clients to connect
- ▶ A local input queue called CreditCheck for the CCS application
- ▶ A model queue called CreditScore. This model queue is used by the BSS client application to generate permanent dynamic queues for the replies sent back by the CCS.

DOTIP is the middle tier queue manager. The following WebSphere MQ objects and message channel agents are defined:

- ▶ A server connection channel TO.DOTIP for the BSS clients to connect
- ▶ A sender channel DOTIP.TO.DOTOP for the queue manager connection to DOTOP

- ▶ A transmission queue DOTOP used by the sender channel DOTIP.TO.DOTOP
- ▶ A receiver channel DOTOP.TO.DOTIP
- ▶ A sender channel DOTIP.TO.DOTEP for the queue manager connection to DOTE
- ▶ A transmission queue DOTE used by the sender channel DOTIP.TO.DOTE
- ▶ A receiver channel DOTE.TO.DOTIP
- ▶ A local queue SOAP.RESPONSE.Portfolio for the SOAP response messages sent back from the Web Service Investment Advisory System (IAS)
- ▶ A remote queue CustomerDetails. It points to the queue CustomerDetails on DOTOP. This queue is needed, because the BSS client application has no direct connection to DOTOP.
- ▶ A model queue called CustomerProfile. This model queue is used by the BSS client application to generate temporary dynamic queues for the replies sent back by the CPS
- ▶ A local queue SOAP.RESPONSE.ShareQuote for the SOAP response messages sent back from the Web Service Share Quote System (SQS)
- ▶ The local input queue for the Web Service IAS called SOAP.Portfolio is generated by the WebSphere MQ transport for SOAP deployment utility

DOTOP is the first tier three queue managers. The following WebSphere MQ objects and message channel agents are defined:

- ▶ A sender channel DOTOP.TO.DOTIP for the queue manager connection to DOTIP
- ▶ A transmission queue DOTIP used by the sender channel DOTOP.TO.DOTIP
- ▶ A receiver channel DOTIP.TO.DOTOP
- ▶ A local input queue called CustomerDetails for the CPS application

DOTE is the second tier three queue managers. The following WebSphere MQ objects and message channel agents are defined:

- ▶ A sender channel DOTE.TO.DOTIP for the queue manager connection to DOTIP
- ▶ A transmission queue DOTIP used by the sender channel DOTE.TO.DOTIP
- ▶ A receiver channel DOTIP.TO.DOTOP
- ▶ The local queue for the Web Service SQS called SOAP.ShareQuote is generated by the WebSphere MQ transport for SOAP deployment utility

All queue managers use their local queue SYSTEM.DEAD.LETTER.QUEUE as the dead letter queue.

The setup of WebSphere MQ is described in detail in 6.3.1, “Core systems overview” on page 110.



Environment setup

This chapter describes the setup of the core systems used to implement the solution to our business case scenario. It also gives installation instructions and describes the hardware and software levels required on each platform in the business case scenario to obtain the configuration in Figure 5-2 on page 83.

6.1 Software prerequisites

The software required to implement the business case scenario solution includes:

- ▶ Windows 2000 Professional
- ▶ IBM WebSphere MQ V5.3
- ▶ Latest WebSphere MQ Customer Service Diskette (CSD)
- ▶ Microsoft Visual Studio .NET Professional 2003
- ▶ Internet Information Services (IIS)
- ▶ Microsoft .NET Framework redistributable (Provided with Microsoft Visual Studio .NET Professional)
- ▶ Java™ 2 SDK and Runtime environment, Standard Edition (build 1.3.1 or above)

6.2 Installation

This section gives installation, configuration and setup instructions.

6.2.1 Installing WebSphere MQ

WebSphere MQ can be installed using the installation guide provided with WebSphere MQ product.

Refer to *WebSphere MQ for Windows Quick Beginnings*, GC34-6073.

6.2.2 Installing WebSphere MQ classes for Microsoft .NET

This can be downloaded from the following URL:

<http://www-3.ibm.com/software/integration/support/supportpacs/individual/ma7p.html>

6.2.3 Installing WebSphere MQ Transport for SOAP

Refer 3.2.4, “Installation” on page 17 for installation instructions.

6.2.4 Installing Internet Information Services (IIS)

Internet Information Services (IIS) is a group of integrated services that is used to configure Windows 2000 as a Web server. This section gives instructions on installing IIS on Windows 2000 Professional.

Before Installation of Internet Information Services (IIS):

- ▶ Ensure Windows 2000 Server / Professional CD is available.
- ▶ Disconnect the machine from the network.
- ▶ Disable any firewall running.

For example, to disable Symantec firewall, click **Start -> Programs**, select **Symantec Desktop Firewall -> Symantec Desktop Firewall** and click **Disable**.



Figure 6-1 Disabling Symantec Desktop Firewall

After completing the pre-installation steps above:

- ▶ Select **Start -> Settings -> Control Panel**.
- ▶ Double-click **Add/Remove Programs**.

Or:

- ▶ Select **Start -> Run**. Type "appwiz.cpl" in the Run window (dialog box) and click **OK**.
- ▶ In the Add/Remove Programs window, click **Add/Remove Windows Components**.
- ▶ In the Windows Components page, select the check box beside **Internet Information Services (IIS)**.

- ▶ Click **Details**.
- ▶ Clear all the check boxes, and select the following check boxes:
 - Common Files
 - Documentation
 - FrontPage 2000 Server Extensions
 - Internet Information Services Snap-In
 - Personal Web Manager
 - World Wide Web Server

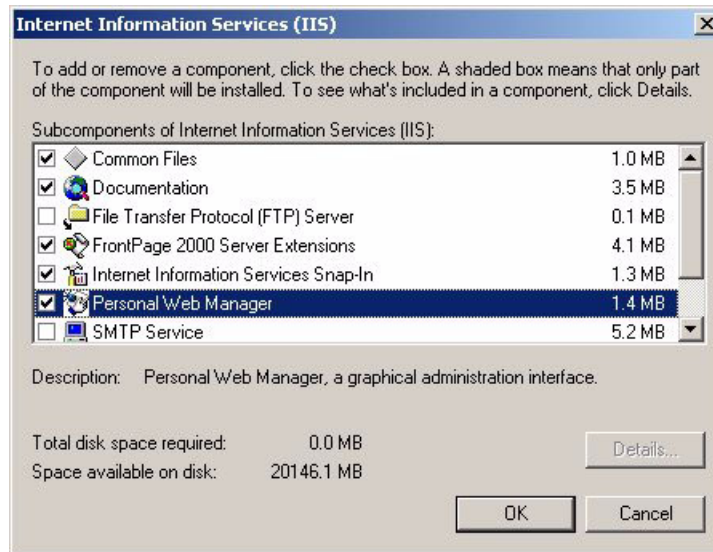


Figure 6-2 Installing Internet Information Services (IIS)

- ▶ Click **OK**.
 - ▶ On the Windows Components page, click **Next**.
- During installation, windows may prompt you for your Windows 2000 CD-ROM. Insert the disk and then click **OK**.

After installation of Internet Information Services (IIS), it is advisable to perform the following steps:

- ▶ For each drive allow administrators full access and deny everyone full access.
 - You can do this by opening My Computer.
 - Right-click your C Drive, select **Sharing**.
 - Select **Do not share this folder**.

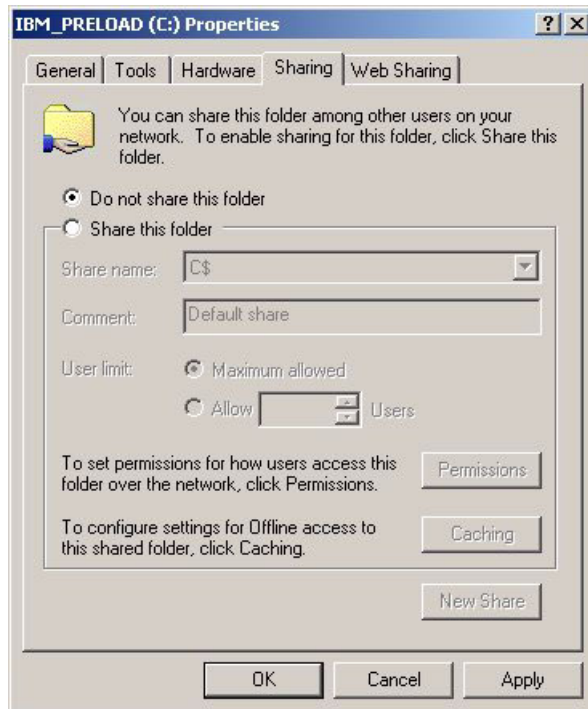


Figure 6-3 Disable sharing

- ▶ Disable the guest account.
- ▶ Obtain the latest service pack from systems support (or windows update) and apply.
- ▶ Reconnect to the network and immediately use windows update to apply all critical windows updates.
- ▶ Run the anti-virus installed on your machine.

6.2.5 Installing Microsoft Visual Studio .NET

Microsoft Visual Studio .NET is a tool for developing applications and is used extensively by the team to develop .NET applications and Web Services for our business case scenario.

- ▶ Insert the Microsoft Visual Studio .NET Professional 2003 Disk 1 CD into the CD drive.
- ▶ In the Microsoft Visual Studio .NET Setup window, select the **Visual Studio .NET Prerequisites** link.

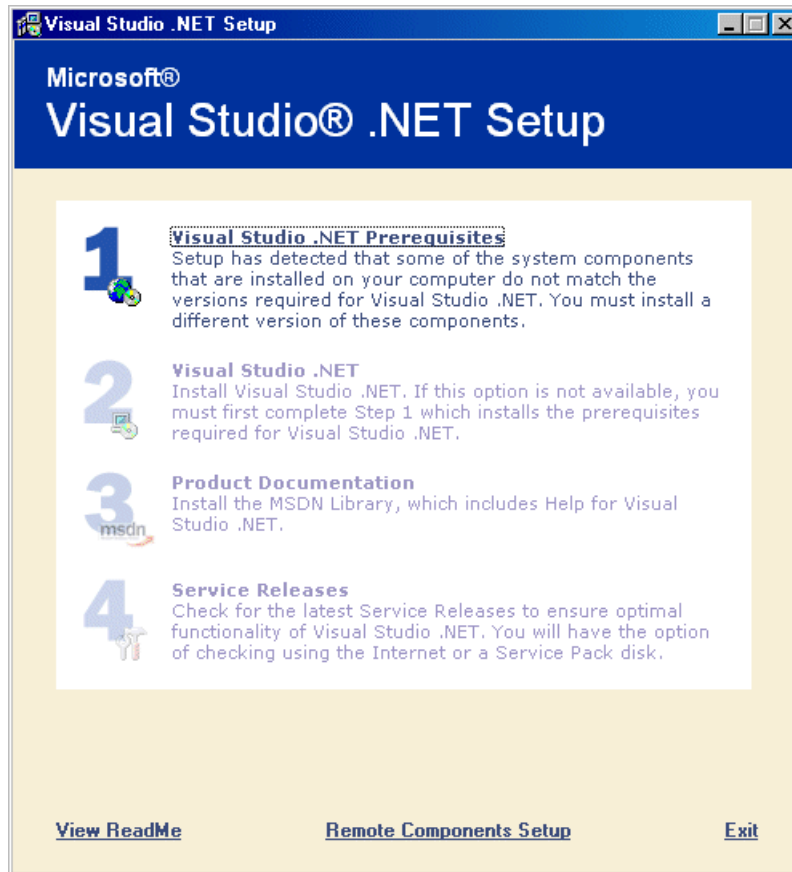


Figure 6-4 Visual Studio .NET Setup

- ▶ You are prompted to insert the Microsoft Visual Studio .NET Prerequisites CD.
- ▶ Browse to the CD Drive and click **OK**.

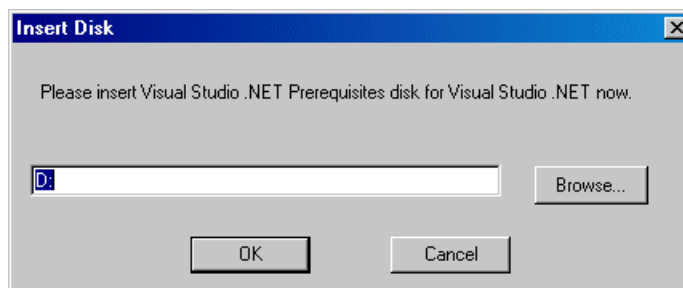


Figure 6-5 Microsoft Visual Studio .NET Prerequisites CD prompt

- In the Microsoft Visual Studio .NET Prerequisites window, read the license agreement and then select **I agree**.
- Click **Continue**.

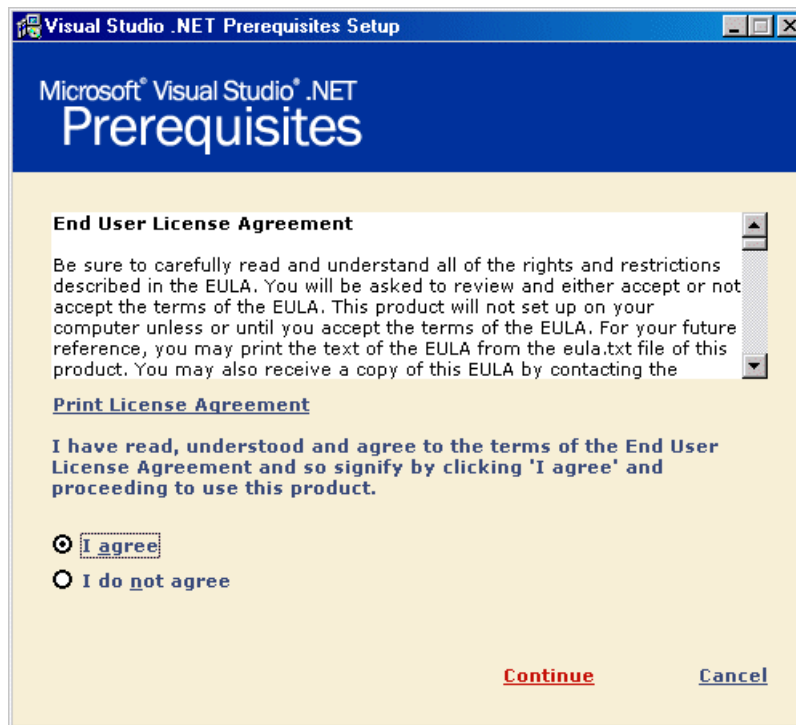


Figure 6-6 License Agreement acceptance

- The Microsoft Visual Studio .NET Prerequisites window is displayed listing all the components to be installed.

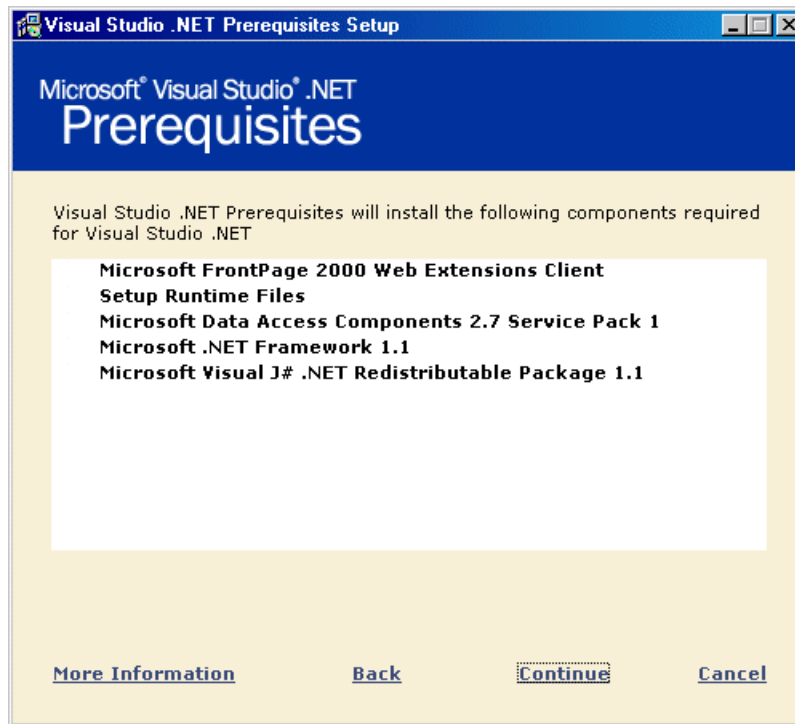


Figure 6-7 Microsoft Visual Studio .NET Prerequisites installation listing

- Click **Continue**.

- In the Microsoft Visual Studio .NET Prerequisites Automatic Log On window, you may optionally specify authentication to automatically log on as shown below:

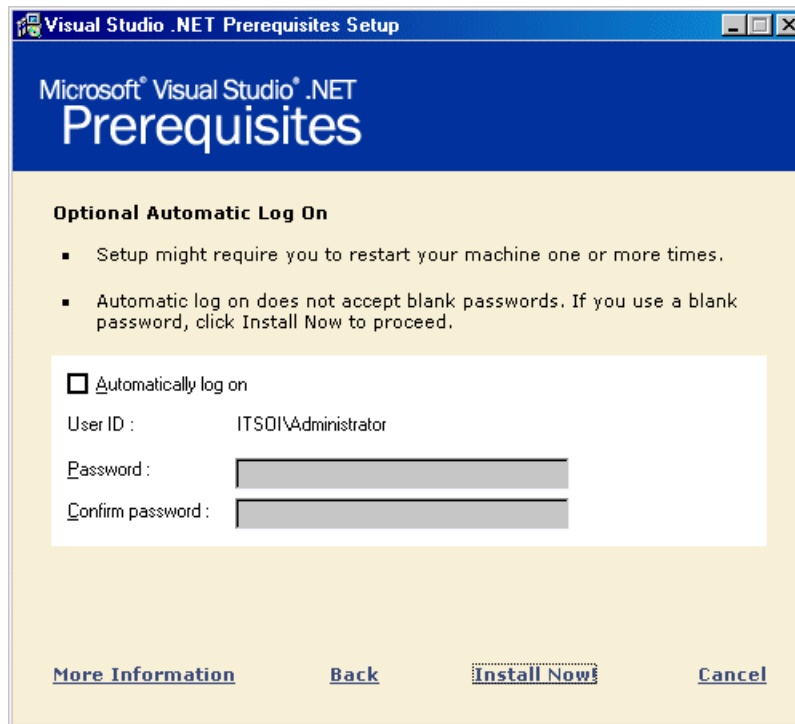


Figure 6-8 Microsoft Visual Studio .NET Prerequisites Automatic Log On

- Microsoft Visual Studio .NET Prerequisites progress bar is displayed showing the progress of the installation as shown below:

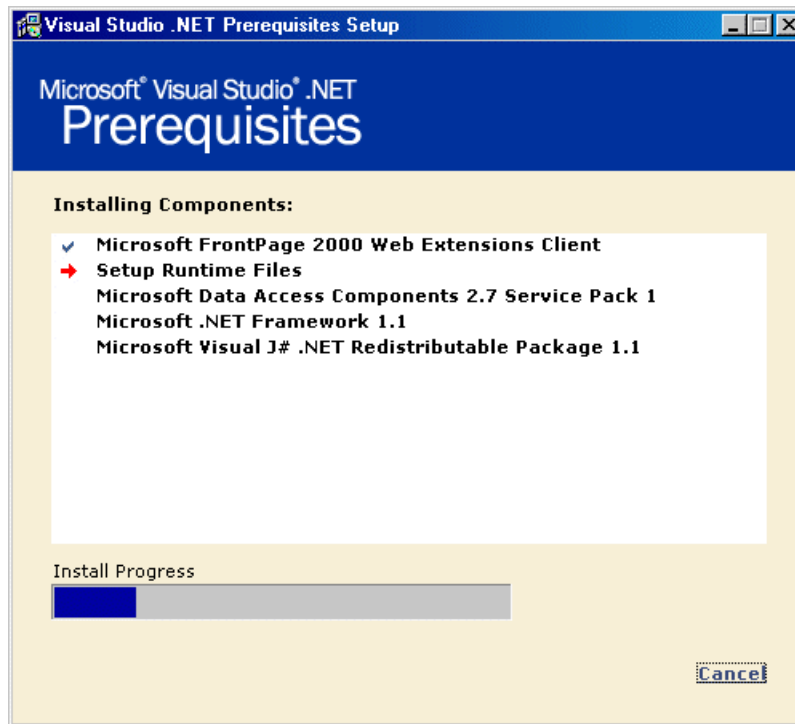


Figure 6-9 Microsoft Visual Studio .NET prerequisites progress bar showing the progress of the installation

- On completion of the installation of prerequisite components, the Microsoft Visual Studio .NET Setup window is again displayed.

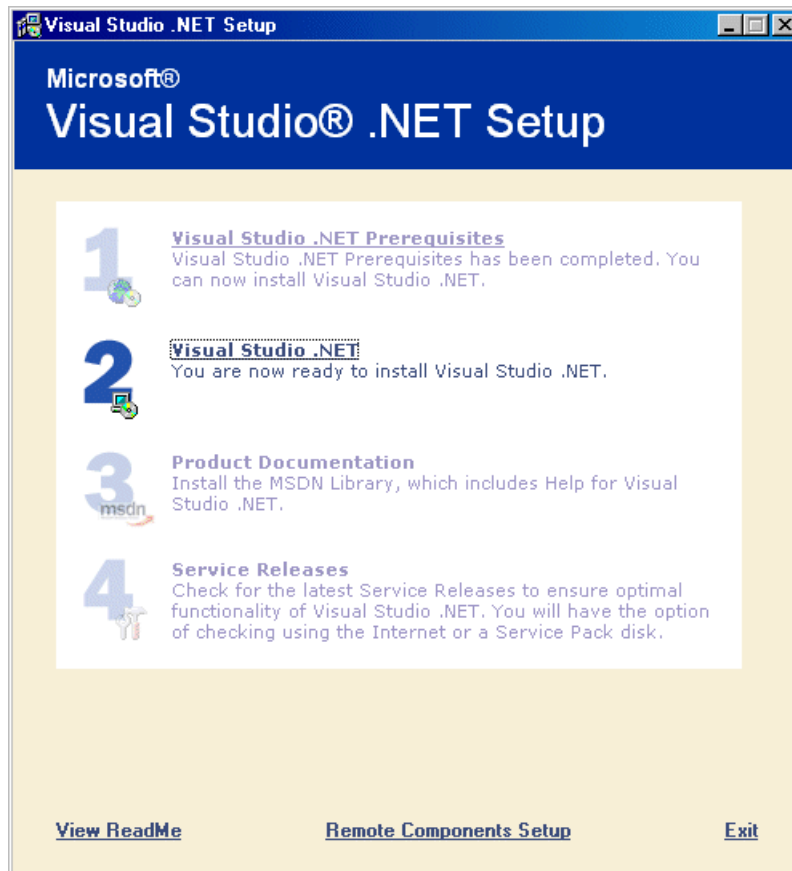


Figure 6-10 Microsoft Visual Studio .NET Setup

- In the Microsoft Visual Studio .NET Setup window, select the Visual Studio.NET link.

- You are prompted to insert the Microsoft Visual Studio .NET Disk1 CD. Browse to the CD Drive and click **OK**.

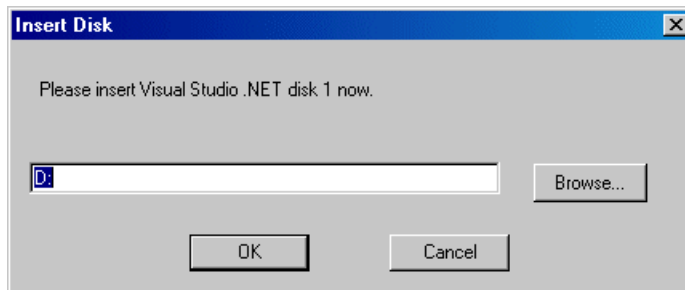


Figure 6-11 Microsoft Visual Studio .NET Disk1 CD prompt

- The installation continues as illustrated below:



Figure 6-12 Installing Microsoft Visual Studio .NET

- ▶ On completion of the installation, the Microsoft Visual Studio .NET Setup window is again displayed.
- ▶ Select the Product Documentation link as shown below:

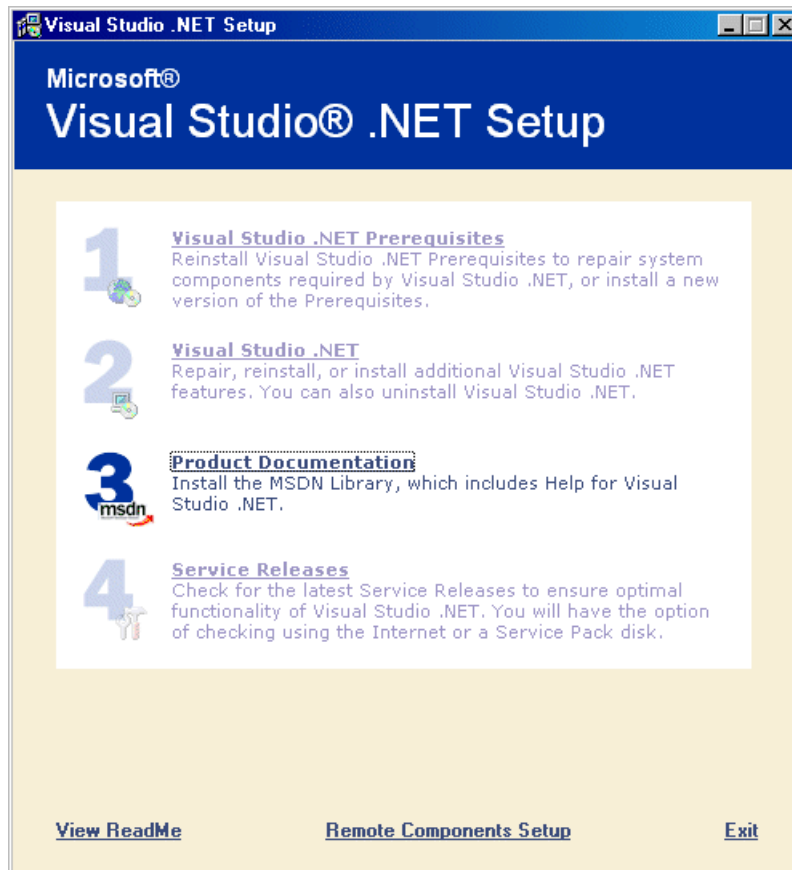


Figure 6-13 Installing the product documentation for Microsoft Visual Studio .NET

- In the MSDN Library for Microsoft Visual Studio .NET setup wizard, select **Next**.

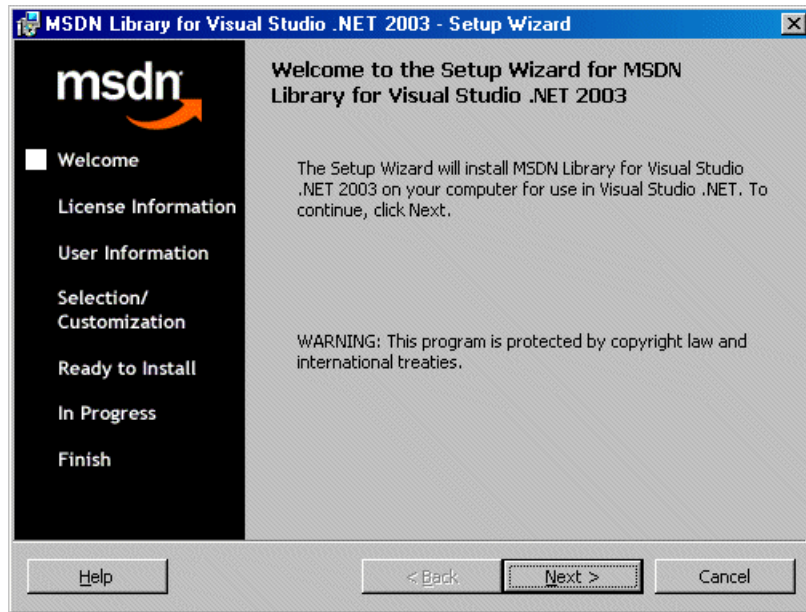


Figure 6-14 MSDN Library for Microsoft Visual Studio .NET setup wizard

Read and accept the license agreement.

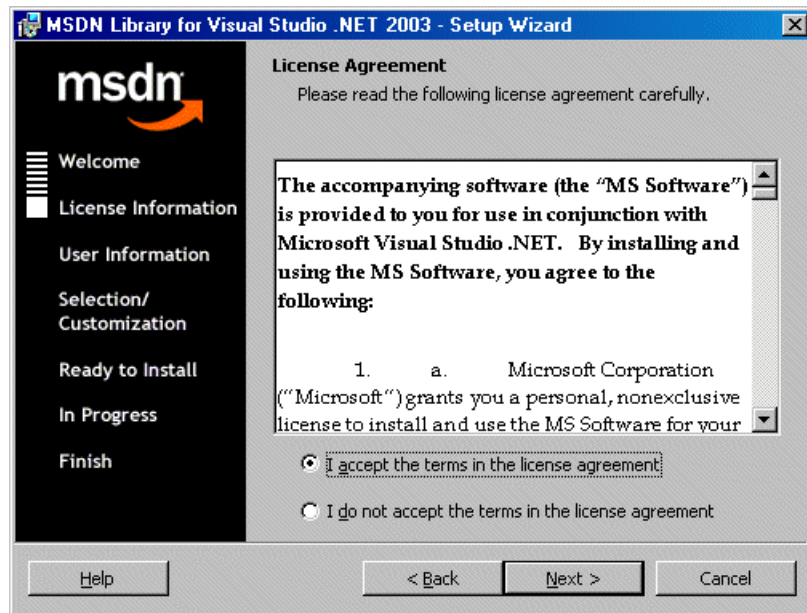


Figure 6-15 MSDN Library for Microsoft Visual Studio .NET License Agreement

- In the MSDN Library for Microsoft Visual Studio .NET customer Information window, enter your username and organization.

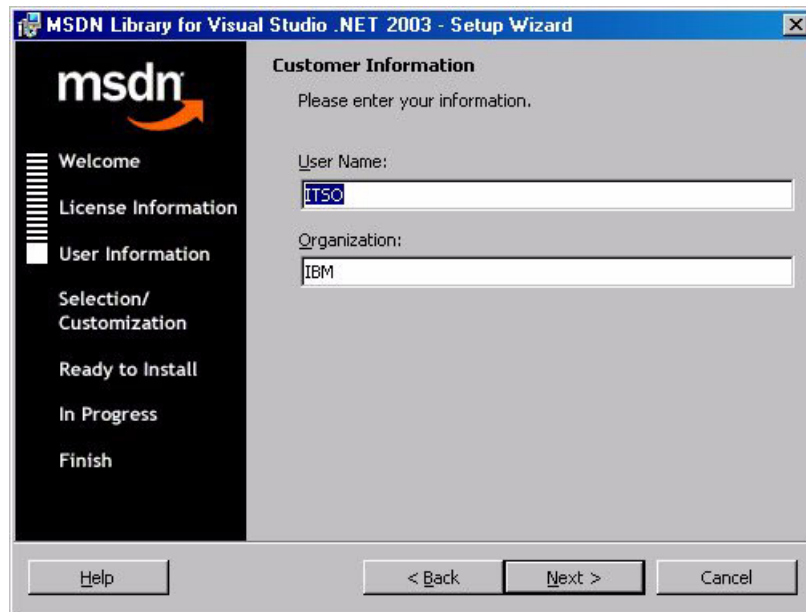


Figure 6-16 MSDN Library for Microsoft Visual Studio .NET customer Information

- In the MSDN Library for Microsoft Visual Studio .NET Selection/Customization window, select Full.

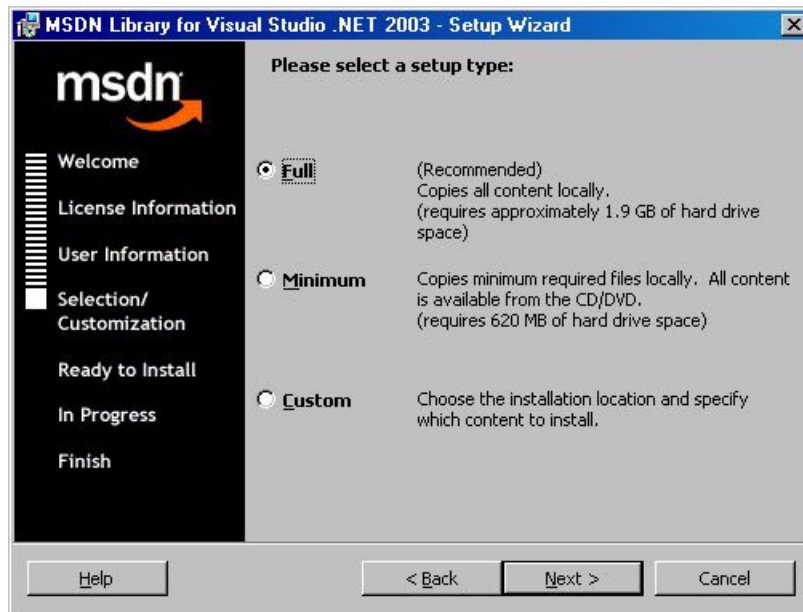


Figure 6-17 MSDN Library for Microsoft Visual Studio .NET Selection/Customization

- In the MSDN Library for Microsoft Visual Studio .NET destination folder window, select **Next** or change the folder directory if necessary.

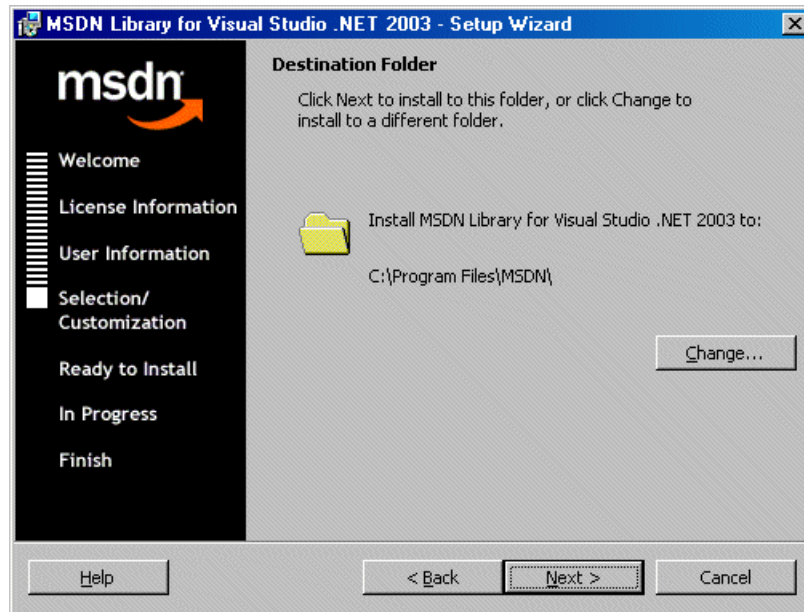


Figure 6-18 MSDN Library for Microsoft Visual Studio .NET destination folder

- In the MSDN Library for Microsoft Visual Studio .NET ready to install window, select **Install**.

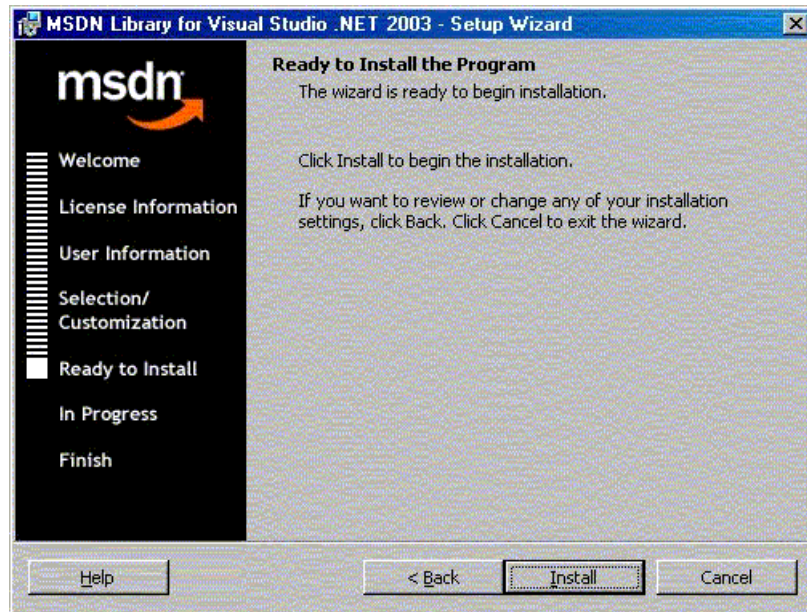


Figure 6-19 MSDN Library for Microsoft Visual Studio .NET ready to install

- Installation begins and an MSDN Library for Microsoft Visual Studio .NET progress bar is shown.

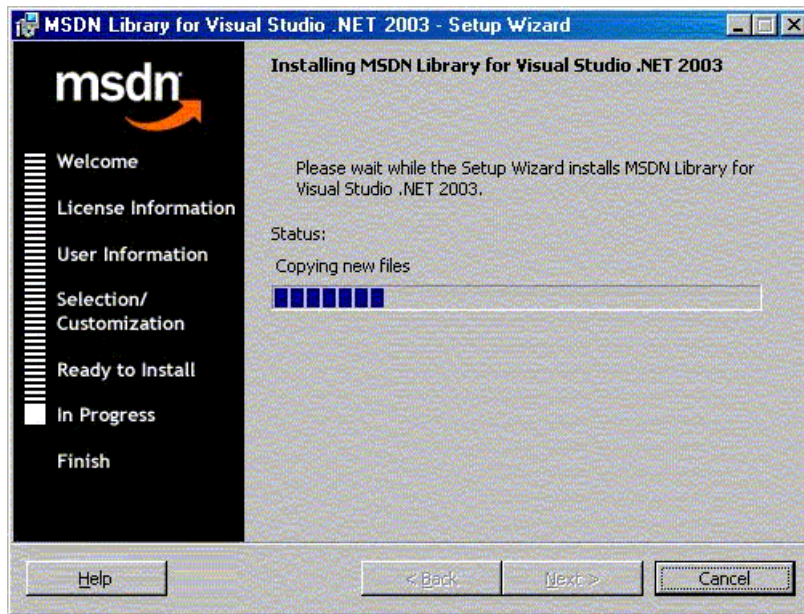


Figure 6-20 MSDN Library for Microsoft Visual Studio .NET in progress

- On the MSDN Library for Microsoft Visual Studio .NET finish window, select **Finish**.

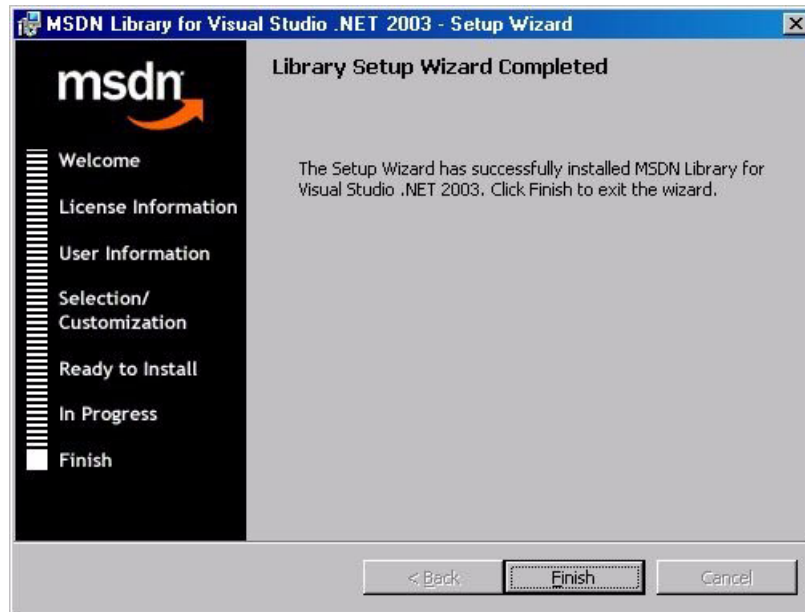


Figure 6-21 MSDN Library for Microsoft Visual Studio .NET finish

- On completion of the installation of product documentation, the Microsoft Visual Studio .NET Setup window is displayed.
- Select the Service Releases link.

6.3 Environment Setup

This section gives an overview of the setup of the core systems used to implement the solution to our business case scenario. It contains information about the software running on each platform in the business case scenario to obtain the configuration in Figure 5-2 on page 83. It also contains information about the WebSphere MQ setup.

6.3.1 Core systems overview

The figure below is a combination of Figure 5-2 on page 83 and Figure 5-3 on page 84 to give an overall description of the environment.

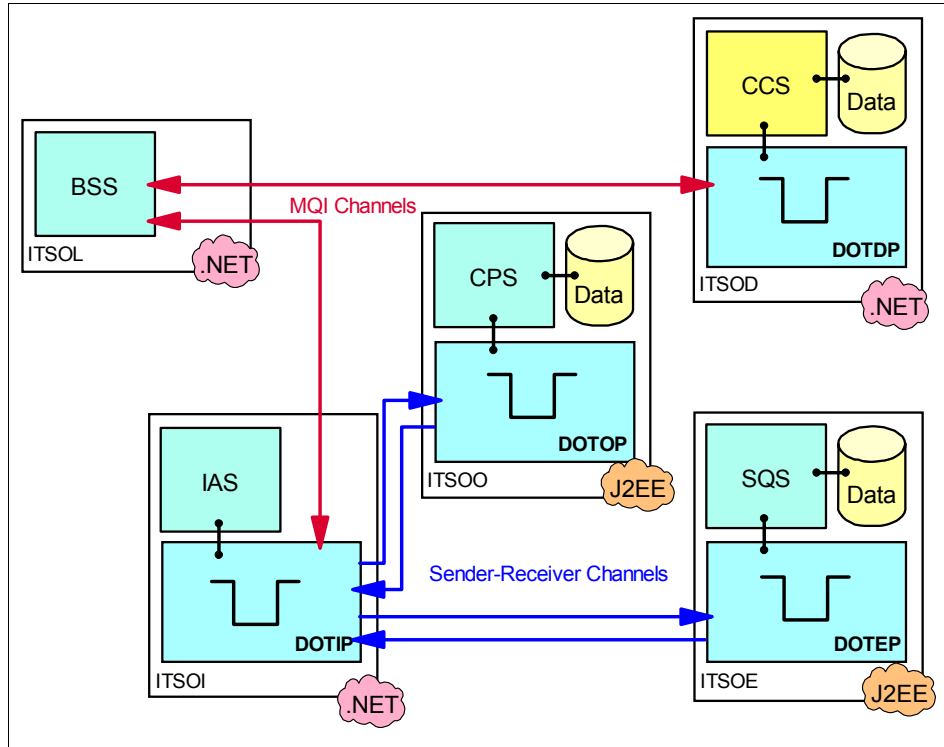


Figure 6-22 Overall representation of the environment

The following table shows the required software on the different servers.

Table 6-1 Core system setup

Software	ITSOL	ITSOD	ITSOI	ITSOO	ITSOE
WebSphere MQ		x	x	x	x
WebSphere MQ classes for Microsoft .NET	x	x	x		
WebSphere MQ Transport for SOAP	x		x		x
Internet Information Services (IIS)	x		(x)		
Microsoft Visual Studio .NET	(x)	(x)	(x)		
.NET Framework	x	x	x		x
Java Virtual Machine	x		x	x	x

Software	ITSOL	ITSOD	ITSOI	ITSOO	ITSOE
Internet Information Services (IIS) needs to be installed on ITSOI but it can be removed later. Microsoft Visual Studio .NET is only needed to develop and debug the applications and is not a prerequisite to run the sample applications.					

WebSphere MQ setup

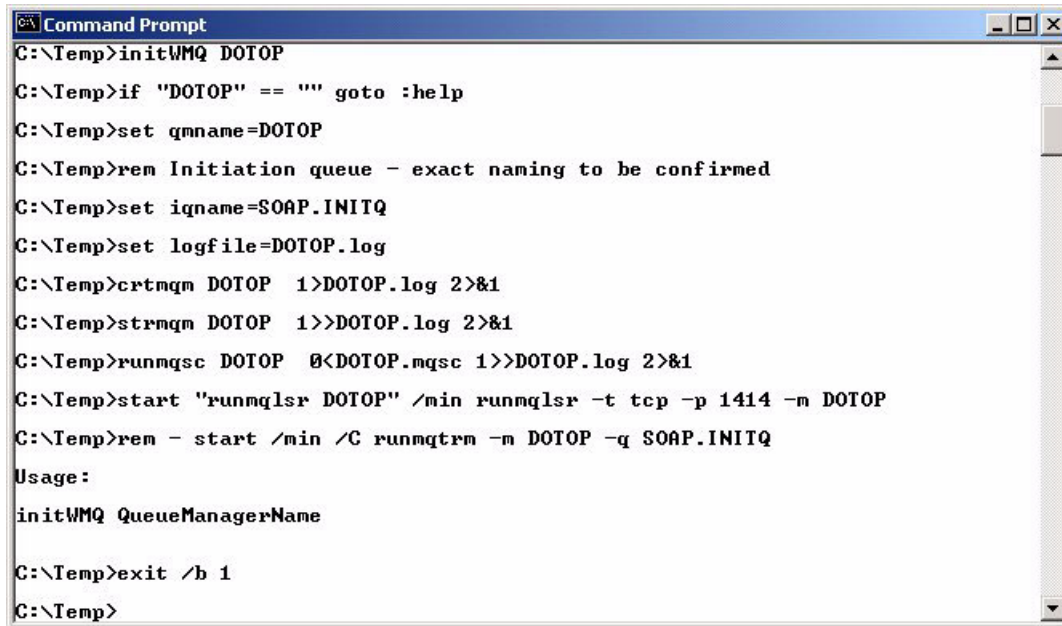
The queue managers and queues for this setup are created using scripts included in the “WebSphere MQ Setup” on page 318.

- ▶ Copy the script from the appendix to a folder on the appropriate machines. For example: Copy the script called DOTOP.mqsc from the appendix to the machine intended to host the DOTOP queue manager.
- ▶ Edit the scripts to change all CONNAME and MCAUSER entries. The CONNAME parameter contains the host name of the machine with the partner queue manager. The MCAUSER parameter contains a local user name, where the user must be a member of the mqm group.
- ▶ Copy initWMQ.bat from appendix to the same folder.
- ▶ Run InitWMQ with the name of the intended queue manager as the parameter in a command prompt.
For example, if you copied DOTOP.mqsc and initWMQ.bat into a folder C:\MQConfig, open a command prompt, change the current directory to C:\MQConfig and type the following:

Example 6-1 WebSphere MQ setup

```
initWMQ DOTOP
```

- The command prompt should look like the one below:



```
C:\Temp>initWMQ DOTOP
C:\Temp>if "DOTOP" == "" goto :help
C:\Temp>set qmname=DOTOP
C:\Temp>rem Initiation queue - exact naming to be confirmed
C:\Temp>set igrname=SOAP.INITQ
C:\Temp>set logfile=DOTOP.log
C:\Temp>crtmqm DOTOP 1>DOTOP.log 2>&1
C:\Temp>strmqm DOTOP 1>>DOTOP.log 2>&1
C:\Temp>runmqsc DOTOP 0<DOTOP.mqsc 1>>DOTOP.log 2>&1
C:\Temp>start "runmqslr DOTOP" /min runmqslr -t tcp -p 1414 -m DOTOP
C:\Temp>rem - start /min /C runmqtrm -m DOTOP -q SOAP.INITQ
Usage:
initWMQ QueueManagerName
C:\Temp>exit /b 1
C:\Temp>
```

Figure 6-23 *initWMQ DOTOP*

- The queue managers, queues and channels and a listener are created.
For the description of what the scripts do, refer to the scripts in Appendix A, "Scripts, source code and test data for YuBank" on page 317, and Chapter 4.
- A log file is created for each setup to check if all scripts run successfully.
- Start all sender channels on the queue managers DOTIP, DOTOP, and DOTEPI. This can be done using the runmqsc command START CHANNEL or using the WebSphere MQ Explorer.



Messaging solution: .NET application to .NET application

This chapter contains the .NET application to .NET application example which is also used in subsequent chapters. The .NET applications use WebSphere MQ classes for Microsoft .NET.

The contents of this chapter are organized as follows:

- ▶ Process overview
 - Scenario overview
- ▶ System context
 - Interface definitions
- ▶ Development
 - Adding the WebSphere MQ reference to the project
 - Bank service application (C#)
 - Credit check application
- ▶ Deployment
 - Deploying BSS

- Deploying CCS
- ▶ Testing
 - How to start BSS
 - How to start CCS
 - Test 1 Pass known data
 - Test 2 Pass unknown user

7.1 Process overview

This process involves part of the account opening use case (use case 1). The business case scenario is a bank performing a credit check for a customer. The bank, after liaison with the customer, sends a WebSphere MQ message to the Credit Check application to establish the customer's credit rating.

The figure below illustrates this part of the account opening use case, with all WebSphere MQ resources involved.

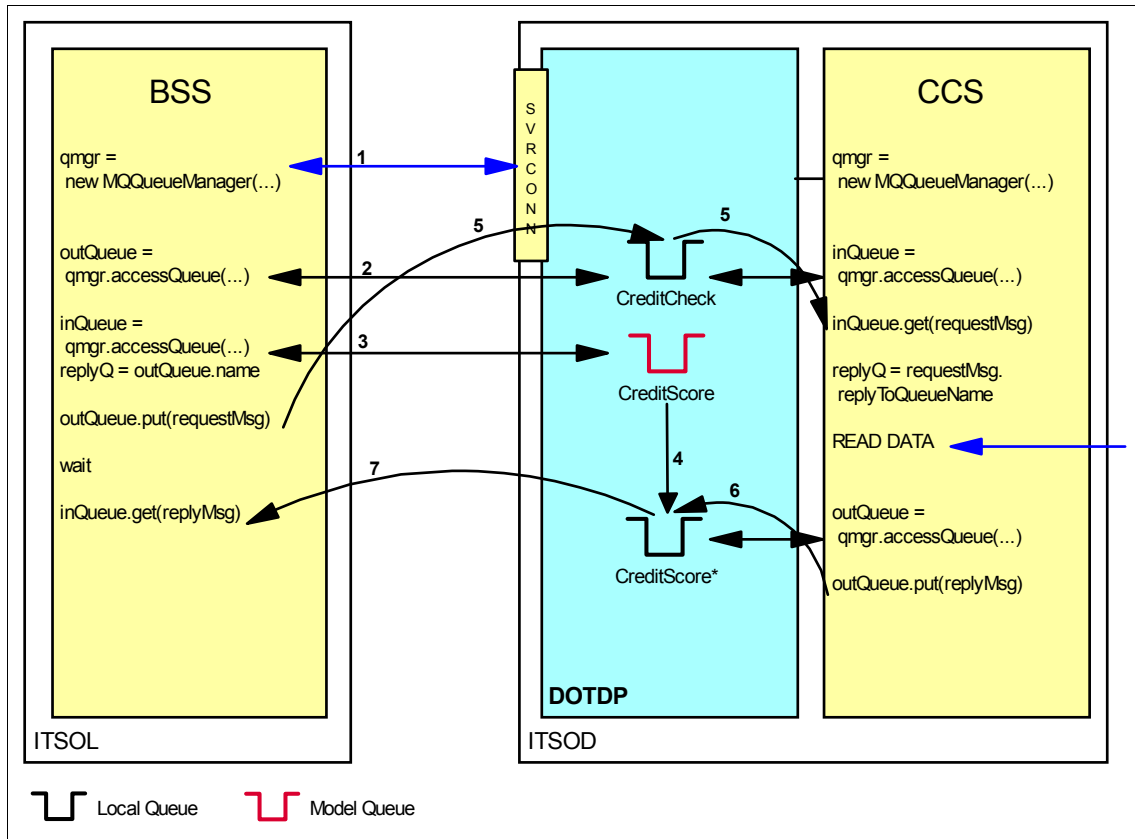


Figure 7-1 Credit Check scenario showing queues involved

The Bank Service System (BSS) application BSS puts a datagram message containing the customer information on the input queue, CreditCheck, of the Credit Check Service (CCS). The CCS gets the message, reads its data and performs a credit check on the customer. CCS puts the credit score on a temporary dynamic queue, CreditScore<unique number> from which BSS extracts it and displays it.

The BSS is an ASP.NET application written in C# and the CCS is a .NET console mode application.

7.1.1 Scenario overview

To open a new investment account the Banking Service System (BSS) gathers information from the customer and then requests a Credit Check using an external system. The Credit Check System (CCS) receives the information using

WebSphere MQ and returns information about the customer's credit rating also using WebSphere MQ but on a dynamic queue extracted from the details contained in the received message. The reply is on a persistent queue because the CCS is typically a paid service. The BSS then subsequently rearranges the information and sends it to the Customer Profile System (CPS). The first scenario deals with the BSS to CCS interchange which both involve .NET applications. The BSS is an ASP.NET applications written in C# and the CCS application is a .NET console mode application. The CCS application is coded in both C# and VB.NET in order to illustrate the WebSphere MQ classes for Microsoft .NET approach in both languages.

7.2 System context

The bank application is an internal banking system and the credit check application is an external bureau that derives an income by providing credit ratings on demand.

7.2.1 Interface definitions

The bank application provides the following customer details:

Table 7-1 Requested customer details

BankID
UniqueID
Name
Address
Date Of Birth
UniqueIDType

In addition, the WebSphere MQ message header is modified to include the additional information:

Table 7-2 WebSphere MQ message header

ReplyToQ
ReplyToQMgr
MessageID
MsgType

The credit check application replies with the information below:

Table 7-3 Credit Check Response

CorrelID
Credit Score
BureauID
Comments
Credit History
Time Stamp
MsgType

Where: CorrelID is assigned the supplied MessageID, and MsgType is assigned MQC.MQMT_REPLY

7.3 Development

To develop the business case scenario both the bank application and the credit check application need to be developed.

7.3.1 Adding the WebSphere MQ reference to the project

Before it is possible to use the WebSphere MQ object's properties, methods, or events, a reference must be created to the object. The technique to add a WebSphere MQ reference is by adding amqmdnet.dll as a reference.

This Dynamic Link Library (DLL) is typically located at:

<C:\Program Files\IBM\WebSphere MQ\bin\amqmdnet.dll>

Techniques for adding a reference are outlined next.

- ▶ In Solution Explorer, click the item requiring the reference.
- ▶ On the Project menu, click **Add Reference**.
- ▶ The Add Reference window appears.

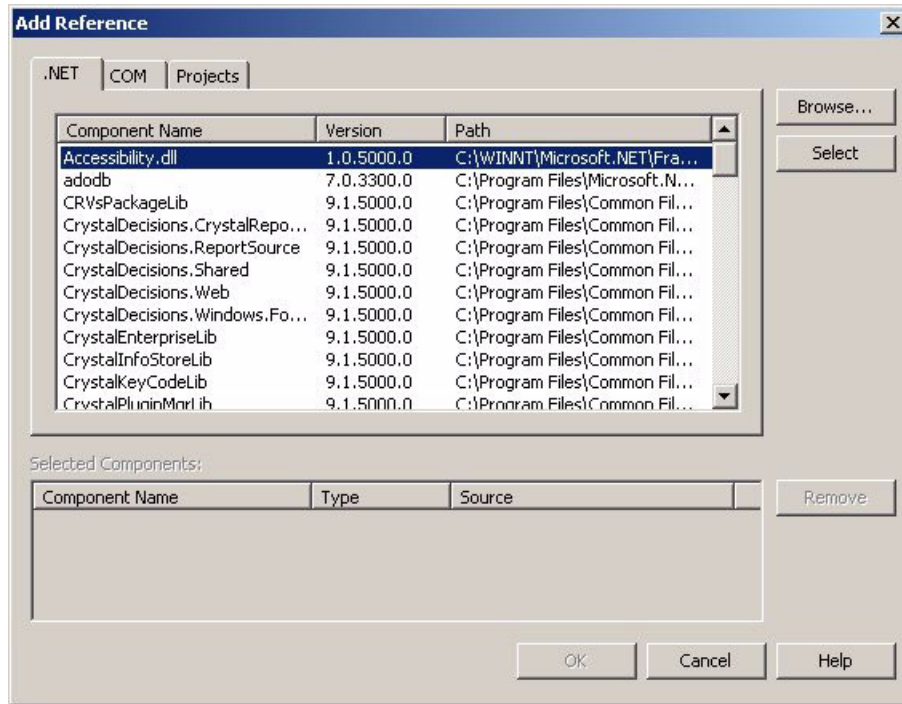


Figure 7-2 Adding a Reference

- ▶ In the Add Reference window, click the tab for the category of reference you are adding. These can be .NET Framework, COM, or Projects, select **.NET**.
- ▶ Click **Browse** and locate the amqmdnet.dll or MQSOAP.dll libraries.
- ▶ Click **Select**, then click **OK**.

As an alternative, you can add a reference by right-clicking the item requiring the reference in Solution Explorer. On the shortcut menu, click Add Reference and the same Add Reference window as the one above appears.

In C# applications the appropriate classes are included as follows:

Example 7-1 how to include WebSphere MQ classes in C#

```
using IBM.WMQ;
```

In VB.NET application the appropriate classes are included as follows:

Example 7-2 how to include WebSphere MQ classes in VB.NET

```
Imports IBM.WMQ
```

7.3.2 Bank service application (C#)

The BSS application gathers the customer details shown in Table 7-1 on page 118 above. Since BSS is a WebSphere MQ client application, it needs appropriate permission to connect to the remote queue manager. The ASP.NET applications run under the user ASPNET. The easiest way to give the ASPNET user permission to connect to the queue manager and use its queues, is to add it to the mqm user group (WebSphere MQ administration group, with access to all WebSphere MQ resources).

Without permission the connect will fail with reason code 2035 (MQRC_NOT_AUTHORIZED).

Prior to sending a WebSphere MQ message to the CCS application the appropriate details are arranged into an eXtensible Markup Language (XML) before sending. The string is an XML string representing a typical WebSphere MQ message using the built-in format MQFMT_STRING and is represented below:

Example 7-3 Sample XML message sent over WebSphere MQ

```
<?xml version="1.0" encoding="utf-16"?>
<BSS BankID="99" UniqueID="1001" Name="Sheppard" Addr="Brisbane"
DOB="19810417" IDType="27" />
```

The following code sample prepares the message in XML format.

Example 7-4 How to assemble the XML message

```
StringWriter sw=new StringWriter();
XmlTextWriter tw = new XmlTextWriter(sw);
// next create the XML stream
tw.WriteStartDocument();
tw.WriteStartElement("BSS");
tw.WriteAttributeString("BankID",textBox1.Text);
tw.WriteAttributeString("UniqueID",textBox2.Text);
tw.WriteAttributeString("Name",textBox3.Text);
tw.WriteAttributeString("Addr",textBox4.Text);
tw.WriteAttributeString("DOB",textBox5.Text);
tw.WriteAttributeString("IDType",textBox8.Text);
tw.WriteEndElement();
tw.WriteEndDocument();
// clean up
tw.Flush();
tw.Close();
```

```
sw.Close();  
string message=sw.ToString();
```

The BSS application connects to a remote queue manager DOTDP through a channel, TO.DOTDP on a host called ITSOD as illustrated in the example below:

Example 7-5 Connecting to remote queue manager

```
try  
{  
    // connect to QueueManager  
    queueManager = new MQQueueManager("DOTDP","TO.DOTDP","ITSOD");  
    System.Console.WriteLine("Connected to QueueManager DOTDP");  
}  
// catch MQExceptions  
catch(MQException ex)  
{  
    System.Console.WriteLine("MQException: compCode: " + ex.CompCode.ToString()  
        + " Reason: " + ex.Reason.ToString());  
}
```

An input queue is created during the WebSphere MQ setup for CCS called CreditCheck. BSS puts datagrams on this queue for CCS to extract. BSS opens the queue as shown below:

Example 7-6 Queue connection

```
//Declare WebSphereMQ variable - queue  
MQQueue queueOut = null  
  
// Open queue out (local queue on CCS)  
queueOut = QM.AccessQueue("CreditCheck", MQC.MQOO_OUTPUT |  
    MQC.MQOO_FAIL_IF_QUIESCING);
```

A model queue, CreditScore, also created during the WebSphere MQ setup, is used to generate a dynamic queue with a unique name. The dynamic queue is used to specify the replyToQueue for the CCS application. This is done by creating a new message object, and setting the CorrelationId field of the message to be received to the value of the MessageId field of the message that was sent to CCS

The following code snippet shows how this is achieved.

Example 7-7 Code to create dynamic queue and define replyToQueue

```
// model queue used to create dynamic queue with unique name - CreditScore*
```

```

queueIn = QM.AccessQueue("CreditScore", MQC.MQOO_INPUT_EXCLUSIVE |
MQC.MQOO_FAIL_IF_QUIESCING, "", "CreditScore*", "");

// save dynamic queue name in a variable in order to define replyToQueue
replyToQueue = queueIn.Name;

// create new message
custDetails = new MQMessage();
// define message properties
custDetails.Persistence = 1;
// the messages' replyToQueue is now unique dynamic queue
custDetails.ReplyToQueueName = replyToQueue;
// set message type to request and format to string
custDetails.MessageType = MQC.MQMT_REQUEST;
custDetails.Format = MQC.MQFMT_STRING;

```

The message containing the customer details is put into the CreditCheck queue as shown in the code snippet below:

Example 7-8 Putting a message on a queue

```

// put message on queueOut
custDetails.WriteString(message);
queueOut.Put(custDetails);

```

CCS extracts the information, performs a credit check on the customer and returns a credit check score to the CreditScore queue. The credit check score is extracted by BSS after a 10 seconds wait. If the reply is not returned within this time period an MQException is thrown with a reason code of MQRC_NO_MSG_AVAILABLE (2033). The following code snippet shows how this is achieved.

Example 7-9 Getting a message from a queue with a 10 seconds wait

```

// create new message
MQMessage creditScore = new MQMessage();
// assign the correlation ID the value of the messageId of the message that
was sent
creditScore.CorrelationId = custDetails.MessageId;
// set getMessageOptions so that there is a 10 seconds wait but stop if
problems occur
gmo = new MQGetMessageOptions();
gmo.Options = MQC.MQGMO_WAIT + MQC.MQGMO_FAIL_IF_QUIESCING;
gmo.WaitInterval = 10000;
// get message
queueIn.Get(creditScore,gmo);

```

After receiving the corresponding reply, the BSS reads the XML string.

Example 7-10 Read message content

```
xmlMessage =  
creditScore.ReadString(creditScore.MessageLength);
```

Finally, the queues are closed, the temporary dynamic queue deleted and the queue manager disconnected.

Example 7-11 Queues closed, temporary dynamic queue deleted, and queue manager disconnected

```
finally  
{  
    // close queueOut  
    queueOut.Close();  
    // close & delete queueIn (dynamic queue)  
    queueIn.CloseOptions = MQC.MQCO_DELETE_PURGE;  
    queueIn.Close();  
    // disconnect QueueManager  
    QM.Disconnect();  
}
```

7.3.3 Credit check application

To ascertain the customer's credit rating the CCS application must access a database.

Credit check database

XML is finding its way into applications beyond those that traditionally utilize markup languages. In particular, XML is becoming popular as a data interchange notation for database oriented applications. As such we are going to interface to a simple flat XML file with the understanding that to extrapolate to a full database is a relatively straightforward exercise.

The following XML file is used by the Credit Check application as its database and is called CreditDatabase.xml.

Example 7-12 Sample CreditDatabase.xml

```
<?xml version="1.0"?>  
<CreditDatabase>  
    <Customer UniqueID="98765" DOB="19780321" IDType="1"  
        score="700" BureauID="1" History="Good rating" Comments="good risk" />  
    <Customer UniqueID="87654" DOB="19830816" IDType="1"  
        score="700" BureauID="2" History="Good rating" Comments="good risk" />
```

```
<Customer UniqueID="76543" DOB="19631109" IDType="1"
score="500" BureauID="1" History="average" Comments="medium risk" />
<Customer UniqueID="54321" DOB="19780321" IDType="1"
score="100" BureauID="3" History="bad debts" Comments="bad risk" />
</CreditDatabase>
```

The Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. In particular an XML DOM is used here as a simple database.

The Credit Check application, after extracting the credit score, sends a WebSphere MQ message back to the bank with the fields outlined in the 7.2.1, “Interface definitions” on page 118.

7.3.4 Credit check application C# snippet

The CCS console mode application is coded in both C# and VB .NET. This section refers to the C# implementation.

How to create the DOM in C#

The following code is taken from the sample.

Example 7-13 Variable declaration

```
XmlDocument DatabaseDOM;
```

The code is:

Example 7-14 Code to create an XML DOM

```
DatabaseDOM = new XmlDocument();
DatabaseDOM.Load(xmlFile);
```

Where, “xmlFile” is the full or relative path for the above xml File. This is surrounded with a try catch block in the sample.

How to access the DatabaseDOM in C#

To get to the attributes:

Example 7-15 Credit Check routine snippet

```
XmlElement root = DatabaseDOM.DocumentElement;
XmlNodeList CustomerList = root.GetElementsByTagName("Customer");
// iterate through all the customers
foreach(XmlNode cust in CustomerList)
```

```

{
    // get the attribute collection for the specific Customer Element
    XmlAttributeCollection attrColl = cust.Attributes;

```

Once we have the attribute for a specific customer we enumerate the attributes again, with `attr.Name` being the attribute name, and `attr.Value` being the attribute value:

```

foreach(XmlAttribute attr in attrColl)
{

```

For the first record in the above sample, it yields:

```

    attr.Name => "UniqueID"
    attr.Value => "98765"}
}

```

In this way it is very easy to check given parameters to see if a match can be found.

How to read a WebSphere MQ message in C#

To read an MQ message it is necessary to first connect to the queue manager and then open the queue:

Example 7-16 How to connect and open a queue in C#

```

//
// Try to connect to the Queue Manager
//
try
{
    mqQMGr = new MQQueueManager( queueManager );
}
catch (MQException mqe)
{
    // Stop if failed
    string Error="MQQueueManager::Connect failed with " + mqe.Message;
    Console.WriteLine(Error);
    return false;
}
//
// Try to open the queue
//
try
{
    mqQueue = mqQMGr.AccessQueue( queueName,
        MQC.MQOO_INPUT_AS_Q_DEF          // open queue for input
        + MQC.MQOO_FAIL_IF_QUIESCING ); // but not if MQM stopping

```

```

    }
    catch (MQException mqe)
    {
        //Stop if failed
        string Error="MQQueueManager::AccessQueue ended with " + mqe.Message;
        Error += mqe.Reason.ToString();
        Console.WriteLine(Error);
        return false;
    }
}

```

The code for reading the message is then as follows:

Example 7-17 Code snippet for reading from a queue in C#

```

MQMessage mqMsg = new MQMessage();
MQGetMessageOptions mqGetMsgOpts = new MQGetMessageOptions();
mqGetMsgOpts.Options = MQC.MQGMO_WAIT + MQC.MQGMO_FAIL_IF QUIESCING;
mqGetMsgOpts.WaitInterval = 15000; // 15 second limit for waiting
try
{
    mqQueue.Get( mqMsg, mqGetMsgOpts );
    if (mqMsg.Format.CompareTo(MQC.MQFMT_STRING) == 0)
    {
        string xmlMessage = mqMsg.ReadString(mqMsg.MessageLength);
        MessageIDIncoming = mqMsg.MessageId;
        QNameIncoming = mqMsg.ReplyToQueueName;
        QMgrIncoming = mqMsg.ReplyToQueueManagerName;
        Console.WriteLine();
        return xmlMessage;
    }
    else
    {
        System.Console.WriteLine( "Non-text message" );
    }
}
catch (MQException mqe)
{
    // report reason, if any
    if ( mqe.Reason == MQC.MQRC_NO_MSG_AVAILABLE )
    {
        // special report for normal end
        if (DotCount++ % 50 == 0)
        {
            Console.WriteLine();
            Console.Write("Idle" );
        }
    }
    else
    {
        Console.Write(".");
    }
}

```

```

    }
}
else
{
    // general report for other reasons
    Console.WriteLine();
    System.Console.WriteLine( "MQQueue::Get ended with " + mqe.Message );
    // treat truncated messages as a failure for this sample
    if ( mqe.Reason == MQC.MQRC_TRUNCATED_MSG_FAILED )
    {
        isContinue = false;
    }
}
}
}

```

Generating WebSphere MQ reply

Once the UniqueID, Date of Birth (DOB) and IDType have been matched the information is retrieved from the DatabaseDOM or, if no match is found, default information is returned.

The information is assembled into an XML message.

Example 7-18 Technique for preparing XML message

```

StringWriter sw=new StringWriter();
XmlTextWriter tw = new XmlTextWriter(sw);
// next create the XML stream
tw.WriteStartDocument();
tw.WriteStartElement("CCS");
tw.WriteAttributeString("Score",CreditScore);
tw.WriteAttributeString("BureauID",BureauID);
tw.WriteAttributeString("Comments",Comments);
tw.WriteAttributeString("History",History);
DateTime dt = DateTime.Now;
tw.WriteAttributeString("Time",dt.ToString());
tw.WriteEndElement();
tw.WriteEndDocument();
// clean up
tw.Flush();
tw.Close();
sw.Close();

string message=sw.ToString();

// put the XML message into the queue

MQMessage mqMsg = new MQMessage();
mqMsg.CorrelationId = MessageIDIncoming;

```

```
mqMsg.MessageType = MQC.MQMT_REPLY;  
mqMsg.WriteString( message );  
mqMsg.Format = MQC.MQFMT_STRING;
```

The CCS Application then returns to the idle state waiting for the next request.

How to Send a WebSphere MQ reply

After creating the XML MQ message the following snippet illustrates how to send the message.

Example 7-19 Sending an MQ message in C#

```
try  
{  
    mqQueue.Put( mqMsg, mqPutMsgOpts );  
}  
catch (MQException mqe)  
{  
    // report the error  
    System.Console.WriteLine( "MQQueue::Put ended with " + mqe.Message );  
    MessageSent = false;  
}  
finally  
{  
    mqQueue.Close();  
}
```

7.3.5 Credit check application VB .NET snippet

The CCS console mode application is coded in both C# and VB.NET. This section refers to the VB .NET implementation.

How to create the DOM in VB .NET

The following code is taken from the sample.

Example 7-20 Variable declaration

```
Dim DatabaseDOM As XmlDocument
```

The code is:

Example 7-21 Code to create an in memory XML DOM

```
Try  
    DatabaseDOM = New XmlDocument  
    DatabaseDOM.Load(xmlFile)
```

```

' Handle the XML exceptions here.
Catch xmlEx As XmlException
    Console.WriteLine("{0}", xmlEx.Message)
' Handle the generic exceptions here.
Catch ex As Exception
    Console.WriteLine("{0}", ex.Message)
Finally
    ' Add code here to finalize.
End Try

```

Where, xmlFile is the full or relative path to the above xml File.

How to access the DatabaseDOM in VB .NET

In this snippet of code the xmlMessage is the string returned from WebSphere MQ.

Example 7-22 How to parse XML in VB

```

Try
    Dim MQMessageDOM As XmlDocument = New XmlDocument
    MQMessageDOM.LoadXml(xmlMessage)
    Dim root As XmlElement = MQMessageDOM.DocumentElement

    Dim attrColl As XmlAttributeCollection = root.Attributes
    For Each attr As XmlAttribute In attrColl
        Console.WriteLine(" attr={0} content={1}", attr.Name, attr.Value)
    Next
Catch e As XmlException
    Console.WriteLine(e.Message)
End Try

```

When using the sample xml database in “Credit check database” on page 124, this sample results in the output shown below:

Example 7-23 Credit check output

```

attr=BankID content=99
attr=UniqueID content=87654
attr=Name content=Sheppard
attr=Addr content=Brisbane
attr=DOB content=19830816
attr=IDType content=1

```

How to read a WebSphere MQ message in VB.NET

To read an MQ message it is necessary to first connect to the queue manager and then open the queue.

Example 7-24 How to connect and open a queue in VB.NET

```
'
' Try to connect to the Queue Manager
'
Try

    mqQMgr = New MQQueueManager(queueManager)

Catch mqe As MulticastNotSupportedException

    ' stop if failed
    Dim Err As String = "MQQueueManager::connect failed with " + mqe.Message
    Console.WriteLine(Err)
    Return False

End Try

' try to open the queuee
Try

    ' open the queue for input but not if MQM stopping
    mqQueue = mqQMgr.AccessQueue(queueName, MQC.MQOO_INPUT_AS_Q_DEF +
    MQC.MQOO_FAIL_IF_QUIESCING)

Catch mqe As MQException

    ' stop if failed
    Dim Err As String = "MQQueueManager::AccessQueue ended with " + mqe.Message
    Err = Err & mqe.Reason.ToString()
    Console.WriteLine(Err)
    Return False

End Try
Return True
```

The code for reading the message is then as follows:

Example 7-25 Code snippet for reading from a queue in VB.NET

```
' MQMessage instance
Dim mqMsg As MQMessage = New MQMessage
' MQGetMessageOptions instance
Dim mqGetMsgOpts As MQGetMessageOptions = New MQGetMessageOptions
mqGetMsgOpts.Options = MQC.MQGMO_WAIT + MQC.MQGMO_FAIL_IF_QUIESCING
mqGetMsgOpts.WaitInterval = 15000 ' 15 second limit for waiting
Try

    mqQueue.Get(mqMsg, mqGetMsgOpts)
    If mqMsg.Format.CompareTo(MQC.MQFMT_STRING) = 0 Then
        Console.WriteLine()
        Dim xmlMessage As String = mqMsg.ReadString(mqMsg.MessageLength)
        MessageIDIncoming = mqMsg.MessageId
    End If
End Try
```

```

        QNameInComing = mqMsg.ReplyToQueueName
        QMgrIncoming = mqMsg.ReplyToQueueManagerName
        Return xmlMessage
    Else
        System.Console.WriteLine("Non-text message")
    End If
Catch mqe As MQException
    ' report reason, if any
    If mqe.Reason = MQC.MQRC_NO_MSG_AVAILABLE Then
        ' Special report for normal end
        If DotCount Mod 50 = 0 Then
            Console.WriteLine()
            Console.Write("Idle")
        Else
            Console.Write(".")
        End If
        DotCount = DotCount + 1
        isContinue = True ' continue until we get a message
    Else
        ' general report for other reasons
        System.Console.WriteLine("MQQueue::Get ended with " + mqe.Message)
        isContinue = False
        ' treat truncated messages as a failure for this example
        If mqe.Reason = MQC.MQRC_TRUNCATED_MSG_FAILED Then
            isContinue = False
        End If
    End If
End Try

```

Generating WebSphere MQ reply

Once the UniqueID, the Date of Birth (DOB) and IDType have been matched the information is retrieved from the DatabaseDOM or, if no match is found, default information is returned.

The information is assembled into an XML message.

Example 7-26 Technique for assembling an XML message in VB.NET

```

' Lets now generate a return XML message
Dim sw As StringWriter = New StringWriter
Dim tw As XmlTextWriter = New XmlTextWriter(sw)
' next create the XML stream
tw.WriteStartDocument()
tw.WriteStartElement("CCS")
tw.WriteAttributeString("Score", CreditScore)
tw.WriteAttributeString("BureauID", BureauID)
tw.WriteAttributeString("Comments", Comments)

```

```

tw.WriteAttributeString("History", History)
Dim dt As DateTime = DateTime.Now
tw.WriteAttributeString("Time", dt.ToString())
tw.WriteEndElement()
tw.WriteEndDocument()
'clean up
tw.Flush()
tw.Close()
sw.Close()

Dim message As String = sw.ToString()
' put the XML message into the queue
Dim mqMsg As MQMessage = New MQMessage
mqMsg.CorrelationId = MessageIDIncoming
mqMsg.MessageType = MQC.MQMT_REPLY
mqMsg.WriteString(message)
mqMsg.Format = MQC.MQFMT_STRING

```

The code for sending the message is then as follows:

Example 7-27 Code snippet for sending to a queue in VB.NET

```

Dim mqPutMsgOpts As MQPutMessageOptions = New MQPutMessageOptions
Try
    mqQueue.Put(mqMsg, mqPutMsgOpts)
Catch mqe As MQException
    ' report the error
    Console.WriteLine("MQQueue::Put ended with " + mqe.Message)
    MessageSent = False
Finally
    mqQueue.Close()
End Try

```

7.4 Deployment

Both the BSS and the CCS applications need to be deployed.

It is assumed that the .NET Framework is installed on the computer using an appropriate .NET Framework installation.

7.4.1 Deploying BSS

The BSS application, being a .NET application, is exposed within a URL namespace and backed using a file system directory located on either a local or remote file share.

By default, an ASP.NET Framework application is automatically configured to use the \bin subdirectory, located immediately under the application root, as its local assembly cache. The \bin directory is also configured to deny any browser access so that a remote client cannot download and steal the code. The following example shows how to layout the BSS (ASP.NET) application, where the \bin directory is immediately under the application root. The application root being:

C:\inetpub\wwwroot\BankingServiceSystem

Example 7-28 Directory structure for BSS

```
AcctOpenFail.aspx
AcctOpenOK.aspx
DisplayError.aspx
RequestAdvice.aspx
DisplayAdvice.aspx
Global.asax
Home.aspx
RequestIDetails.aspx
RequestPDDetails.aspx
Web.config
\bin
BankingServiceSystem.dll
MQSOAP.dll
\img
HomePage.gif
OtherPages.gif
```

Next, configuring Internet Information Services (IIS) is required.

Configuring IIS for BSS Web Application

The ASP.NET Web application requires that both IIS and FrontPage Server Extensions are installed for local development of these project types. The ASP.NET application must also be installed and registered.

Open the Microsoft Management Console (MMC) to the Default Web Site invoking MMC using Start->Run and typing MMC.

In the Default Web Site, click the subdirectory BankingSystemService in order to designate it an application root. Right-click this directory so that it can be made into the application root, and then click Properties.

In the Application Settings section on the Directory tab, click Create.

In the Application name text box, enter the name of the application, and then click OK.

This creates the virtual directory for the BSS Application.

Configuring BSS to use SOAP

To use WebSphere MQ Transport for SOAP in BSS, the condition variable `#define` is uncommented as shown below:

Example 7-29 Configuring BSS to use WebSphere MQ Transport for SOAP

```
#define USE_SOAP
```

This line of code can be found at the first line of `RequestAdvice.aspx`.

Likewise, when WebSphere MQ Transport for SOAP is not required for the Web Services, the condition variable is commented out as shown below:

Example 7-30 Configuring BSS not to use WebSphere MQ Transport for SOAP

```
///

## Configuring BSS to use SSL



The web.config has been altered to define environment variables to cater for the use of SSL as well as the technique of SSL used in BSS.



If SSL is to be used, edit the web.config such that EnableSSL is set to Yes as shown below:



Example 7-31 Configuring BSS to use SSL



---



```
<appSettings>
 .
 .
 <add key="EnableSSL" value="Yes" />
 .
 .
</appSettings>
```



---



If SSL is used and the MQEnvironment technique is preferred, then UseMQEnvironment is set to Yes as shown below:



Example 7-32 Configuring BSS to use SSL and apply the MQEnvironment technique



---



```
<appSettings>
```



Chapter 7. Messaging solution: .NET application to .NET application 135


```

```
.  
.  
<add key="UseMQEnvironment" value="Yes"/>  
</appSettings>
```

If SSL is used and the hashtable technique is preferred, then UseMQEnvironment is set to No as shown below:

Example 7-33 Configuring BSS to use SSL and apply the hashtable technique

```
<appSettings>  
.  
.  
<add key="UseMQEnvironment" value="No"/>  
</appSettings>
```

7.4.2 Deploying CCS

The CCS application is extremely simple to deploy. There are only two files CCS.EXE (or CCSVB.EXE) and the Credit Database CreditDatabase.xml.

The simplest method is to create a directory, say CCS, and copy the two files. The following example illustrates this.

Example 7-34 C# and VB.NET versions

```
Directory of C:\CCS\  
CCS.exe  
CCSVB.exe  
CreditDatabase.xml
```

7.5 Testing

To test the BSS to CCS intercommunication it is necessary to perform two tests. The first test is when the supplied information is known to exist in the database of the CCS and the second test is when the supplied information is known not to be in the database. It is preferable to start CCS running before any testing starts.

7.5.1 How to start BSS

BSS is readily started by launching Internet Explorer and specifying the following URL:

<http://localhost/BankingSystemService/Home.aspx>

7.5.2 How to start CCS

The program, being a console mode program, can be started with the following command:

```
CCS CreditDatabase.xml
```

Or, if using the VB.NET version:

```
CCSVB CreditDatabase.xml
```

Where, the credit check database optionally can be a full path to any location.

7.5.3 Test 1 pass known data

Using the sample CreditDatabase.xml supplied we note the record.

Example 7-35 Chosen record to match

```
<Customer UniqueID="98765" DOB="19780321" IDType="1" core="700"
BureauID="1" History="Good rating" Comments="good risk"
/>
```

As a first test the BSS passes the following information to CCS.

Example 7-36 Information passed for a match

```
UniqueID ="98765"
DOB="19780321"
TypeID="1"
Name=can be anything
Address=can be anything
BankID=can be anything
```

Since the result is known, it is a simple matter to compare the output provided by CCS with the expected reply of:

Example 7-37 returned data reflecting the data contained in the database

```
Credit Score = 700
BureauID=1
Comments=good risk
Credit History=Good rating
Time Stamp=Time the information was provided.
```

The test was performed successfully with both the C# and the VB.NET versions.

7.5.4 Test 2 pass unknown user

The result is SUCCESS.

Using the sample CreditDatabase.xml supplied we identify an unknown customer.

Example 7-38 Information passed expecting a mismatch

UniqueID ="99999"
DOB="19771118"
TypeID="1"
Name=can be anything
Address=can be anything
BankID=can be anything

We expect to get back default data in reply.

Example 7-39 returned data reflecting default data

Credit Score = 500
BureauID=1
Comments=Not Known
Credit History=No records
Time Stamp=Time the information was provided.

The test was performed successfully both the C# and the VB.NET versions.



Messaging solution: .NET application to J2EE application

This chapter contains the .NET application to J2EE application example. The .NET application uses the WebSphere MQ classes for Microsoft .NET and J2EE application uses the Java Message Service (JMS) API.

It only describes part of the business scenario use cases, only where .NET applications send messages to a J2EE application is covered here. Chapter 7, “Messaging solution: .NET application to .NET application” on page 115 is a prerequisite for a basic understanding of the WebSphere MQ classes for Microsoft .NET.

The contents of this chapter are organized as follows:

- ▶ Process overview
 - Account opening
 - Investment advisory
- ▶ System context
 - Bank service application
 - Investment advisory application
 - Customer profile application

- Database
- JMS administered objects
- ▶ Development
 - Bank service application
 - Investment advisory application
 - Customer profile application
- ▶ Deployment
 - Deploying BSS
 - Deploying CPS
- ▶ Testing
- ▶ Alternative solutions
 - WebSphere MQ classes for Microsoft .NET and WebSphere MQ classes for Java
 - Bridge between WebSphere MQ and Microsoft Message Queuing (MSMQ)

8.1 Process overview

The Customer Profile System (CPS) is implemented as a J2EE application. It is involved in the two use cases described in the business case scenario. In the account opening use case the Bank Service System (BSS) sends a customer profile to the CPS to store it in the database. In the investment advisory use case the Investment Advisory System (IAS) sends a request for a specific customer profile to the CPS. The CPS reads the data out of the database and sends it back to the IAS.

8.1.1 Account opening

Figure 8-1 shows part of the account opening use case, with all WebSphere MQ resources and applications involved.

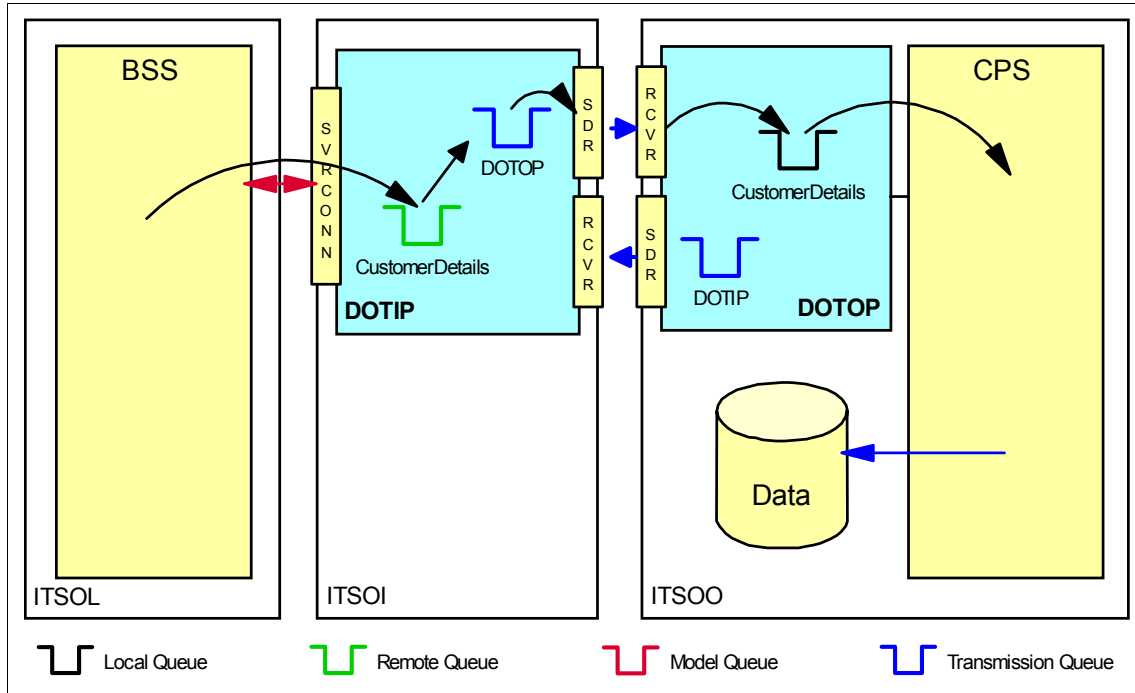


Figure 8-1 BSS to CPS communication

To open a new customer account, the Banking Service System (BSS) requests credit information for the customer from an external credit bureau. This is described in Chapter 7, "Messaging solution: .NET application to .NET application" on page 115.

After the credit check, the BSS puts a datagram message containing all customer information on the input queue of the Customer Profile System (CPS). The CPS gets the message, reads its data and stores the data in a local database.

The BSS is an ASP.NET application written in C# and the CPS is a J2EE console mode application.

8.1.2 Investment advisory

Figure 8-2 shows part of the investment advisory use case, with all WebSphere MQ resources and applications involved.

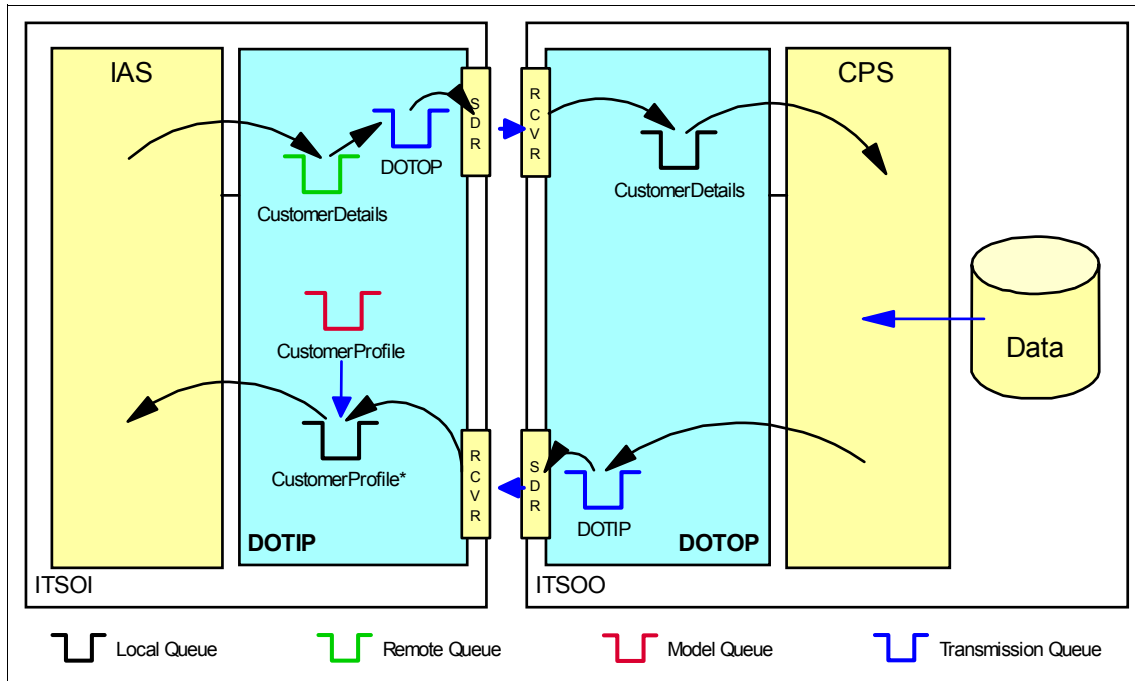


Figure 8-2 IAS to CPS communication

To create an investment advice, the Investment Advisory System (IAS) Web Service requests a customer profile from the Customer Profile System (CPS) whenever it is invoked.

To receive the latest customer profile the IAS sends a request message to the input queue of the CPS. The CPS gets this request, reads the relevant data out of the local database and sends a reply containing this data back to the reply queue.

The IAS is a .NET Web Service written in C# and the CPS is a J2EE console mode application.

8.2 System context

The BSS and the IAS use different data structures to communicate with the CPS system.

8.2.1 Bank service application

The BSS sends a WebSphere MQ persistent datagram message with the following fields:

Table 8-1 Account opening datagram

MsgObjective
AccountNum
CustId
CustName
IDType
CreditScore
BureauID
Timestamp
InitInvest
RiskLevel
InvestPeriod
RetExp
FamilyIncome
ExistAsset
CurDebt
DOB
Addr

The MsgObjective field is used by the CPS application to check whether this message should be used to open a new account or to send back existing data. The BSS uses the value “OpenAccount” to indicate that CPS has to write the content of the message into the database.

The WebSphere MQ built-in format MQFMT_STRING is used, so that it can be handled as a TextMessage by the J2EE application. The string is an XML string, shown by Example 8-1.

Example 8-1 OpenAccount message

```
<?xml version="1.0" encoding="utf-16"?>
```

```
<Message MsgObjective="OpenAccount">
  <Account AccountNum="317" CustID="98765" CustName="Ope Soyannwo" IDType=""
    CreditScore="700" BureauID="1" Timestamp="09/07/2003 15:14:43"
    InitInvest="500" RiskLevel="1" InvestPeriod="27" RetExp="2"
    FamilyIncome="40000" ExistAsset="120000" CurDebt="9000" DOB="19780321"
    Addr="ITSO, IBM Hursley, Hursley, United Kingdom" />
</Message>
```

8.2.2 Investment advisory application

The IAS sends a non-persistent request message to the CPS containing the following fields:

Table 8-2 Customer profile request

MsgObjective
AccountNum

The MsgObjective field is used by the CPS application to check whether this message should be used to open a new account or to send back existing data. The IAS uses the value “QueryCustomerProfile” to indicate that CPS has to return account information to the IAS.

Just as for BSS the WebSphere MQ’s built-in format MQFMT_STRING is used. The string is an XML string, shown by Example 8-2.

Example 8-2 QueryCustomerProfile message

```
<?xml version="1.0" encoding="utf-16"?>
<Message MsgObjective='QueryCustomerProfile'>
  <Account AccountNum='222' />
</Message>
```

After sending the request the IAS waits for a reply from CPS synchronously.

8.2.3 Customer profile application

The CPS is a Java application that listens on its input queue for messages. Its behavior is dependent on the content of the XML message it receives. When it gets a message with a value of “OpenAccount” in the MsgObjective field, it stores the data of the message in the local database “CustomerProfile”. If the messages MsgObjective field contains the value “QueryCustomerProfile” it reads a customer profile from the database and sends it back to the requesting application in a reply message.

The reply message is non-persistent and contains the following fields:

Table 8-3 Customer profile reply

RiskLevel
InvestPeriod
RetExp
FamilyIncome
ExistAsset
CurDebt

Correlation between the customer query message and the customer profile reply use the WebSphere MQ message descriptor fields `messageId` and `correlationId`.

8.2.4 Database

Our database is an XML file on the local directory `C:\CPS\XML`. The file is called `CustomerProfile.xml`.

The format of the XML file is shown in the Example 8-19 with two existing account entries.

Example 8-3 Database file

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomerProfileDatabase>
  <Account AccountNum="127" CustID="98765" CustName="Ope Soyannwo" IDType=""
    CreditScore="700" BureauID="1" Timestamp="09/07/2003 17:46:12"
    InitInvest="2000" RiskLevel="2" InvestPeriod="36" RetExp="5"
    FamilyIncom="45000" ExistAsset="15500" CurDebt="3000" DOB="19780321"
    Addr="ITSO, IBM Hursley, Hursley, United Kingdom" />
  <Account AccountNum="101" CustID="98945" CustName="Michael Hamann"
    IDType="" CreditScore="600" BureauID="1" Timestamp="09/07/2003 17:48:47"
    InitInvest="5000" RiskLevel="3" InvestPeriod="36" RetExp="9"
    FamilyIncom="60000" ExistAsset="24000" CurDebt="3000" DOB="19780321"
    Addr="ITSO, IBM Herrenberg, Herrenberg, Germany" />
  .
  .
  .
</CustomerProfileDatabase>
```

8.2.5 JMS administered objects

Within Java Message Service (JMS) applications all provider-specific information is encapsulated within administered objects, so only interfaces in `javax.jms` are referenced. These administered objects are stored in a Java Naming and Directory Interface (JNDI) namespace using a provider-supplied administration tool.

The CPS application uses a `QueueConnectionFactory` to create the `Connection`. This `QueueConnectionFactory` is the first administered object used. It is defined in the following way:

Example 8-4 QueueConnectionFactory definition

```
DEFINE QCF(ServerQCF) QMANAGER(DOTOP) POLLINGINT(1000)
```

This `QueueConnectionFactory` called `ServerQCF` connects to the local queue manager `DOTOP` using the bindings mode. An asynchronous listener is used within the CPS, so the WebSphere MQ implementation of JMS has to poll for messages on a specific queue. A polling interval of one second is chosen, the default is five seconds. A longer polling interval is acceptable for the first use case, in which the CPS stores the customer profile in the database, but a short polling interval is needed for the second use case, where the IAS waits for a reply from the CPS synchronously.

One input queue for the CPS called `CustomerDetails` is created. This queue is used by the BSS to put datagrams in, and by the IAS to send the requests. This WebSphere MQ queue name has to be encapsulated in a queue object, which is our second administered object. This is defined in the following way:

Example 8-5 Queue definition

```
DEFINE Q(CustomerDetails) QUEUE(CustomerDetails) QMANAGER(DOTOP) TARGCLIENT(MQ) PERSISTENCE(APP)
```

This queue object points to the WebSphere MQ queue `CustomerDetails` on the queue manager `DOTOP`. Our JMS clients exchanges messages with WebSphere MQ applications over this queue, so the `TARGCLIENT` property of this queue, has to be set to `MQ`.

There is no need to define an output queue as an administered object for the CPS, because the IAS sends the reply queue name and the reply queue manager name in the request message. These names in the `ReplyToQueueName` and `ReplyToQueueManagerName` fields of the message descriptor are mapped to a `Destination` object in the `JMSReplyTo` field of the `JMSMessage` object.

8.3 Development

The bank service application and the investment advisory Web Service are written in C#. The customer profile application is written in Java. In the following sections only the messaging logic is described. For the complete code refer to the “Use case 1” on page 321.

8.3.1 Bank service application

The BSS application is written in C#. After communicating with the CCS it connects to the queue manager DOTIP using the client connection, where TO.DOTIP is the name of the server connection channel and ITS01 is the host name.

Example 8-6 Connect to queue manager

```
MQQueueManager QM;  
QM = new MQQueueManager("DOTIP","TO.DOTIP","ITS01");
```

Then it opens the queue CustomerDetails on DOTIP for output, which is a remote queue pointing to the queue CustomerDetails on DOTOP, on which the CPS application is listening on.

Example 8-7 Open output queue

```
MQQueue queueOut = null;  
queueOut = QM.AccessQueue("CustomerDetails", MQC.MQOO_OUTPUT |  
    MQC.MQOO_FAIL_IF_QUIESCING);
```

The MQMessage constructor creates a message object with default values. A persistent message with a built-in string format is required. The message string contains the XML data.

Example 8-8 Set up message

```
MQMessage custDetails = null;  
custDetails = new MQMessage();  
custDetails.Persistence = MQC.MQPER_PERSISTENT;  
custDetails.Format = MQC.MQFMT_STRING;  
custDetails.WriteString(message);
```

After setting up the message object, the BSS puts the message on the output queue.

Example 8-9 Put message

```
queueOut.Put(custDetails);
```

Finally the BSS closes the output queue and disconnects from the queue manager.

Example 8-10 Close and disconnect

```
queueOut.Close();  
QM.Disconnect();
```

For more details on the BSS application refer to Chapter 7, “Messaging solution: .NET application to .NET application” on page 115, the full code is in the “Use case 1” on page 321.

8.3.2 Investment advisory application

The IAS Web Service is written in C#. Whenever invoked by the BSS it connects to its local queue manager DOTIP using a bindings connection.

Example 8-11 Connect to queue manager

```
private MQQueueManager _QM;  
private string _QMConnectionString;  
_QMConnectionString="DOTIP";  
_QM = new MQQueueManager(_QMConnectionString);
```

Then it opens the queue CustomerDetails for output, which is a remote queue pointing to the queue CustomerDetails on DOTOP, on which the CPS application is listening on.

Example 8-12 Open output queue

```
private MQQueue _QueueOut;  
private string _strOutputQueue;  
_strOutputQueue="CustomerDetails";  
_QueueOut = _QM.AccessQueue(_strOutputQueue, MQC.MQOO_OUTPUT |  
    MQC.MQOO_FAIL_IF_QUIESCING);
```

Next the IAS opens the model queue CustomerProfile, a temporary dynamic queue is generated with a unique name starting with the string CustomerProfile. The generated queue name is stored in the variable replyToQueue.

Example 8-13 Open model queue for input

```
private string _strInputQueue;  
_strInputQueue="CustomerProfile";
```

```
MQQueue _QueueIn = _QM.AccessQueue(_strInputQueue, MQC.MQOO_INPUT_EXCLUSIVE |
    MQC.MQOO_FAIL_IF_QUIESCING, "", _strInputQueue+"*", "");
string replyToQueue = _QueueIn.Name;
```

The MQMessage constructor creates a message object with default values. A non-persistent request message and the built-in string format is required. The unique name of the dynamic queue is copied to the ReplyToQueueName field of the message. The InputMessage string contains the XML data.

Example 8-14 Set up output message

```
MQMessage _OutputMessage = new MQMessage();
_OutputMessage.Persistence = MQC.MQPER_NOT_PERSISTENT;
_OutputMessage.Format = MQC.MQFMT_STRING;
_OutputMessage.MessageType = MQC.MQMT_REQUEST;
_OutputMessage.ReplyToQueueName = replyToQueue;
_OutputMessage.WriteString(InputMessage);
```

After setting up the message object, the IAS puts the message on the output queue.

Example 8-15 Put message

```
_QueueOut.Put(_OutputMessage);
```

Then it creates a new message object, and sets the CorrelationId field of the message to receive to the value of the MessageId field of the message it has just sent to the CPS.

Example 8-16 Set up input message

```
MQMessage _InputMessage = new MQMessage();
_InputMessage.CorrelationId = _OutputMessage.MessageId;
```

The IAS uses the message object with a specified CorrelationId in the get method with the wait option and a wait interval. So it waits for the specific reply on the dynamic queue for at most ten seconds. If the reply is not returned within this time period an MQException is thrown with a reason code of MQRC_NO_MSG_AVAILABLE (2033).

Example 8-17 Wait for input message

```
private int _wait;
_wait=10000;
MQGetMessageOptions _GetMsgOpt = new MQGetMessageOptions();
_GetMsgOpt.Options = MQC.MQGMO_WAIT + MQC.MQGMO_FAIL_IF_QUIESCING;
_GetMsgOpt.WaitInterval = _wait;
```

```
_QueueIn.Get(_InputMessage,_GetMsgOpt);
```

After receiving the corresponding reply, the IAS reads the XML string.

Example 8-18 Read message content

```
_InputMessage.ReadString(_InputMessage.MessageLength);
```

Finally it closes the dynamic queue. Because it is a temporary dynamic queue, it is deleted, without setting a close option. Then the IAS disconnects from the queue manager.

Example 8-19 Close the queue and disconnect

```
_QueueOut.Close();  
_QM.Disconnect();
```

For more details on the IAS application, refer to:

- ▶ Chapter 9, “Messaging solution: .NET client to .NET Web Services using WebSphere MQ SOAP transport” on page 163
- ▶ Chapter 10, “Messaging solution: .NET client to J2EE Web Services using WebSphere MQ SOAP transport” on page 207

The full code is in “Use case 2” on page 322.

8.3.3 Customer profile application

The CPS application is written in Java. It uses the JMS interface. The CPS is a long running application that waits for incoming messages. The messaging code of the CPS can be found in the CustomerProfile class with its main method and the MsgHandler class, which is the message listener.

When the application gets started, it first looks up the QueueConnectionFactory and queue objects in the JNDI namespace. The file system is used as the namespace.

Example 8-20 JNDI configuration and look up

```
String url = "file:/C:/CPS/JNDI";  
String icf = "com.sun.jndi.fscontext.RefFSContextFactory";  
InitialContext ctx;
```

```
String queueName = "CustomerDetails";  
String qcfName = "ServerQCF";
```

```
QueueConnectionFactory qcf;
```

```
Queue queue;

//Initial context for JNDI lookup
Hashtable environment = new Hashtable();
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
environment.put(Context.PROVIDER_URL, url);
ctx = new InitialContext(environment);

//look up administered object.
qcf = (QueueConnectionFactory)ctx.lookup(qcfName);
queue = (Queue)ctx.lookup(queueName);
```

Then it uses the QueueConnectionFactory to create a QueueConnection. In WebSphere MQ terms, the connection holds the parameters that control how to connect to the queue manager, these informations are provided by the factory. Connections in JMS are thread safe.

Example 8-21 Create connection

```
QueueConnection qConnection;
qConnection = qcf.createQueueConnection();
```

With this connection the CPS creates a non-transacted QueueSession that automatically acknowledges every incoming messages. In WebSphere MQ terms the session contains the HCONN and defines the transactional scope. Sessions in JMS are not thread safe.

Example 8-22 Create session

```
QueueSession qSession;
qSession = qConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
```

The session is used to create a queue receiver for the input queue. In WebSphere MQ terms the receiver contains the HOBJ that describes the particular queue for reading. A message listener is created and registered with the queue receiver. The message listener runs in its own thread and handles all incoming messages. Our message listener constructor takes the connection as an argument, to create its own session.

Example 8-23 Create receiver and set listener

```
QueueReceiver qReceiver;
qReceiver = qSession.createReceiver(queue);
MsgHandler handler = new MsgHandler(qConnection);
qReceiver.setMessageListener(handler);
```

After creation the connection is in stopped mode. To receive messages within the connection it has to be started.

Example 8-24 Start connection

```
qConnection.start();
```

The CPS runs until the user ends it. When it stops it closes the session and the connection.

Example 8-25 Close session and connection

```
qSession.close();  
qConnection.close();
```

The `MsgHandler` class is the message listener that handles all incoming messages on the receiver queue. It has to implement the `MessageListener` interface.

Example 8-26 MsgHandler class

```
public class MsgHandler implements MessageListener
```

The following objects are defined in this class. The session, sender, and message objects are only needed when a reply has to be sent.

Example 8-27 Define objects

```
private final QueueConnection qConnection;  
private QueueSession qSession;  
private QueueSender qSender;  
private TextMessage replyMsg;
```

The constructor takes the queue connection of the `CustomerProfiles` main method as an argument. The connection is thread safe, so it can be shared between `CustomerProfile` and `MsgHandler`. It is needed to create session and sender to send a reply if required.

Example 8-28 MsgHandler constructor

```
public MsgHandler(QueueConnection qc) {  
    this.qConnection = qc;  
}
```

The `onMessage` method needs to be implemented because it is part of the `MessageListener` interface. Whenever a message gets available on the queue

the listener is listening on, the `onMessage` method is called and the message is handed to it as an argument.

Example 8-29 onMessage method

```
public void onMessage(Message message)
    TextMessage requestMsg = null;
```

In the `onMessage` method it has to be verified, whether the message is in the format we expect it. JMS defines five different message types:

- ▶ `ObjectMessage` is an object serialized by the Java Runtime
- ▶ `TextMessage` is an encoded string
- ▶ `BytesMessage` is a sequence of bytes
- ▶ `MapMessage` is a string containing a set of XML name/type/value triplets, each given an element name
- ▶ `StreamMessage` is a string containing a set of XML name/type/value triplets without element names

When a WebSphere MQ application send a message with the format of `MQFMT_STRING` to a JMS client, it is mapped to a `TextMessage`. Both the BSS and the IAS send messages in the `MQFMT_STRING` format. A `TextMessage` is expected to arrive, other message types are not handled within the CPS.

Example 8-30 Test message type

```
if(message instanceof TextMessage)
```

The message object is cast to a `TextMessage` and the containing string can be received using the `getText` method.

Example 8-31 Extract message content

```
requestMsg = (TextMessage)message;
String content = requestMsg.getText();
```

The CPS offers to services, open new accounts and reply to queries for existing customer profiles. In our solution all messages are sent to the same input queue of CPS. So the message listener first has to check, which kind of message it just received. By first check if the message is to open a new account, a new account can be added to the customer profile database.

Note: Here **md** is not the message descriptor, but a `messageDetails` object. See `MessageDetails.java` in the “Use case 1” on page 321 for details.

Example 8-32 OpenAccount messages

```
if(md.GetMsgObjective().equals("OpenAccount"))  
    //This is a message to open a new account in customer profile database
```

If it is not a message to open a new account, check whether it is a message to query a customer profile. Other services are currently not offered in our solution, so messages with other message objectives are not supported.

Example 8-33 QueryCustomerProfile messages

```
else if(md.GetMsgObjective().equals("QueryCustomerProfile"))  
    //This is a message to query an existing account in the database
```

To send back a reply containing the customer profile needs a session. This session can be created using the connection from CustomerProfile. A non-transacted session with auto-acknowledge is used.

Example 8-34 Create session

```
qSession = qConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
```

The IAS uses dynamic queues and expects the CPS to send the replies to the appropriate one. Therefore a destination object is stored in the JMSReplyTo property of the requesting message. To create a queue sender this destination object has to be cast to a queue object.

Example 8-35 Get reply queue

```
Destination replyDest = message.getJMSReplyTo();  
Queue replyQueue = (Queue)replyDest;
```

The IAS is a .NET application that uses the WebSphere MQ classes for Microsoft .NET. It expects the reply message with the default message header MQMD followed by the message body in the format MQFMT_STRING.

When JMS applications send messages to a WebSphere MQ queue, the JMS implementation stores all JMS message information that cannot be mapped to the message descriptor or the message body in an extra header called “Rules and formatting header 2” (MQRFH2).

The JMS implementation does not recognize that the message replying on is sent by a non-JMS application. Therefore the queue object received in the JMSReplyTo property of the JMS request message acts like an administered object of type queue with TARGCLIENT(JMS). If this queue object is used, the reply is stored in the queue in the format shown in Table 8-4:

Table 8-4 JMS message mapped to WebSphere MQ message

MQMD	Message descriptor Format field: MQRFH2
MQRFH2	Rules and formatting header 2 Format field: MQSTR <mcd>Text</mcd> <jms>JMS header fields not mapped to MQMD</jms> <usr>Application-specific properties</usr>
Body	Message content: XML string

This message format is not expected by the IAS. The reply can be received, but the ReadString method returns the MQRFH2 and the body (See Example 8-15 on page 149). This string cannot be parsed by the XML parser, because it starts with the MQRFH2 structure. There are two possible solutions to solve this problem:

- ▶ The IAS handles this message type.
 After receiving the reply message, the IAS checks the format field of the message descriptor. If it contains the value MQFMT_RF_HEADER_2, it has to check the length of the header structure in the StrucLenght field of the MQRFH2. Then it can move to the end of the MQRFH2 using the Seek method of the MQMessage class and read the remaining string using the ReadString method.
- ▶ The CPS sends back messages without the rules and formatting header 2.
 The CPS checks if the message is sent by a non-JMS application using the getJMSDestination method. This method returns a destination object when the message is sent by a JMS application, it returns zero if it is sent by a non-JMS application. Then it casts the destination to MQDestination and sets the target client to the value JMSC.MQJMS_CLIENT_NONJMS_MQ.

We decided to handle this problem in the CPS. First check if the message contains a non-JMS destination. If it does, set the target client to MQ which will remove the MQRFH2. Then set the persistence of the queue to non-persistent. Non-persistent messages are needed, because the reply is sent to a temporary dynamic queue. This type of queue can only store non-persistent messages.

Note: This solution is not JMS compliant, because we use WebSphere MQ specific classes here.

At the time of writing, the WebSphere MQ JMS implementation does not recognize if the requester is a non-JMS client and does not react accordingly. This may change in the future, so use the latest CSD and test the JMS compliant code first.

Example 8-36 Handle non-JMS destinations

```
if (message.getJMSDestination() != null) {  
    ((MQDestination)replyQueue).setTargetClient(JMSC.MQJMS_CLIENT_NONJMS_MQ);  
    ((MQDestination)replyQueue).setPersistence(JMSC.MQJMS_PER_NON);  
}
```

The CPS creates a queue sender for the queue object received using the queue session.

Example 8-37 Create sender

```
qSender = qSession.createSender(replyQueue);
```

When a JMS client sends a `ObjectMessage` or `BytesMessage` to a WebSphere MQ application, the format field of the message is set to `MQFMT_NONE`, when it sends a `TextMessage`, `StreamMessage`, or `MapMessage`, it is set to `MQFMT_STRING`. The IAS expects a string format to contain the XML string, therefor we create a `TextMessage` object.

Copy the `JMSMessageId` of the received request to the `JMSCorrelationId` of the reply to send. These fields are mapped to the `messageId` and `correlationId` field of the message descriptor. The same persistence for the reply and the request is used.

Note: The persistence of the message we send is overwritten by the persistence setting of the `MQDestination` object, when the requesting application is a non-JMS client.

Example 8-38 Set up reply message

```
// create a reply message.  
replyMsg = qSession.createTextMessage();  
replyMsg.setJMSCorrelationID(message.getJMSMessageID());  
replyMsg.setJMSDeliveryMode(message.getJMSDeliveryMode());  
replyMsg.setText(xmlmsg);
```

The CPS puts the message on the reply queue.

Example 8-39 Send the reply

```
qSender.send(replyMsg);
```

Then it closes the session.

Example 8-40 Close the session

```
qSession.close();
```

When using a message listener, the application code cannot catch exceptions raised by failures to receive messages. This is because the application does not explicit calls to the receive method. To catch errors in asynchronous processing, an exception listener is required. An exception listener is not used, because JMS programming is not in the scope of this book and the team decided to keep the code simple. For an example of a exception listener refer to *MQSeries Programming Patterns*, SG24-6506.

For more details on the CPS application refer to the code in the “Use case 1” on page 321.

8.4 Deployment

Both the BSS and the CCS applications need to be deployed.

8.4.1 Deploying BSS

For instructions on how to deploy the BSS application refer to 7.4.1, “Deploying BSS” on page 134.

8.4.2 Deploying CPS

Before deploying the CPS application, check if all prerequisites are installed:

- ▶ WebSphere MQ V5.3 or later
 - Check if the following jar files are in the folder
C:\Program Files\IBM\WebSphere MQ\Java\lib
 - jms.jar
 - jndi.jar
 - fscontext.jar
 - connector.jar
 - providerutil.jar

- com.ibm.mqjms.jar
- com.ibm.mq.jar
- Check if the JMSAdmin tool is in the folder
C:\Program Files\IBM\WebSphere MQ\Java\bin
 - JMSAdmin.bat
 - JMSAdmin.config
- JRE 1.3.1 or later

If the jar files are in another directory, the cps.bat file has to be edited.

Create a directory C:\CPS and copy all CPS files from the appendix in this directory. When copying keep the directory structure. The CPS folder should look like this:

Example 8-41 CPS directory

```
CPSAdminObjects.txt
cps.bat
cps\CustomerProfile.java
cps\CustomerProfile.class
cps\MessageDetails.java
cps\MessageDetails.class
cps\MsgHandler.java
cps\MsgHandler.class
cps\OpenAccount.java
cps\OpenAccount.class
cps\QueryAccount.java
cps\QueryAccount.class
jar\jaxen-core.jar
jar\jaxen-jdom.jar
jar\jdom.jar
jar\saxpath.jar
jar\xerces.jar
jar\xml-apis.jar
jndi\
xml\CustomerProfile.xml
```

Setting up the JMS administered objects

Before running the CPS Java code, the JMS administered objects must be set up. The file store was used as the JNDI namespace to have less software dependencies.

1. Edit the file:
C:\Program Files\IBM\WebSphere MQ\Java\bin\JMSAdmin.config
 - Activate the RefFSContextFactory:

Example 8-42 Initial context factory settings

```
#INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory  
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory  
#INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory  
#INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WsnInitialContextFactory
```

- Set the URL to file:/C:/CPS/JNDI:

Example 8-43 URL settings

```
#PROVIDER_URL=ldap://polaris/o=ibm,c=us  
PROVIDER_URL=file:/C:/CPS/JNDI  
#PROVIDER_URL=iiop://localhost/
```

- Save JMSAdmin.config.
2. Run JMSAdmin. Use the provided CPSAdminObjects.txt to create the administered objects for the CPS.

Example 8-44 Run JMSAdmin

```
cd C:\Program Files\IBM\WebSphere MQ\Java\bin  
JMSAdmin < C:\CPS\CPSAdminObjects.txt
```

The output should look like this:



Figure 8-3 JMSAdmin output

8.5 Testing

1. Make sure the local queue manager DOTOP is running. Setup description for this queue manager and connections to other queue managers are provided in “WebSphere MQ setup” on page 112.
2. Open a command prompt and go to the CPS directory and run cps.bat.

Example 8-45

```
cd cps
cps
```

The command prompt window should look like this:

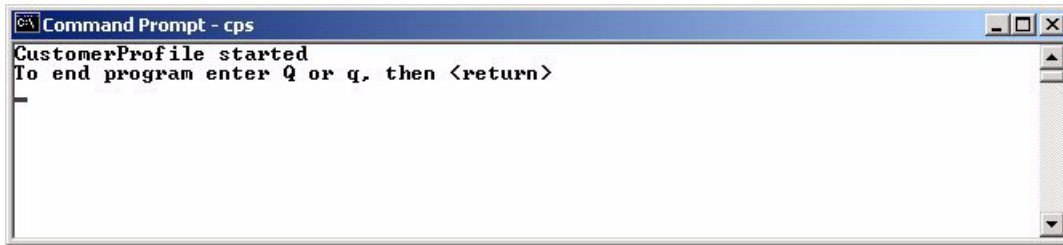


Figure 8-4 Running CPS application

3. Start BSS. For instructions on how to start the BSS refer to 7.5.1, “How to start BSS” on page 136.
4. Click the “Open Account” tab.
5. On the “Request Personal Details” page enter some data. Then click the “Submit Personal Details” button.
6. On the “Request Investment Details” page enter a initial deposit and click the “Submit Investment Details” button.
7. The “Account Open Successful” page opens returning a new account number.
8. Go to the CPS command prompt window. It should look like this:

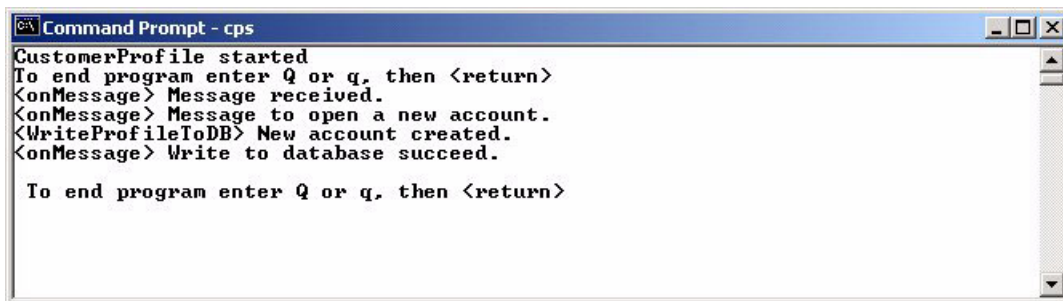


Figure 8-5 CPS open account

9. Close the BSS window.
10. End the CPS by entering “q” or “Q” in the CPS command prompt.

8.6 Alternative solutions

There are several possible ways to connect .NET and J2EE applications using messaging. This chapter shows the use of the WebSphere MQ classes for Microsoft .NET in the .NET application and the use of the JMS in the J2EE application. The .NET application uses the WebSphere MQ specific API while the J2EE application uses an open standard API. The J2EE application could also include Message Driven Beans to receive messages from the .NET application.

There are some advantages for using an open standard, such as protection of investment and the ability to plug in other JMS implementations. The JMS also offers some extra functions, not available in the WebSphere MQ classes for Java. These functions are:

- ▶ Asynchronous message delivery
- ▶ Message selectors
- ▶ Support for publish/subscribe messaging

Another advantage of using the JMS is the API is more abstract than the WebSphere MQ API. The implementation details are left to the JMS provider.

There are some limitations in the use of the JMS, when JMS clients communicate with native WebSphere MQ applications. Some provider-specific functions can be used in JMS clients, but these clients are not longer independent on the JMS provider and so only run with WebSphere MQ. These functions are:

- ▶ Reports
- ▶ WebSphere MQ message types
- ▶ Last message in group
- ▶ Format, encoding and character set

8.6.1 WebSphere MQ classes for Microsoft .NET and WebSphere MQ classes for Java

Whenever the use of an open standard is not required the WebSphere MQ classes for Java can be used within the J2EE application.

Disadvantages of the use of the WebSphere MQ classes for Java are:

- ▶ The WebSphere MQ API is more complex than the JMS API.
- ▶ Implementation details have to be considered by the application programmer.
- ▶ There is no direct support for publish/subscribe messaging.

But there are also some advantages:

- ▶ All WebSphere MQ functions can be used.

- Better performance.

The use of the WebSphere MQ classes for Java is very similar to the use of the WebSphere MQ classes for Microsoft .NET, so it is not shown in this redbook. There are examples of Java applications using the redbook *MQSeries Programming Patterns*, SG24-6506.

8.6.2 Web Services

Web Services are implemented in .NET and J2EE, so using Web Services to invoke services and to exchange data is another option to connect .NET and J2EE.

The invocation of a J2EE Web Service from a .NET application using WebSphere MQ Transport for SOAP is described in Chapter 10 on page 207.

The invocation of a .NET Web Service from a J2EE application is not shown in this redbook, because this is handled by the WebSphere MQ Transport for SOAP.

8.6.3 Bridge between WebSphere MQ and Microsoft Message Queuing (MSMQ)

An alternative way to connect .NET and J2EE is to use Microsoft Message Queuing in .NET and WebSphere MQ in J2EE, and use a bridge between these two messaging middleware products. This is a good choice, when .NET applications that already use Microsoft Message Queuing have to communicate with existing WebSphere MQ applications.

However, it has a disadvantage — that it becomes necessary to administer WebSphere MQ, Microsoft Message Queuing and the bridge.

Using WebSphere MQ classes for Microsoft .NET requires configuration of WebSphere MQ only.

There are many bridges between Microsoft Message Queuing and WebSphere MQ available from different vendors. One example is the MSMQ-Bridge provided by Microsoft as part of the Host Integration Server 2000.



Messaging solution: .NET client to .NET Web Services using WebSphere MQ SOAP transport

The previous chapter discusses the scenario where a .NET client connects to a J2EE application using WebSphere MQ infrastructure. This chapter introduces a Web Service called Investment Advisory Web Service (IAS). YuBank provides this service to its prospective investors. Based on the customer profile, IAS is intended to provide suitable Share investment advice.

YuBank is planning to upgrade its bank Web application (BSS) to include an investment advice user interface. YuBank also plans to develop the advisory Web Service in a .NET environment.

Web Services use Simple Object Access Protocol (SOAP) as a communication protocol and HTTP as a transport protocol for SOAP. However, YuBank has an existing investment in IBM WebSphere MQ and prefers WebSphere MQ over HTTP for its reliability and guaranteed delivery aspects.

This chapter describes how a .NET Web Application communicates with a Web Service (.NET) using WebSphere MQ as a SOAP transport protocol.

The contents of this chapter are organized as follows:

- ▶ Process overview: explains the high level process
- ▶ System Context: describes the interactions between systems and message definitions
- ▶ Development: Solution development is distributed in three sub-sections
 - IAS Web Service development
 - Proxy for IAS Web Service using WebSphere MQ Transport for SOAP
 - Bank Web Application (BSS) and communication with IAS
- ▶ Deployment: This section describes deployment of all three component mentioned above
- ▶ Testing: Basic testing of Web Application, IAS Web Service and proxy for SQS is outlined in this chapter.

Prerequisite: It is important to understand the basic principles of using WebSphere MQ Transport for SOAP described in Chapter 3 before reading this chapter.

9.1 Process overview

The figure below illustrates the use case view of the scenario used in this chapter.

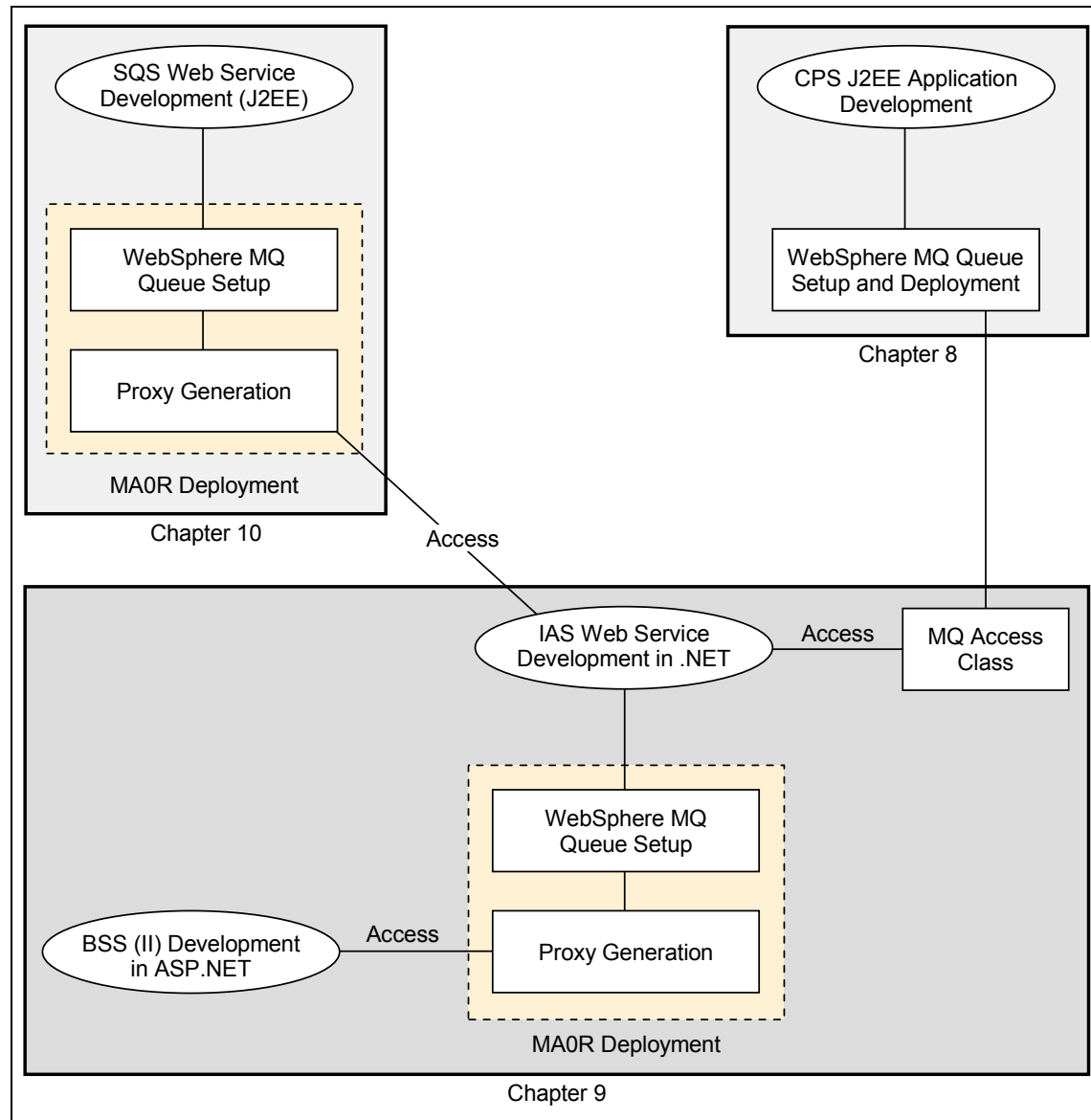


Figure 9-1 Process view of IAS Web Service

The process

Customer accesses the Bank Service System (BSS) for investment application. The Customer provides the YuBank client identification number and the proposed investment amount.

Based on the customer's financial background and investment related expectations, the Investment Advisory System (IAS) is intended to provide share investment advice.

The IAS is a Web Service which communicates with the YuBank Customer Profile System to retrieve financial data of the prospective investor.

The IAS then communicates with the Share Quote System for obtaining share prices and historic performance data via Web Service. The IAS analyzes the stock data and determines the suitable shares for the investor. The IAS then selects the appropriate quantity matching client's investment capacity and prepares a formal advice statement.

The BSS coordinates communication between the YuBank client and the IAS Web Service and delivers the investment advice in single Web session.

9.2 System context

As explained above, this scenario involves federation of services provided by various internal and external systems. This section outlines the message view involved in the processes described in previous section.

Figure 9-2 illustrates the message sequence involving various parties and their environments.

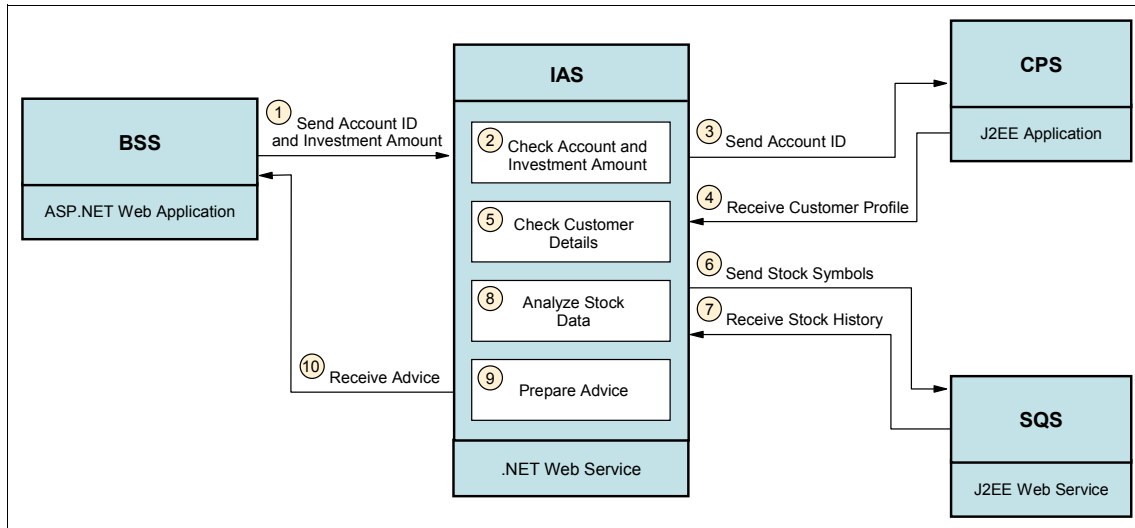


Figure 9-2 Message flow

The process description and message types of steps denoted on the Figure 9-2 are discussed below:

Step 1: Request from BSS to IAS Web Service using WebSphere MQ Transport for SOAP

Description - Customer accesses the Bank Service System (BSS) and sends account information along with proposed investment amount to Investment Advisory (IAS) Web Service.

Message Format: A SOAP message containing the following fields are passed from BSS application to IAS Web Service via a WebSphere MQ queue.

Field	Type	Constraint
CustomerId	string	required
InvestmentAmt	double	required and greater than 1000

Note: Type and constraints mentioned above are enforced in the user interface only. The SOAP message carries string representation of the values of both parameters with no range validation.

A SOAP message that arrives in the associated WebSphere MQ queue is shown Example 9-1.

Example 9-1 SOAP request from BSS to IAS Web Service

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://tempuri.org/" xmlns:types="http://tempuri.org/encodedTypes"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <tns:GetAdvice>
      <accountId xsi:type="xsd:string">198</accountId>
      <investAmt xsi:type="xsd:string">6000</investAmt>
    </tns:GetAdvice>
  </soap:Body>
</soap:Envelope>
```

Step 2: Amount and account ID check by IAS Web Service

Description: Account number check is performed and amount verified as per the criteria (more than 1000) by IAS Web Service.

Step 3: Customer profile request from IAS Web Service to CPS

Description: The account ID provided by the customer is transformed into an XML message and submitted to Customer Profile Service (J2EE App) via WebSphere MQ

Here is the message format:

Example 9-2 Input message from IAS Web Service to CPS

```
<Message MsgObjective="QueryCustomerProfile">
  <Account AccountNum="98765" />
</Message>
```

Step 4: Customer profile response from CPS J2EE application to IAS Web Service

Description: CPS application sends the results of the customer query back to IAS Web Service via a WebSphere MQ queue. MQAccessCPS class provides the functionality for the following tasks:

Message Formats: The following example shows an output message in XML format:

Example 9-3 Output message from CPS (via MQAccessCPS) to IAS Web Service

```
<?xml version="1.0" ?>
<Message>
```

```
<Account MsgObjective="ReplyToQuery" RiskLevel="3" InvestPeriod="12"
RetExp="10" FamilyIncome="3242143913" ExistAsset="37482155123"
CurDebt="23423552" />
</Message>
```

Step 5: Customer profile check

Description: The IAS Web Service ensures that the customer details are found and the customer has completed the profile information.

Step 6: Share quotes request from IAS Web Service to SQS Web Service using WebSphere MQ Transport for SOAP

Description: YuBank sends a list of preselected stock symbols to the Share Quote Service (SQS). The SQS has been developed in J2EE environment as a Web Service. Using WebSphere MQ transport for SOAP, a proxy class is generated for SQS access. The IAS Web Service uses this proxy class in its code to receive share quote information from SQS.

Discussion on share quote Web Service (SQS) and proxy generation using WebSphere MQ Transport for SOAP can be found in 10.3, “Development” on page 213 and 9.5, “Testing” on page 199.

Message format: Using SQS proxy IAS invokes the GetQuotes service by passing stock symbols in XML format.

Example 9-4 Share quote request from IAS to SQS Web Service

```
<ShareQuoteInquiry><Share><name>IBM</name></Share><Share><name>MSFT</name></Share></ShareQuoteInquiry>
```

This call is then converted into a SOAP request. Example 9-5 shows stock symbol XML encapsulated in the SOAP request to Share Quote Web Service (SQS).

Note: The developer of the Web Service does not need to understand the SOAP messages shown below. This part is included to give an understanding of lower level communication that takes place between a Web Service and its clients. You can safely ignore the SOAP messages in Example 9-4 and Example 9-5 and related discussion.

Web Services framework uses Simple Object Access Protocol (SOAP) for communication with client programs. WebSphere MQ Transport for SOAP delivers the SOAP messages involved in Web Service communication. It is important to note that the WebSphere MQ platform does not handle SOAP encoding or decoding. This part is managed by the SOAP layer running on client and server platforms.

Example 9-5 SOAP request transformation of share quote request

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="ShareQuote_Wmq" xmlns:types="ShareQuote_Wmq/encodedTypes"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <tns:getQuote>
    <in0 xsi:type="xsd:string"><?xml version="1.0"
      encoding="UTF-8"?><ShareQuoteInquiry><Share><name>IBM</name></Share><Share><name>MSFT</name></Share></ShareQuoteInquiry></in0>
    </tns:getQuote>
  </soap:Body>
</soap:Envelope>
```

Step 7: Share quote history response from SQS to IAS

Description: The Share Quote Web Service (SQS) provides historic stock data and the current price for the requested shares. Overall business logic and development of Stock Quote Web Service in J2EE environment is discussed in Chapter 10, “Messaging solution: .NET client to J2EE Web Services using WebSphere MQ SOAP transport” on page 207.

Message Format: The following example shows the SOAP response received from Share Quote Service (SQS).

Example 9-6 Share quote response from SQS to IAS Web Service

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

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
  <ns1:getQuoteResponse
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:ns1="ShareQuote_Wmq">
    <getQuoteReturn xsi:type="xsd:string">
      <ShareQuoteResult>
        <Share>
          <name>IBM</name>
          <price>85</price>
          <history>
            <m1>1.5</m1>
            <m3>3.3</m3>
            <m6>6</m6>
            <m9>9.1</m9>
            <m12>12</m12>
          </history>
        </Share>
        <Share>
          <name>MSFT</name>
          <price>25</price>
          <history>
            <m1>1</m1>
            <m3>-3</m3>
            <m6>9</m6>
            <m9>5</m9>
            <m12>8</m12>
          </history>
        </Share>
      </ShareQuoteResult>
    </getQuoteReturn>
  </ns1:getQuoteResponse>
</soapenv:Body>
</soapenv:Envelope>

```

Step 8: Share analysis and selection by IAS

Description: After receiving share data from SQS, IAS Web Service analyzes the data as described in “IAS Business logic” on page 177. Shares matching customer’s risk level and return expectation are selected.

Step 9: Prepare advice

Description: Based on the investment amount provided by the customer, IAS selects quantities of selected shares and prepares the advice statement.

Step 10: Advice response from IAS Web Service to Bank application (BSS)

Description: IAS returns the advice using WebSphere MQ Transport for SOAPWeb Service

Message Formats: Advice XML contains two elements Advice its child element - Errors. In normal case the process returns the advice statement as shown in Example 9-7.

Example 9-7 Advice XML return by IAS Web Service

```
<Advice>According your profile, you consider investing in following  
stocks.{symbol[Quantity]} IBM[71]<Errors></Errors></Advice>
```

As explained in Chapter 3, WebSphere MQ transport for SOAP carries SOAP representation of Web Services communication. The following example shows SOAP encoding for advice message.

Example 9-8 Advice from IAS Web Service in SOAP response format

```
<?xml version="1.0" encoding="utf-8" ?>  
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"  
  xmlns:tns="http://tempuri.org/" xmlns:types="http://tempuri.org/encodedTypes"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">  
    <tns:GetAdviceResponse>  
      <GetAdviceResult xsi:type="xsd:string"><Advice>According your  
        profile, you consider investing in following  
        stocks.{symbol[Quantity]}  
        IBM[71]<Errors></Errors></Advice></GetAdviceResult>  
    </tns:GetAdviceResponse>  
  </soap:Body>  
</soap:Envelope>
```

In case of errors, as shown in the following example, the IAS inserts appropriate error message and its point of origin under Error element of advice statement.

Example 9-9 Error message returned by IAS Web Service

```
<Advice><Errors><error>Customer profile service error[CompCode: 2, Reason:  
2033]</error></Errors></Advice>
```

Error handling in the IAS Web Service is described in “Error handling in the Web Service” on page 202 of this chapter.

9.3 Development

The IAS Web Service is utilized by the Bank application to provide advice to prospective investors. As explained in 9.2, “System context” on page 166, the IAS accesses two other services internally. Remember that Investment Advisory Web Service (IAS) plays dual role in this scenario. Its major role is to provide advice service to its client - the Bank application (BSS). However, to fulfill this role, IAS requires services from other sources such as SQS. In this case the same IAS Web Service acts as a client for the SQS Web Service.

Figure 9-1 on page 165 explains the overall development approach for this use case scenario.

In this section we describe:

- ▶ IAS Web Service solution development
- ▶ Proxy generation using WebSphere MQ Transport for SOAP
- ▶ BSS Web Application design and integration of IAS

9.3.1 .NET Web Service development

The IAS is a .NET Web Service developed using Microsoft Visual Studio .NET. The Application wizard provided by Microsoft Visual Studio .NET creates a template class. This class inherits from System.Web.Services.

The following figure shows the New Project wizard provided by Microsoft Visual Studio .NET.

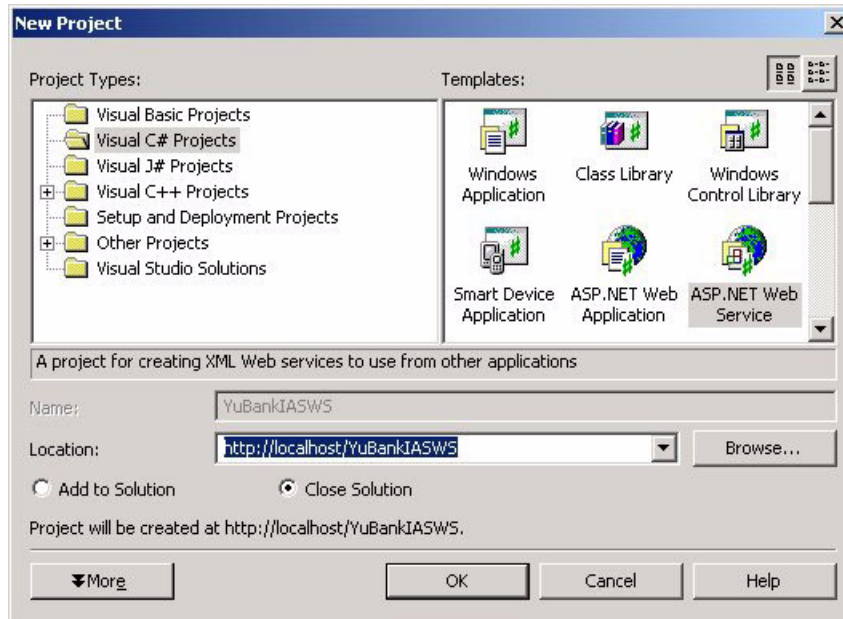


Figure 9-3 New .NET Web Service project wizard

In the generated code un-comment the function marked with [WebMethod] attribute and write a service functionality. Metadata attributes inserted into the code provide instructions to the .NET Framework for managing tasks such as identity, transaction, security, version control, deployment, message encoding and so on.

The [WebMethod] attribute identifies a Web Service within the class. It is possible to have multiple Web Services in one class. In fact almost all object oriented functionality can be implemented in the Web Service class. Appropriate tags might need to be included while doing advanced Web Service programming. For example, using overloading for Web Methods, the MessageType attribute must be used.

[WebService] attributes provide general information about the service such as Name, Description, Namespace and Type.

Other such attributes include [SoapRpc] - for default SOAP encoding. For complete listing of Web Service related attributes, refer to the following URL:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconextendingmetadatausingattributes.asp>

The following code snippet shows the Portfolio class declaration and the GetAdvice Web Method.

Example 9-10 Web Service and Web Method declaration and use of attributes

```
[WebService (Description="Following Web Service is part of scenario described
in IBM Redbook - 'Developing WebSphere
messaging",Namespace="http://redbooks.com/webservices",Name="YuBank - Portfolio
Web Service")]
public class Portfolio : System.Web.Services.WebService
{
    private string _accountId;
    private int _riskLevel=0;
    private double _expectedReturns=0.0;
    private double _amtInvest=0.0;

    private double[] nodeValues;
    private double _maxVariance=0.00;

    /// Advice will be returned as an xml string. following are related xml
    objects

    private XmlDocument _xmlAdvice;
    private XmlElement _rootElAdvice;
    private XmlElement _errorElAdvice;
    private XmlDocument xmlSelectedStocks;
    private XmlElement rootElStocks;

    private int _HistoryRecords ;

    public Portfolio()
    {
        //CODEGEN: This call is required by the ASP.NET Web Services Designer
        InitializeComponent();
    }

    [WebMethod (TransactionOption=TransactionOption.RequiresNew)]
    public string GetAdvice(string accountId, string investAmt)
    {

        //business logic code

    }
```

ASMX file

An asmx file extension is the entry point for .NET Web Service processing. It contains the processing directive (WebService) and the reference to the XML Web Service class.

```
<%@ WebService Language="c#" Codebehind="Portfolio.asmx.cs"
Class="YuBankIASWS.Portfolio" %>
```

9.3.2 IAS Web Service solution

YuBank's Investment Advisory Web Service is developed using Microsoft .NET Framework and Microsoft Visual Studio .NET. Apart from normal Web Service related files and classes, it contains access or classes and proxy classes for the following services:

1. Customer Profile Service (CPS) - A J2EE application that provides customer data. It connects to this service using WebSphere MQ classes for Microsoft .NET. A detailed description about WebSphere MQ aspects of this connectivity can be found in Chapter 8, "Messaging solution: .NET application to J2EE application" on page 139.
2. Share Quote Service (SQS) - A J2EE Web Service that provides Share Quote data. IAS Web Service connects this service using WebSphere MQ Transport for SOAP. Concepts and examples of WebSphere MQ Transport for SOAP are explained in Chapter 3, "WebSphere MQ Transport for SOAP" on page 11.

IAS is a simple .NET Web Service class which offers the investment advice Web Service. Example 9-10 on page 175 shows the Portfolio class declaration generated using Microsoft Visual Studio .NET. The GetAdvice method of Portfolio class is marked with [WebMethod] tag and thereby exposed as a Web Service.

The following figure illustrates the entire class structure of the YuBankIASWS project.

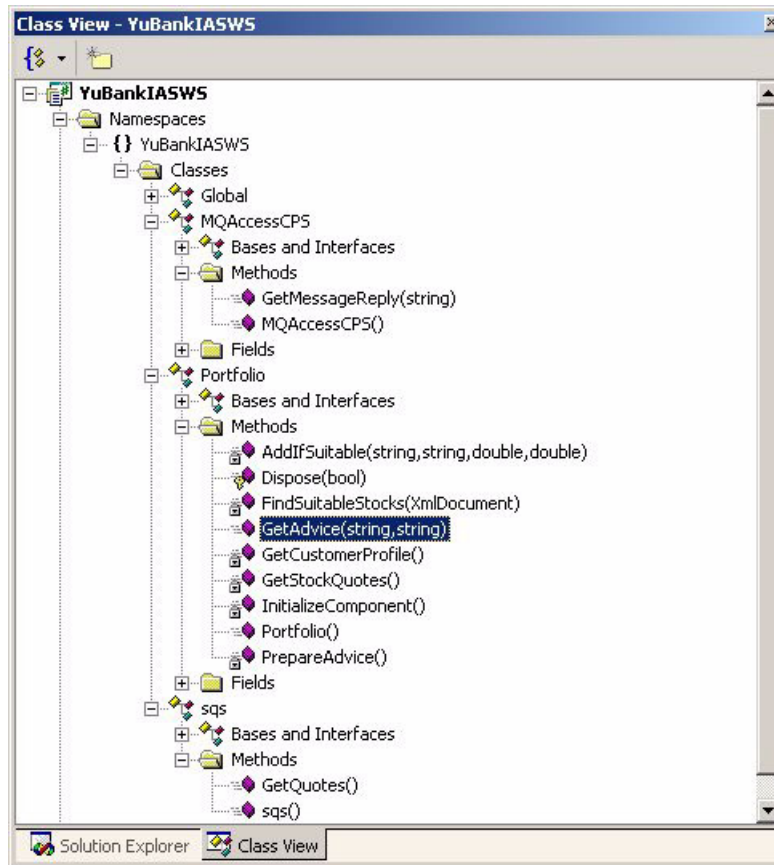


Figure 9-4 Class model for IAS Web Service solution

IAS Business logic

In this section the class structure of YuBankIASWS project and the business logic expressed within its functions is described. Class and method descriptions are arranged according to the sequence of steps shown in Figure 9-2 on page 167.

► Class Method: **Portfolio.GetAdvice**

(Step 2: Amount and account id check by IAS Web Service)

Description: This is the only WebMethod offered by the Portfolio class. This method calls all other services using the respective functions. These calls are arranged so that possible errors result in immediate and graceful exits from the Web Service.

Before moving on to other business logic, this method checks the input parameters provided by the Web Service caller - BSS Web Application.

Parameters and return values: accountId (string) investmentAmt (string)

Business Logic: The following code fragment performs the check over the parameters.

Example 9-12 GetAdvice WebMethod and initialization of variables

```
public string GetAdvice(string accountId, string investAmt)
{
    if(accountId.Length>0)
    {
        _accountId=accountId;
        try
        {
            _amtInvest=Convert.ToDouble(investAmt);
        }
        catch (Exception ex)
        {
            //write error message in advice xml
            XmlElement er1 = _xmlAdvice.CreateElement("error");
            er1.InnerText="Invalid investment amount ["+ex.GetType()+"]";
            _errorElAdvice.AppendChild(er1);
        }
        //Verify that Client through BSS has entered appropriate investment
        amount
        if(_amtInvest>1000)
        {
            //Make sure that both CPS(J2EE App) and SQS WS are successful before
            preparing the advice
            if (GetCustomerProfile() && GetStockQuotes())
                PrepareAdvice();
        }
    }
}
```

Exception handling in the IAS Web Service is covered later in this chapter.

► Class / Method: **Portfolio.GetProfile() (Part I)**

(Step 3: Customer profile request from IAS Web Service to CPS)

Description: This part of the method calls the WebSphere MQ queue accessor function

Input parameters and return values: GetCustomer profile returns boolean value. Value of this boolean return is determined by the outcome of CPS service invocation.

Business Logic: The following code shows the XML transformation and the call to WebSphere MQ queue accessor function which results in CPS invocation.

Example 9-13 GetCustomerProfile method

```
private bool GetCustomerProfile()
{
    string strProfileXml="";
    XmlDocument xmlCps= new XmlDocument();
    try
    {
        MQAccessCPS proxyCPS = new MQAccessCPS();
        strProfileXml =
        proxyCPS.GetMessageReply("<Message
        MsgObjective='QueryCustomerProfile'><Account
        AccountNum='"+_accountId+"' /></Message>");
        xmlCps.LoadXml(strProfileXml);
    }
    catch(Exception ex)
    {
        XmlElement er7 = _xmlAdvice.CreateElement("error");
        er7.InnerText="Customer profile service error["+strProfileXml+"]";
        _errorElAdvice.AppendChild(er7);
        return false;
    }
}
```

► Class Method: **MQAccess.GetMessageReply**

(Step 4: Customer profile response from CPS J2EE application to IAS Web Service)

Description: The CPS application sends the results of customer query back to a WebSphere MQ queue. A helper (MQAccessCPS) class collects this reply from the queue and delivers it back to the caller (Portfolio.GetProfile())

MQAccessCPS class provides functionality for the following tasks:

- a. submit the Profile request to a WebSphere MQ queue
- b. receive reply from CPS J2EE application in return WebSphere MQ queue
- c. correlates the response with the original request
- d. return message (XML format) to the caller

Business logic: Refer to 8.3.2, “Investment advisory application” on page 148 which describes how MQAccessCPS class handles WebSphere MQ message submit and reply collection operations.

► Class/Method: **Portfolio.GetCustomerProfile (Part II)**

(Step 5: Check customer profile)

Description: This part of GetCustomerProfile methods checks the CPS response for valid customer details.

Business Logic: This part of the method receives reply from MQAccessCPS and transform it into an XML document. In case of an error on CPS's part, MQAccess does not return a valid XML string and thus GetProfile operation fails to complete. The following code fragment shows this part of the GetProfile() method:

Example 9-14 Validate customer profile data

```
private bool GetCustomerProfile()
{
    string strProfileXml="";
    XmlDocument xmlCps= new XmlDocument();
    try
    {
        MQAccessCPS proxyCPS = new MQAccessCPS();
        strProfileXml =
        proxyCPS.GetMessageReply("<Message
        MsgObjective='QueryCustomerProfile'><Account
        AccountNum='"+_accountId+"'></Message>");
        xmlCps.LoadXml(strProfileXml);
    }
    catch(Exception ex)
    {
        XmlElement er7 = _xmlAdvice.CreateElement("error");
        er7.InnerText="Customer profile service error["+strProfileXml+"]";
        _errorElAdvice.AppendChild(er7);
        return false;
    }

    if(xmlCps.DocumentElement.HasChildNodes)
    {
        //Code for coversion of Risk level and Expected return values
        return true;
    }
    else
    {
        //if customer is not in the CPS database then write appropriate
        message in advice xml
        XmlElement er6 = _xmlAdvice.CreateElement("error");
        er6.InnerText="No Customer record found [CPS Application]";
        _errorElAdvice.AppendChild(er6);
        return false;
    }
}
```

► Class/Method: **Portfolio.GetStockQuotes**

(Step 6: Stock quotes request from IAS Web Service to SQS Web Service using WebSphere MQ Transport for SOAP)

Description: This method sends pre-selected stocks symbols (XML string) to the SQS proxy for stock information. Figure 9-5 depicts the sequence of method calls involved in sending and receiving responses from SQS.

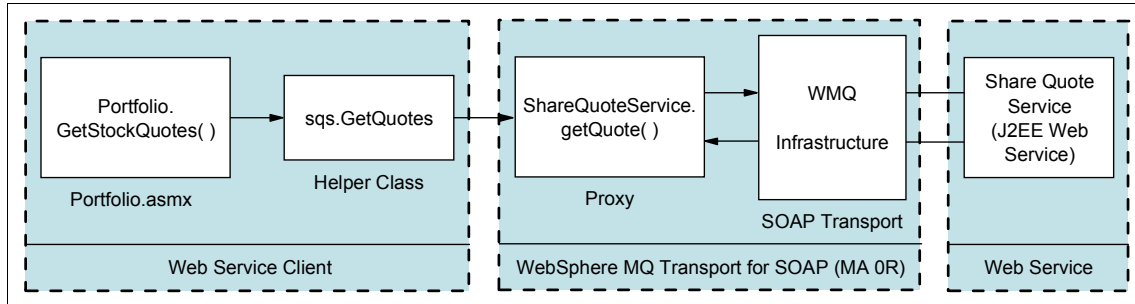


Figure 9-5 Invoking SQS Web Service (J2EE) from IAS .NET Web Service

Note: The helper class (sqs) is used here for a specific test and debugging purposes. In normal practice we recommend to use proxy generated by WebSphere MQ transport for SOAP (ShareQuoteService.cs) directly in the client code.

► Class/Method: **sqs.GetQuotes**

(Step 7: Get share quote history response from SQS to IAS)

Parameters and return values: No input parameter, Method returns string value.

Description: sqs class invokes the SQS Web Service using ShareQuoteService proxy generated using the WebSphere MQ Transport for SOAP utility.

Business Logic: Pre-selected shares are stored on a local folder in XML format. Stock pre-selection is a manual process and depends on external factors such as recommendations, bank policy and so on. The following code snippet shows how pre-selected shares are transferred to SQS using getQuote method of the SQS proxy. Results of this method call are returned to the caller (Potfolio.GetStockQuotes), and in case of errors accessing SQS service, this function returns error message to the caller instead.

Example 9-15 Receive share data from SQS service

```
public string GetQuotes()
{
```

```

string strSQSResponse = "";
XmlDocument xmlSQSRequest = new XmlDocument();
xmlSQSRequest.Load(@"C:\Inetpub\wwwroot\YuBankIASWS\requestSQS.xml");

try
{
    ShareQuoteService shareobj = new ShareQuoteService();
    shareobj.Url =
        "wmq:SOAP.ShareQuote@DOTEP?connectQueueManager=DOTIP,replyToQueue=SOA
        P.RESPONSE.ShareQuote";
    strSQSResponse = shareobj.getQuote(xmlSQSRequest.OuterXml);
    return strSQSResponse ;
}
catch (System.Exception e)
{
    return e.GetType().ToString();
}
}

```

wmq Url property of the proxy class and other deployment concepts are explained in Chapter 3., “WebSphere MQ Transport for SOAP” on page 11.

► Class/Method: **Portfolio.FindSuitableStocks Portfolio.AddIfSuitable**

(Step 8: Share analysis and selection by IAS)

Input parameters and return values: FindSuitableStocks accepts the XmlDocument object (containing SQS result)

Description: This method analyzes the share performance and calculates variance and average.

Business logic: After receiving share data from SQS, the IAS Web Service analyzes the data as described in 9.2, “System context” on page 166.

Calculate average return of the shares from SQS historic data using the formula explained in 4.2.2, “Use case 2: Investment advisory” on page 76.

Calculate the fluctuating value of shares using the following formula:

Example 9-16 Fluctuating value formula

$$\frac{\text{MAX}[(\text{average return}-\text{minimum return}),(\text{maximum return}-\text{average return})]}{\text{average return}}$$

Note: This formula and other calculations in this code are entirely fictitious. The shares selection process in reality is a very complicated process and often involves human intervention.

```
private void FindSuitableStocks(XmlDocument xmlSQResult)
{
    _HistoryRecords =(12/_HistoryPeriod)+1;
    //standard xml iteration logic
    XmlNode root,row;
    root = xmlSQResult.DocumentElement;
    IEnumerator ienum;
    ienum=root.GetEnumerator();

    while (ienum.MoveNext())
    {
        row =(XmlNode)ienum.Current;
        double aveReturn,variance,nodeValue=0.00;
        double totalReturns=0.00;
        int rCount = row.SelectSingleNode("history").ChildNodes.Count;
        nodeValues = new double[rCount];

        for (int i =0;i<rCount;i++)
        {
            nodeValue=Convert.ToDouble(row.SelectSingleNode("history").ChildNodes
            [i].InnerText);
            nodeValues[i]=nodeValue;
            //specific business logic formula. please refer related chapter in
            the redbook
            switch(i)
            {
                case 0:
                    totalReturns+=nodeValue*12;
                    nodeValues[i]=nodeValue*12;
                    break;
                case 1:
                    totalReturns+=nodeValue*4;
                    nodeValues[i]=nodeValue*4;
                    break;
                case 2:
                    totalReturns+=nodeValue*2;
                    nodeValues[i]=nodeValue*2;
                    break;
                case 3:
                    totalReturns+=nodeValue*1.33;
                    nodeValues[i]=nodeValue*1.33;
                    break;
                case 4:
                    totalReturns+=nodeValue*1;
                    nodeValues[i]=nodeValue*1;
                    break;
            }
        }
    }
}
```

```

    }
    aveReturn=totalReturns /rCount;
    double minVal,maxVal = 0.00;
    Array.Sort(nodeValues);
    minVal=nodeValues[0];
    maxVal=nodeValues[rCount-1];

    //specific business logic formula. you could also use standard deviation
    logic
    variance =(Math.Max((aveReturn-minVal),(maxVal-aveReturn)))/aveReturn;

    //creates an xml and adds the suitable stocks as records(elements)
    AddIfSuitable(row.SelectSingleNode("name").InnerText,row.SelectSingleNode("price").InnerText,aveReturn,variance);
}
}

```

Portfolio.AddIfSuitable

Description: This method adds the shares provided it matches the customer's expectation and maximum variance value.

Input parameters and return values:

Input: stock symbol (string), stock unit price (string), average return (double) calculated by FindSuitableStock method, variance (double) also calculated by FindSuitableStock.

Return value: Null. If the share matches the criteria it adds the share in XML document in the following format:

```
<selectedstocks><stock symbol='IBM' price='85' /></selectedstocks>
```

Example 9-18 Add suitable shares to the list

```

private void AddIfSuitable(string symbol, string price, double aveReturn,
double variance)
{
    if (aveReturn>=_expectedReturns && variance<=_maxVariance)
    {
        XmlElement elStock=xmlSelectedStocks.CreateElement("stock");
        elStock.SetAttribute("symbol",symbol);
        elStock.SetAttribute("price",price);
        rootElStocks.AppendChild(elStock);
    }
}

```

Note: Discussion on data manipulation techniques using ADO.NET is outside of the scope of this book. However, an attempt has been made to demonstrate a few XML data handling techniques through the sample solution programs. Code fragments in Example 9-17 and Example 9-18 show how to use System.Data.XML classes to:

- ▶ Iterate through XML document
- ▶ Navigate XML document using X-Path statements.
- ▶ Create an XML document

▶ Class/Method: **Portfolio.PrepareAdvice**

(Step 10: Advice response from IAS Web Service to Bank application (BSS))

Description: Advice is passed back to BSS Web application using WebSphere MQ Transport for SOAP Web Service.

Input parameters and return values: Null. This method uses selectedItems XML document and prepares investment advice XML.

Business Logic: This demonstration program distributes total investment amount (provided by the customer in BSS application) equally among the selected items. Based on the price of shares, PrepareAdvice calculates the quantity.

9.3.3 WebSphere MQ transport for SOAP deployment for IAS

Here the deployment of the IAS Web Service for use by the BSS client is illustrated. This Web Service returns a proposed investment portfolio for a given customer and target investment amount.

The Web Service itself is implemented in the file Portfolio.asmx.cs. The source for this service can be any convenient directory. It can also remain a normal Web project folder under inetpub\wwwroot. In our case it was copied in the folder c:\\$user\YuBankIASWS.

Important: As discussed in 9.3.2, “IAS Web Service solution” on page 176, the services makes use of several C# source files. The classname for the service itself is called “Portfolio” and this is declared within a namespace called “YuBankIASWS”. The Assembly Name for the application needs to be specifically set to the classname that defines the service otherwise the WebSphere MQ transport for SOAP deployment process fails. In this instance the Assembly Name is reset to “Portfolio”.

Then the service code is built using the Microsoft Visual Studio .NET IDE. This creates the DLL file Portfolio.dll in the sub-directory “bin” as shown in the following figure:

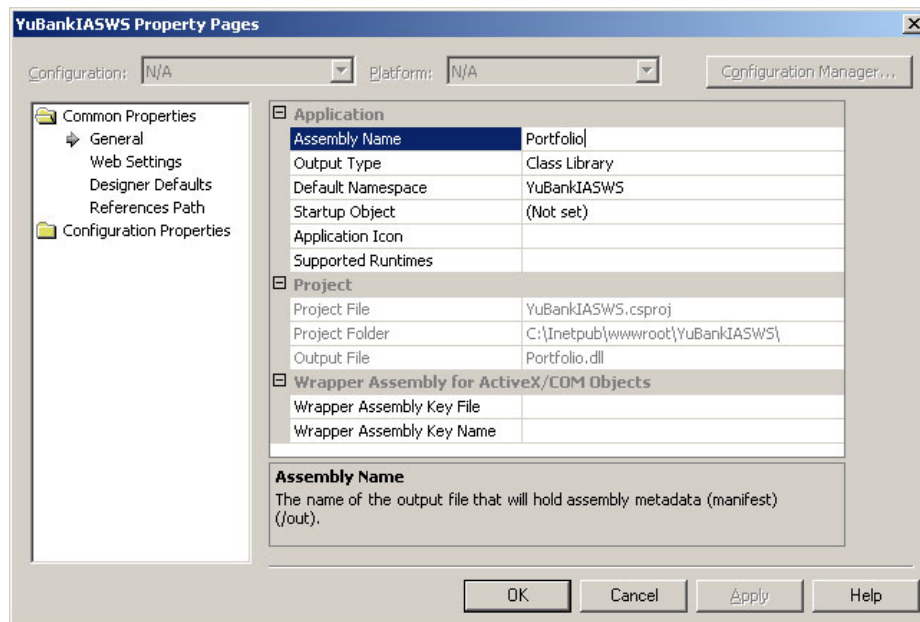


Figure 9-6 Resetting the assembly name in Microsoft Visual Studio .NET

Next an assembly directive (ASMX file) containing the following single line definition is created:

Example 9-19 The Assembly Directive file for the Portfolio service

```
<%@ WebService Language="c#" Codebehind="Portfolio.asmx.cs" Class="YuBankIASWS.Portfolio" %>
```

This directive specifies that the source for the service is located in Portfolio.asmx.cs and that its classname, with the namespace definition, is YuBankIASWS.generatePortfolio.

The service can now be deployed. Before doing so ensure that the CLASSPATH environment variable is set correctly for WebSphere MQ transport for SOAP. (Refer to 3.7.3, “Deploying the Microsoft .NET service” on page 39 for details.) The WebSphere MQ transport for SOAP deployment utility can now be invoked as follows:

Example 9-20 WebSphere MQ transport for SOAP deployment for the Portfolio Web Service

```
C:\$user\YuBankIASWS>deployWMQService -m DOTIP -f Port
```

```
folio.asmx
Package name: DefaultNamespace
Generating WSDL...
mqsoapwsdl wmq:SOAP.Portfolio@DOTIP?connectQueueManager=DOTIP Portfolio.asmx hel
pers\Portfolio_Wmq.wsdl
Preparing listener...
Configuring MQ...
Generating and compiling proxy code...
java com.ibm.mq.maOr.tools.RunWSDL2Java --output helpers -p dotNetService helper
s\Portfolio_Wmq.wsdl

C:\$user\YuBankIASWS>
```

When running the deployment utility use the -m flag to specify the queue manager DOTIP and the -f flag to specify the input ASMX file.

The deployment utility creates the queue SOAP.Portfolio. This is the queue into which our client BSS application writes request messages for a particular investment portfolio. No response queue is automatically generated by the utility however. The default response queue that WebSphere MQ transport for SOAP assumes is MQSOAP.RESPONSE.QUEUE. A different response queue specific to our service is required, in this case SOAP.RESPONSE.Portfolio. This response queue name can then be referenced in the URI as described above.

As well as creating the request queue, the deployment process also creates the proxy code that can be used by the BSS client application to invoke the Portfolio service. This proxy code is located in the file “helpers\Portfolio.cs”. Refer to “Adding IAS Web Service proxy” on page 191 for a description of how to build the client code with this proxy.

Deployment also creates the script “helpers\listen_Portfolio.bat”. This is the script to start the WebSphere MQ transport for SOAP Microsoft .NET listener. The script is specifically customized to run the listener for the service.

Before accessing this service from the BSS client the channel for communication between the BSS client and IAS server needs to be created. Such tasks cannot be performed by the deployment process as they are specific to the target WebSphere MQ topology. The channel is configured when running our WebSphere MQ setup scripts. Refer to “WebSphere MQ setup” on page 112 for more details.

9.3.4 BSS client

YuBank’s Banking Service System (BSS) is an ASP .NET application containing eight aspx pages of which two, RequestAdvice.aspx and DisplayAdvice.aspx, act

as clients to the IAS Web Service. One to collect information required by the Web Service, the other to display the response.

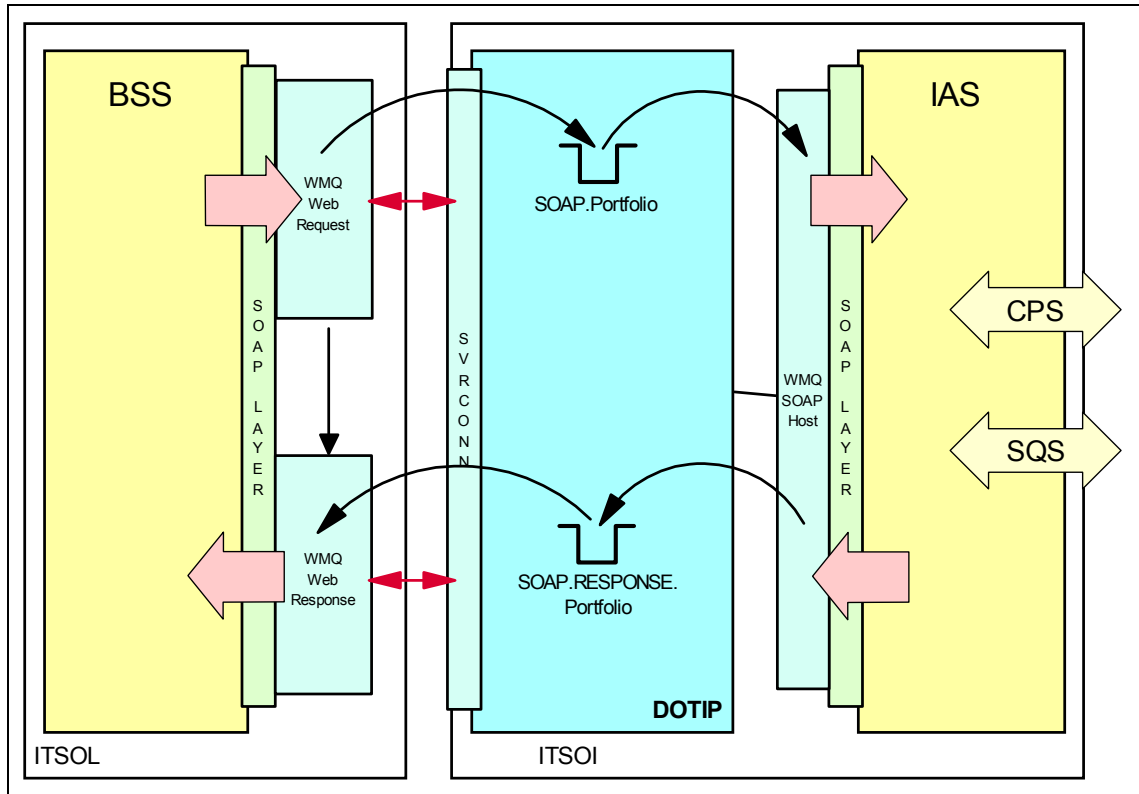


Figure 9-7 Communication between IAS Web Service and BSS Web Application using WebSphere MQ

For more details on the content of the BSS application, refer to 7.4.1, “Deploying BSS” on page 134.

BSS is developed using Microsoft Visual Studio .NET. Application Wizard provided by Microsoft Visual Studio .NET. The following figure shows the New Project wizard provided by Microsoft Visual Studio .NET to create ASP .NET Web application.

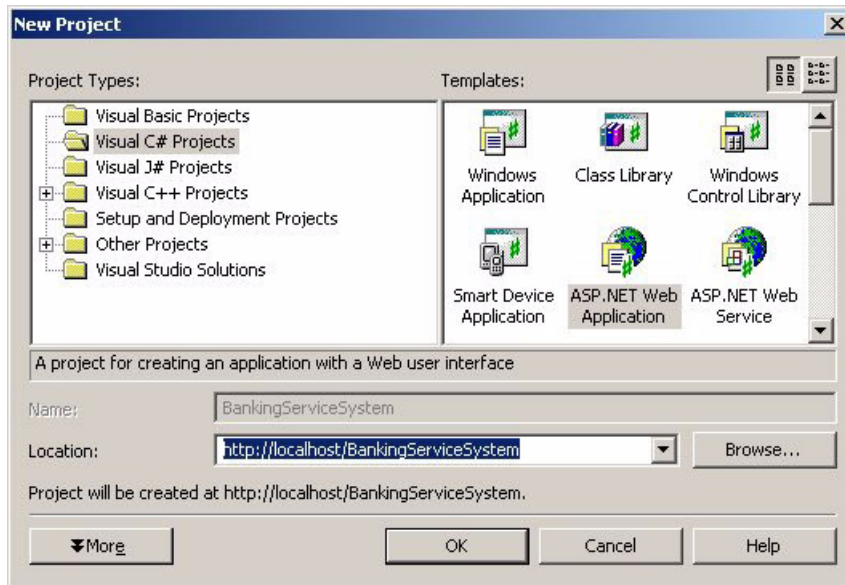


Figure 9-8 Microsoft Visual Studio .NET. Application Wizard

To use WebSphere MQ object's properties, methods, or events, WebSphere MQ Transport for SOAP has to be referenced, imported and registered.

Adding a reference to WebSphere MQ Transport for SOAP

Add Reference to System.Web.dll and MQSOAP.dll by right-clicking References in the Solution Explorer and selecting Add Reference. An Add Reference window appears.

In the .NET tab of the Add Reference window, double-click System.Web.dll.

Select Browse and locate the MQSOAP.dll. At the time of writing this redbook, the DLL was located in the MA0R bin directory. Click on it to add it to the selected components as shown below. Click **OK**.

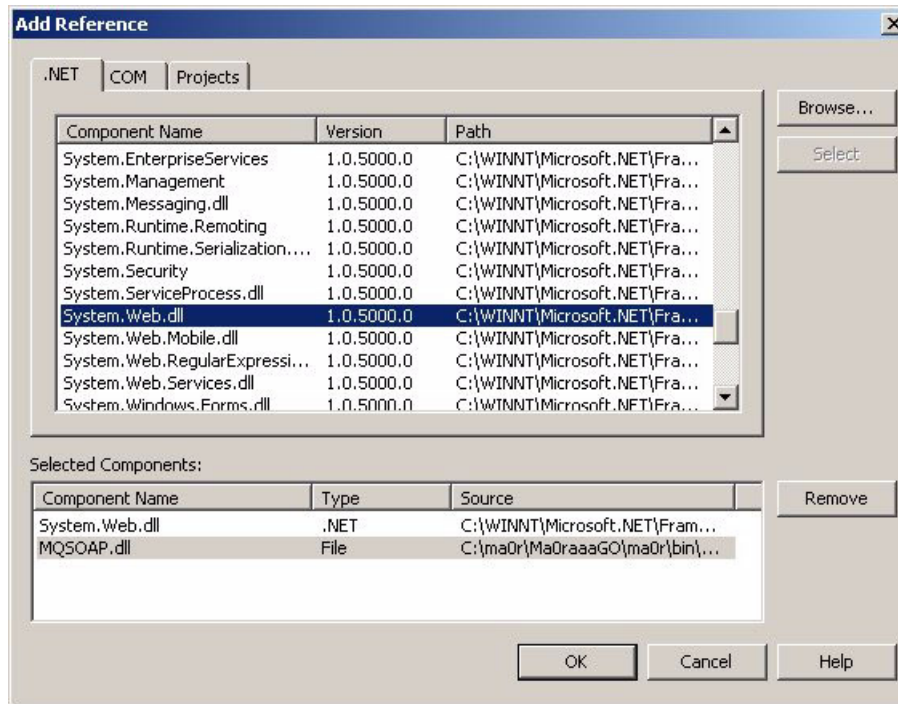


Figure 9-9 Adding reference to WebSphere MQ Transport for SOAP

Importing WebSphere MQ Transport for SOAP

In C# applications the appropriate classes are included as follows

Example 9-21 How to include WebSphere MQ classes in C#

```
using IBM.WMQ;
```

in VB.NET applications the appropriate classes are included as follows.

Example 9-22 How to include WebSphere MQ classes in VB.NET

```
Imports IBM.WMQ
```

Registering WebSphere MQ Transport for SOAP

To register WebSphere MQ Transport for SOAP, the following line of code has to be added to the page load function of your .aspx page which is used to access the Web Service proxy as shown below:

Example 9-23 Registering WebSphere MQ Transport for SOAP

```
private void Page_Load(object sender, System.EventArgs e)
{
    // register WMQ transport for SOAP
    try
    {
        MQSOAP.MQWebRequest.Register();
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine("exception caught:" + ex.ToString());
    }
}
```

Adding IAS Web Service proxy

On completing the three steps above, the Web Service proxy provided by the IAS Web Service should be copied into the solution. Right-click the project, and select **Add -> Add Existing Item** as shown below:

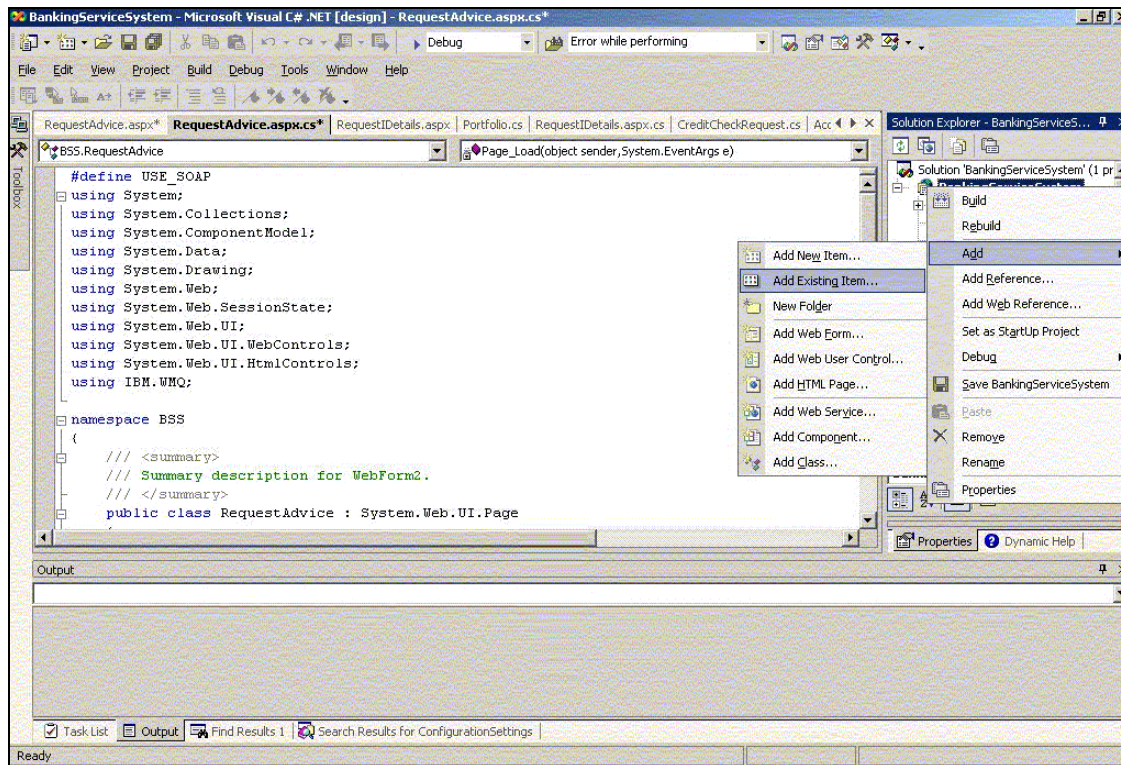


Figure 9-10 Adding the Web Service proxy to the solution

In the Add existing item window, select the Web Service proxy you want to add, in this case, portfolio.cs, and click **Open**.

9.3.5 BSS Web Application solution

BSS uses two aspx forms to access the IAS Web Service. One to collect information required by the Web Service, the other to display the response.

Design the forms as desired. One of the forms should have two text boxes and a button. The text boxes allow input of the investment amount and customer account number. These parameters are required by the IAS Web Service in order to return suitable advice to the customer. These parameters are wrapped up in a SOAP message and transported using WebSphere MQ to the IAS Web method. The IAS Web Service in turn returns an advice which can be displayed on the second form.

In this section the structure of the two forms used is described, RequestAdvice and DisplayAdvice.aspx pages and how they call the IAS Web Service.

RequestAdvice.aspx

In the form load method of the aspx form, RequestAdvice.aspx, designed to enter investment advice, register the WebSphere MQ Transport for SOAP as shown in “Registering WebSphere MQ Transport for SOAP” on page 190.

In the button click method, call the Web Service using the Web Service proxy provided, passing it the customers account number and investment amount as parameters shown in the snippet below:

Example 9-24 Calling Web Service from RequestAdvice.aspx

```
private void Button1_Click(object sender, System.EventArgs e)
{

    string acctNum = accNum.Text;
    string invAmt = invAmount.Text;

    try
    {

        //make new instance of Web Service proxy
        sendDetails = new Portfolio();

        //call services and assign advice the value of the response
        string advice = sendDetails.GetAdvice(acctNum, invAmt);

        //Store results in a session variable for display on DisplayAdvice.aspx
        Session.Contents["adviceResult"] = advice;
```

```

        Response.Redirect(Request.ApplicationPath + "/DisplayAdvice.aspx");
    }
    catch(MQException ex)
    {
        System.Console.WriteLine("MQException: compCode: " +
            ex.CompCode.ToString() + " Reason: " + ex.Reason.ToString());
        //Display error on a different page
        Response.Redirect(Request.ApplicationPath + "/DisplayError.aspx?msg=" +
            "Error occured while accessing IAS Web Service : MQException: compCode: " +
            ex.CompCode.ToString() + " Reason: " + ex.Reason.ToString());

    }
    catch (Exception sysex)
    {
        System.Console.WriteLine(sysex.ToString());
        //Display error on a different page
        Response.Redirect(Request.ApplicationPath + "/DisplayError.aspx?msg=" +
            "System error " + sysex.ToString());
    }
}

```

When the Web Service is called, a SOAP message is put on a queue, SOAP.Portfolio on ITSOI through channel TO.DOTIP and the response is taken from a queue called SOAP.RESPONSE.Portfolio. As described in the proxy code below, where portfolio is the name of the proxy:

Example 9-25 IAS Web Service proxy code

```

public Portfolio() {
    this.Url =
        "wmq:SOAP.Portfolio?clientConnection=ITSOI.hursley.ibm.com,clientChannel=TO.DOTIP,replyToQueue=SOAP.RESPONSE.Portfolio";
}

```

DisplayAdvice.aspx

The response from the IAS Web Service is stored in a session variable as shown below:

Example 9-26 Storing results in a session variable for display on another page

```

//Store results in a session variable for display on DisplayAdvice.aspx
Session.Contents["adviceResult"] = advice;
Response.Redirect(Request.ApplicationPath + "/DisplayAdvice.aspx");

```

And finally, the result from the Web Service is extracted from the session variable and displayed on DisplayAdvice.aspx.

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Put user code to initialize the page here
    string advice = Session.Contents["adviceResult"].ToString();
    adviceLabel.Text = advice;
}
```

9.4 Deployment

Both the BSS application and the CSS Web Service need to be deployed.

9.4.1 IAS Web Service deployment

There are two possible ways to deploy the IAS Web Service, the folder copy method and the Web Setup project method.

Folder copy method

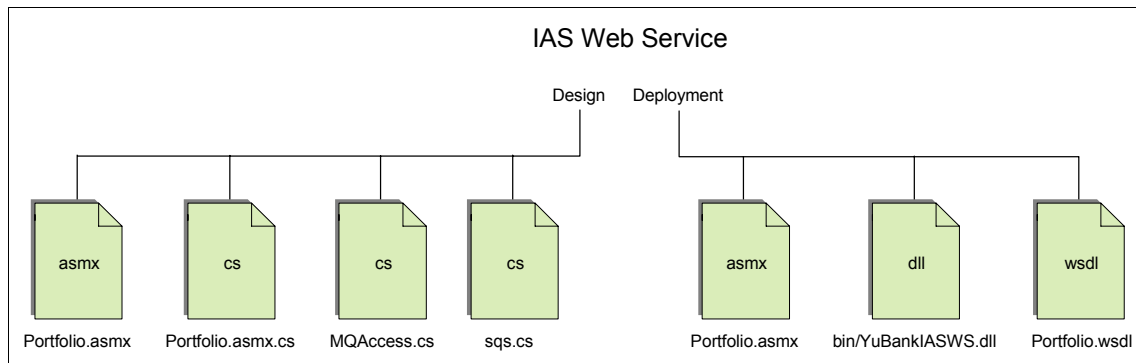


Figure 9-11 IAS Web Service Deployment

The following steps explain how to deploy Portfolio Web Service on the target server:

- ▶ Compile YuBankIASWS Web project in **Release** mode in Microsoft Visual Studio .NET.
- ▶ Copy contents of virtual root (YuBankIASWS sub folder in 'C:\inetpub\wwwroot' including 'bin' subfolder) to a virtual directory on the target server. It is not advisable to copy source files (.cs) over to the deployment server.

- For the entire process involved in this option, visit the following URL:
<http://samples.gotdotnet.com/quickstart/aspplus/doc/deployment.aspx>

Web Setup project

.NET Web Services can also be deployed using a setup application. The following steps explain the process for creating setup application for YuBank IAS Web Service:

1. Create a Web Setup project template of Microsoft Visual Studio .NET.

The following figure shows the Web Setup project wizard for creating the YuBankIASWSSetup project in Microsoft Visual Studio .NET.

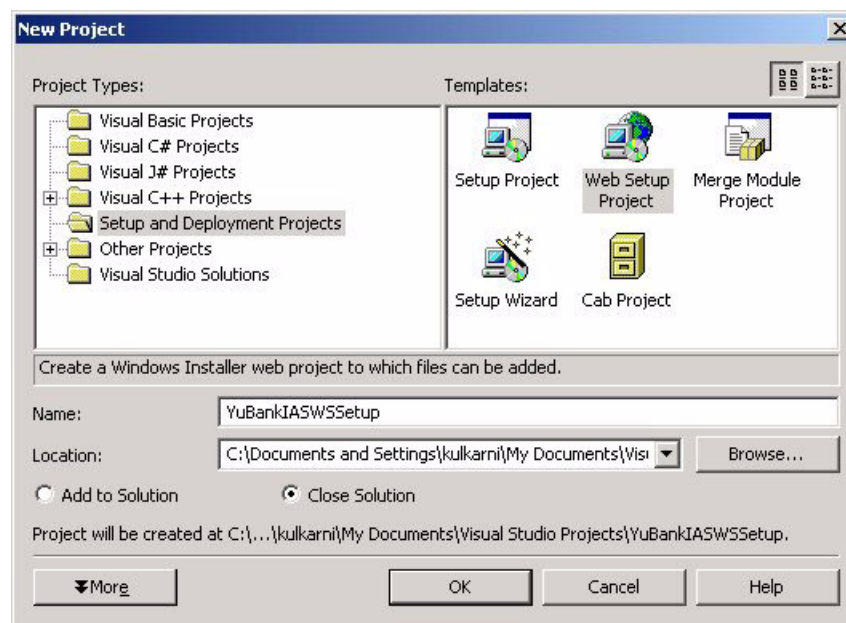


Figure 9-12 Web Setup project wizard in Microsoft Visual Studio .NET

2. Add this setup project to the solution containing the Web Service project.
3. Add the output of YuBankIASWS Web Service to YuBankIASWSSetup project using the following steps:
 - a. Navigate to the solution explorer, right-click the YuBankIASWSSetup project, point to **Add**, and click **Project Output**.
 - b. In the Add Project Output Group window, ensure that YuBankIASWS is selected in the Project box. From the list select **Primary Output**, **Debug Symbols**, and **Content Files** and then click **OK**.

The following figure shows the Add Project Output Group window.

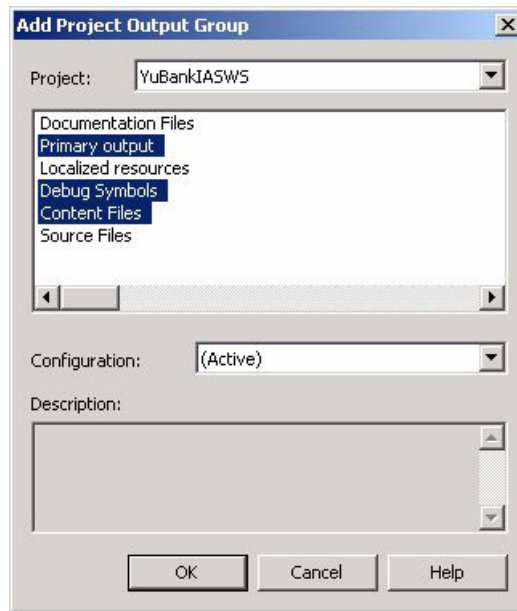


Figure 9-13 Add project output to setup project

4. Build YuBankIASWSSetup project.
5. Execute YuBankIASWSSetup.msi or Setup.exe on the target machine to install YuBankIASWS.

Note: Remember that before executing YuBankIASWSSetup.msi the following software is to be installed on the target server:

- ▶ Microsoft .NET Framework
- ▶ Windows Installer version
- ▶ Internet Information Services (IIS)

9.4.2 WebSphere MQ queue setup and WebSphere MQ transport for SOAP deployment

The following steps are required to implement this functionality:

- ▶ Code the Web Service implementation.
- ▶ Write the client interface code.
- ▶ Generate WSDL defining the implementation of the service.

- ▶ Generate the required proxy code for the client.
- ▶ Prepare a .NET listener for the IAS Web Service.
- ▶ Configure required MQ request and response queues.
- ▶ Configure a trigger monitor for automated listener start-up (optional).

A WebSphere MQ .NET listener is provided with WebSphere MQ transport for SOAP. This is specially prepared to read messages from the queue SOAP.Portfolio and then the target Web Service is invoked through the .NET Web Services framework. The Web Service assesses which companies are most suitable investment and the investment amount for each. This list of recommendations is encapsulated into a SOAP response string which is then dispatched back to the banking client application over the MQ channel. The response message is placed into the queue SOAP.RESPONSE.Portfolio.

9.4.3 BSS Web Application deployment

For instructions on how to deploy the BSS application refer to 7.4.1, “Deploying BSS” on page 134.

9.4.4 Securing the IAS Web Service

This section describes security setup for IAS.

Standard Security

Authentication and authorization services provided by IIS and Windows 2000 form the basis for .NET Web Service security. You can secure the Web Service using authentication and authorization parameters in Web.config file. This file is located in the virtual root folder of your Web Service. (For example, C:\inetpub\wwwroot\YuBankIASWS).

Depending on the requirements the following attribute values for authenticating Web Service users can be used.

Table 9-1 Attribute values for authenticating Web Service users

Value	Description
None	No ASP.NET authentication services are active except for those on IIS authentication services.
Windows	Authentication services attach a WindowsPrincipal to the current request. For more information see ‘System.Security.Principal.WindowsPrincipal’ topic in Microsoft Visual Studio .NET help).

Value	Description
Forms	Developer writes a logon application to collect credentials. Authentication services manage cookies and redirect unauthenticated users to a logon page.
Passport	Web Service uses Microsoft Passport services as an independent authentication provider.

The following example shows how to use authentication mode and authorization in the web.config file. This particular setting in the following example allows only user "BSS" to access the Web Service using the POST method. All other POST and GET requests coming from other users are rejected.

Example 9-28 Authentication and authorization settings in web.config

```

<configuration>
  <system.web>
    <authentication mode="Windows" />
    <authorization>
      <allow VERB="POST" users="BSS" />
      <deny VERB="POST" users="*" />
      <deny VERB="GET" users="*" />
    </authorization>
  </system.web>
</configuration>

```

Customizing authentication using SOAPHeaders

SOAP headers are useful for passing extra information with the request to XML messaging in Web Service. The following steps explain how to use SOAP header to implement custom Web Service authentication and authorization.

1. Create a class that derives from SOAPHeader with properties such as username, password. Declare a public field of that type in your Web Service. This is way it is exposed in the client proxy generated from WSDL.

Example 9-29 Custom SOAP Header class

```

public class IASHeader : SoapHeader
{
    public string Username;
    public string Password;
}

```

2. In your Web Service, just before your Web Method declaration, use [SoapHeader] custom to define a header for your Web Service. During runtime it sets the value of this input header before the method is invoked.

```
[WebMethod]
[SoapHeader("authHeader")]
public string GetAdvice() {
    if(authHeader==null)
        XmlElement er0 = _xmlAdvice.CreateElement("error");
        er0.InnerText="Please supply valid credentials [Authentication Error]";
        _errorElAdvice.AppendChild(er0);
    else
    {
        //business logic
    }
}
```

3. Web Service client sets the header on the proxy class directly before making a Web Service call.

```
private void Button1_Click(object sender, System.EventArgs e)
{
    Portfolio proxyAdvice = new Portfolio();
    proxyAdvice.IASHeader myHeader = new proxyAdvice.IASHeader();
    myHeader.Username = txtId.Text;
    myHeader.Password = txtPass.Text;
    Response.Write(proxyAdvice.GetAdvice());
}
```

9.5 Testing

This scenario is developed as an experimental application for explaining the subject technologies of the redbook. The scope therefore did not permit us to carry out or explain comprehensive testing of this sample application. This section describes basic application and deployment testing that this team found useful in troubleshooting and debugging during the development cycle.

Refer to Chapter 11, “System integration and functional test” on page 227 for basic testing concepts and scenario test cases.

9.5.1 IAS Web Service testing using Microsoft Visual Studio .NET

IAS Web Service contains three major units of operation.

1. Customer profile access and extraction of information from profile data

2. Share Quote System access and share performance analysis
3. Share selection as per customer criterion and advice preparation

For this specific application, the team decided to build the skeleton service first and add units in reverse order.

Attention: Remember that to perform testing in this manner, you need to finalize all message flow and types before commencing the development.

Web Service project testing and input check using default asmx test client in browser

As described in 9.3.1, “.NET Web Service development” on page 173 Microsoft Visual Studio .NET is used to create the template for this Web Service. After adding the account Id and amount information supplied it is tested via a default test application in a browser. The following figure shows the test application generated by .NET Framework and Internet Information Services (IIS).

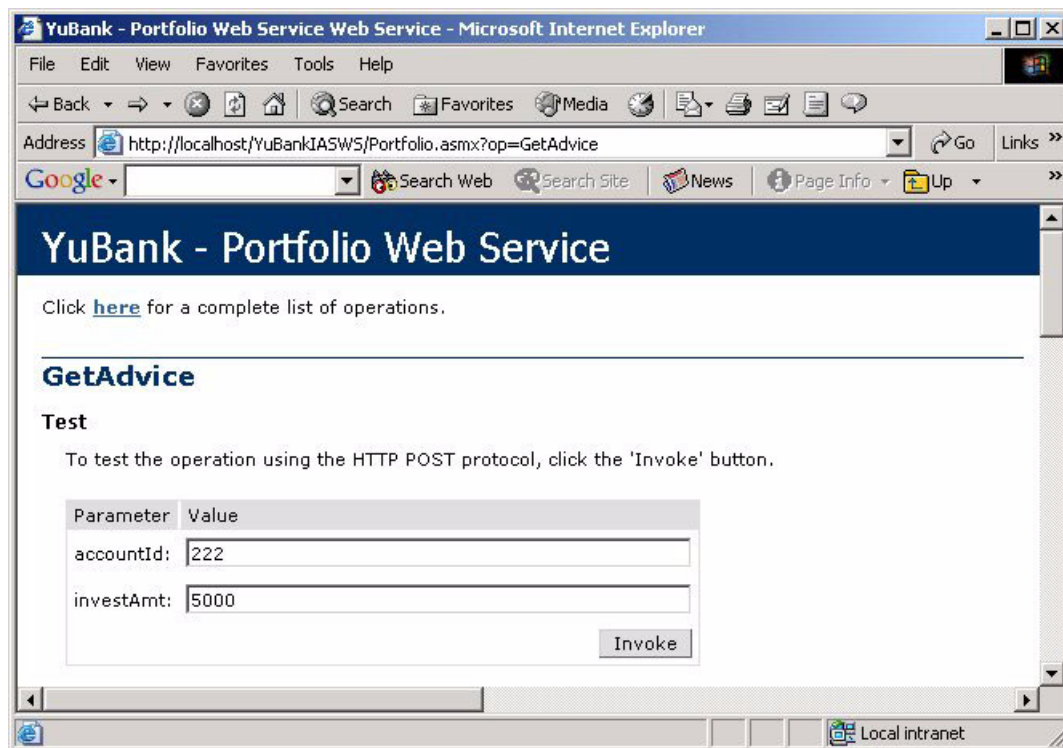


Figure 9-14 Default test application for IAS Web Service

Business logic testing

FindSuitableStocks and PrepareAdvice functions of IAS Web Service contain the major part of the business logic. After this coding is finalized, the sample data for customer profile information and share information is used. This sample information is then replaced by information received from real application (CPS and SQS). This way separates the business logic part of the service from the rest of the communication logic. The following examples show the use of sample XML data that acts as a response from CPS (Example 9-32) and SQS (Example 9-33).

Example 9-32 Business logic testing by using sample CPS data

```
private bool GetCustomerProfile()
{
    XmlDocument xmlCps= new XmlDocument();
    xmlCps.Load(Server.MapPath("MsgQueryResult.xml"));

    if(xmlCps.DocumentElement.HasChildNodes)
    {
        //Conversion of Risk level and Expected return values
    }
}
```

Example 9-33 Business logic testing by using sample SQS data

```
private bool GetStockQuotes ()
{
    //set docs and variables
    xmlSelectedStocks = new XmlDocument();
    rootElStocks=xmlSelectedStocks.CreateElement("selectedstocks");
    xmlSelectedStocks.AppendChild(rootElStocks);

    XmlDocument xmlSQ = new XmlDocument();
    //Call SQS here
    try
    {
        xmlSQ.Load(Server.MapPath("sqsooutput.xml"));
    }
}
```

CPS and SQS communication logic testing

Similarly CPS is tested by keeping sample SQS data. SQS communication testing is performed using real CPS data and functional business logic.

Integration

Once all units are tested individually, end-to-end functionality is tested using default asmx browser application (Figure 9-14 on page 200).

Following this test, the proxy generated by WebSphere MQ Transport for SOAP utility (described in 9.3.4, “BSS client” on page 187) is then tested using a console application.

After successful testing of the WebSphere MQ transport and the IAS Web Service functionality, the IAS proxy was handed over to the BSS application. BSS testing and end-to-end scenario testing is described in 11.2, “System integration” on page 228.

Error handling in the Web Service

Unlike standalone desktop applications, handling exceptions in a Web Service becomes a tricky task. Deployed Web Service do not pass detail information about the error occurred at the server. It's up to the Web Service developer to handle exception gracefully without crashing the service or the process running the services.

In the Portfolio Web Service the following data format for the Web Service return message is used.

Whenever an exception is raised by the application, the corresponding Try-Catch block intercepts it, and writes the error message into the advice XML before returning failure back to the caller.

Function calls are arranged in such a way that the Web Service returns the message immediately back to the Web Service caller without processing the request any further. This pattern avoids causing unnecessary calls (perhaps to the paid services such as SQS) and running into more erroneous stage. The figure below is the error the Web Service returns when there is no match found for the customer.

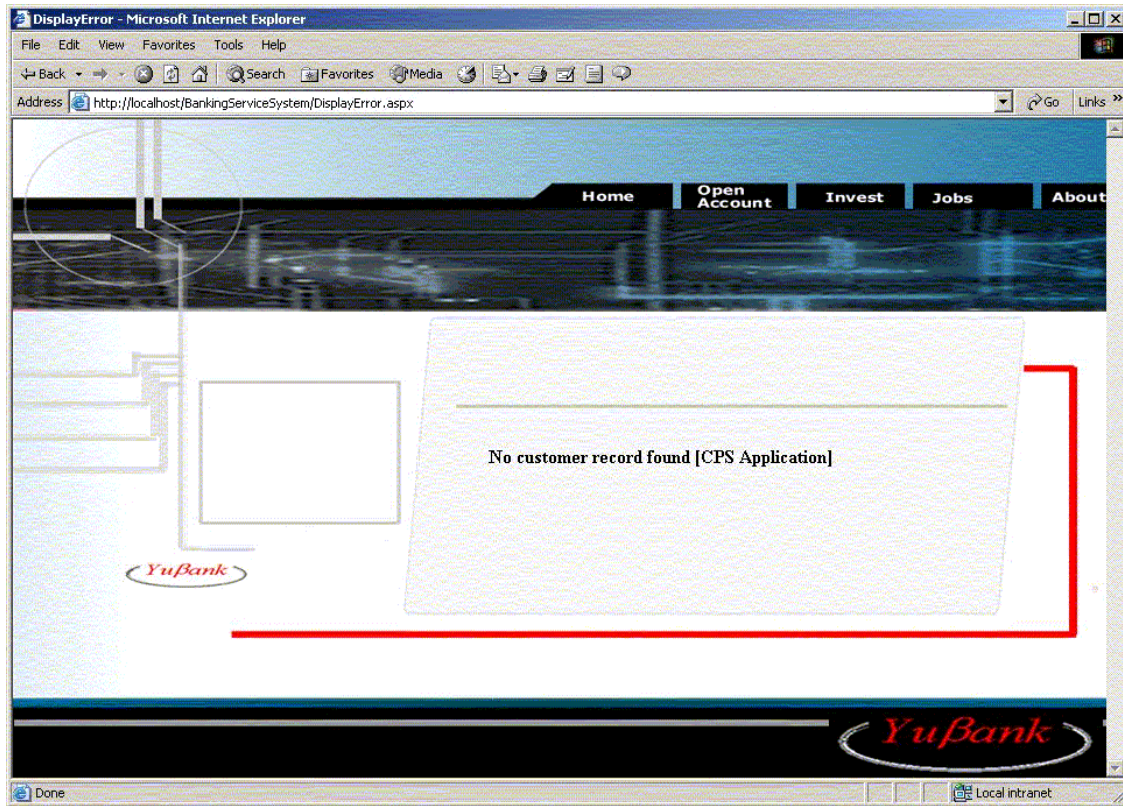


Figure 9-15 Exception handling in Web Service

9.5.2 BSS user interface testing

BSS used simple try catch statements to catch errors and determine situations where the resources, queues, queue manger and so on are unavailable.

BSS sends a request to IAS with the customer account number and investment amount and waits for a response while catching any exceptions that might occur in the process. This is illustrated in the code snippet below:

Example 9-34 Testing in BSS

```
try
{
    //make new instance of Web Service proxy
    sendDetails = new Portfolio();

    //call services and assign advice the value of the response
```

```

string advice = sendDetails.GetAdvice(acctNum, invAmt);
}
catch(MQException ex)
{
System.Console.WriteLine("MQException: compCode: " + ex.CompCode.ToString() + "
Reason: " + ex.Reason.ToString());
}
catch (Exception sysex)
{
System.Console.WriteLine(sysex.ToString());
}
}

```

Below is the final test of the BSS functionality. The investment and amount to be invested are sent as parameters to the Web Service.

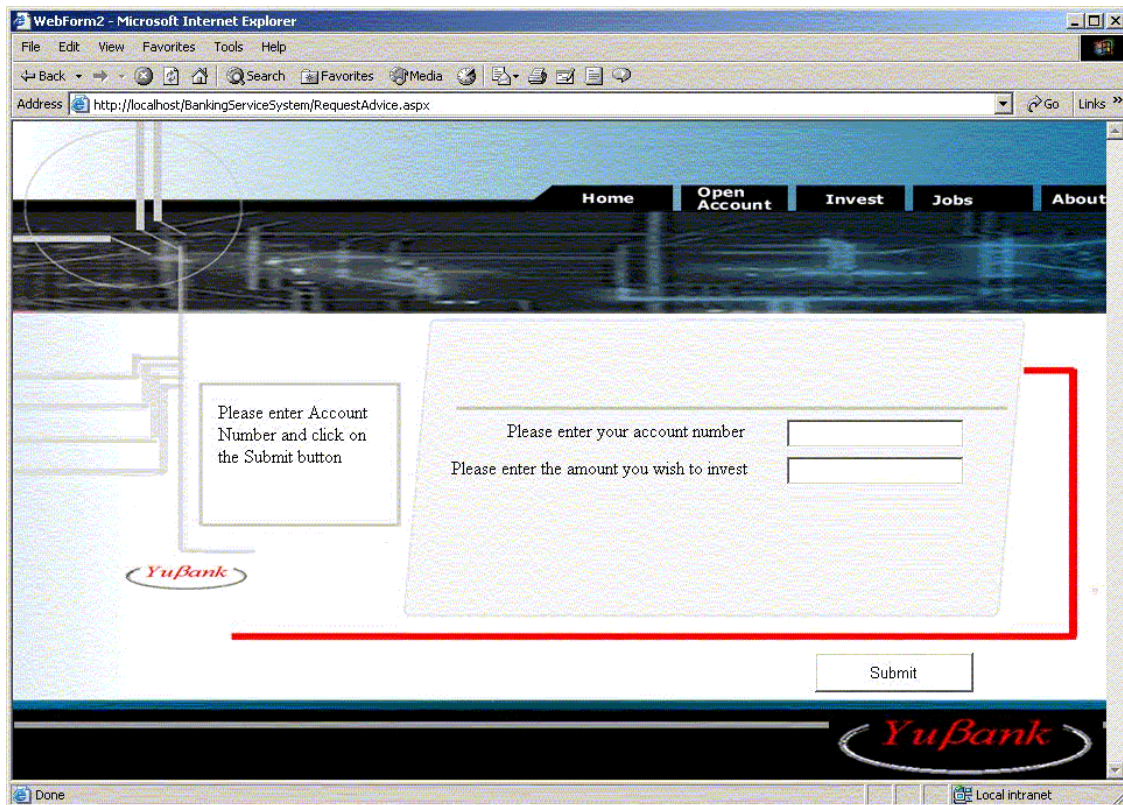


Figure 9-16 BSS request investment amount and customer account number

The Web Service returns the message below:

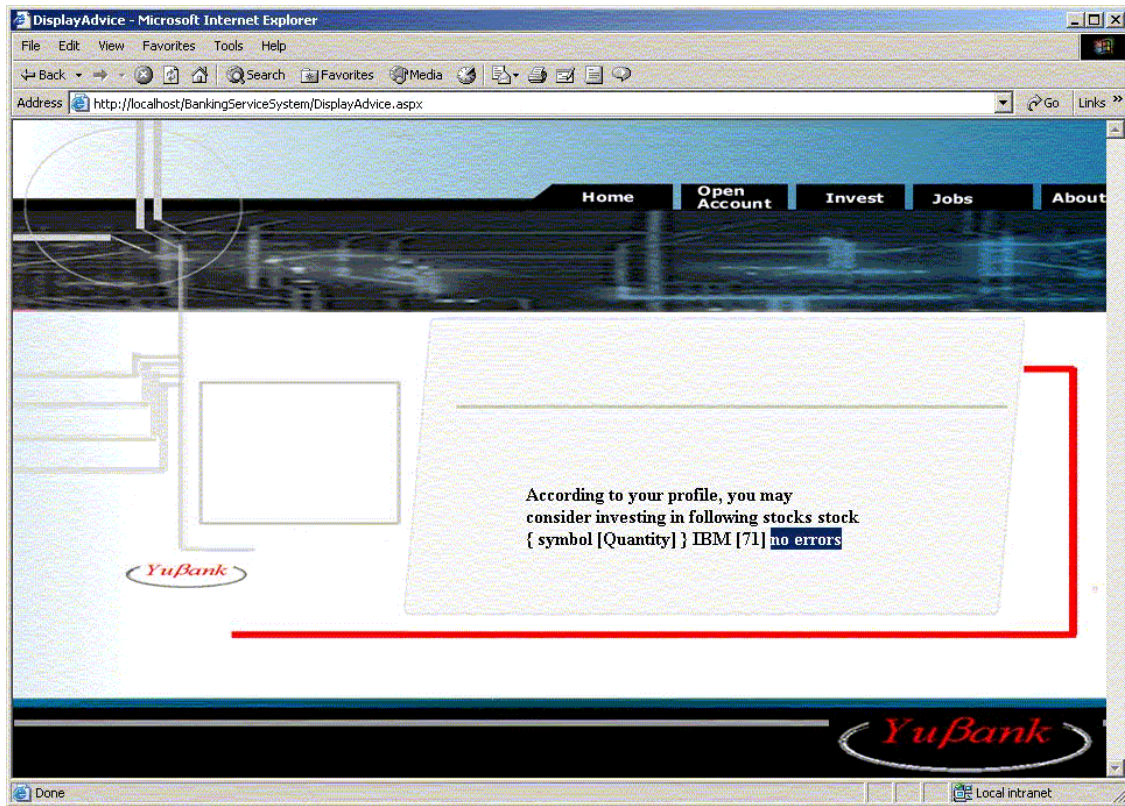


Figure 9-17 Advice from the IAS Web Service



Messaging solution: .NET client to J2EE Web Services using WebSphere MQ SOAP transport

The implementation of the Share Quote System (SQS) is presented in this chapter, which is based on the business model defined in Chapter 4, “Business case scenario” on page 73. This services a request from the Investment Advisory System (IAS) for the current purchase price and historical performance for a given set of shares.

The contents of this chapter are organized as follows:

- ▶ Process overview
- ▶ System context
 - Interface definition
 - Service operation definition
 - XML data format
- ▶ Development
 - Business logic implementation

- Persistent data storage
- WebSphere MQ definition
- Build process
- ▶ Deployment
 - Runtime environment
 - ShareQuote service deployment
- ▶ Testing
 - Calling the service from the IAS client
 - Test result
- ▶ Solution discussion

10.1 Process overview

The Share Quote System is implemented as a J2EE Web Service. It is invoked with an XML string argument, which specifies the stock symbols of the companies for which purchase prices are requested. The SQS service determines the current share prices for the stock and also the return percentages in the past 12 months. These are returned to IAS as a response message from the Web Service. IAS acts as a C# client and uses the proxy class for the ShareQuote service to generate a SOAP formatted request for the service. This SOAP message is transmitted to SQS via WebSphere MQ. A WebSphere MQ transport for SOAP JMS listener is used to retrieve the SOAP message and invoke the SQS service. The SOAP formatted response message is dispatched back via the WebSphere MQ reply queue to the IAS system, which extracts the RPC style response from the SOAP message.

10.2 System context

This section contains the definitions of the interface between the IAS and SQS, service operation in SQS, and XML data format.

10.2.1 Interface definition

The data elements in request and response data to the SQS service are defined in Table 10-1 and Table 10-2, respectively.

Table 10-1 Request data

Data element	Description	Data type	Data sample
name	name of the stock symbol	String	IBM

Table 10-2 Response data

Data element	Description	Data type	Data sample
name	name of the stock symbol	String	IBM
price	Price of the stock in the unit of US Dollar	String	85
m1	Return percentage in the past 1 month	String	1
m3	Return percentage in the past 3 month	String	2
m6	Return percentage in the past 6 month	String	2
m9	Return percentage in the past 9 month	String	6
m12	Return percentage in the past 12 month	String	8

The historical data section provides the stock performance in the periods of 1, 3, 6, 9, and 12 months. Due to the restriction of the XML tag naming format defined in the XML Specification 1.0, the tag names in the history section are in the format of m#. The contents in these elements are return percentages. In case the share symbol is invalid or the share date could not be found, the contents in the price and history elements are NA.

10.2.2 Service operation definition

Access to the SQS to query the stock information is via a RPC mechanism. The signature of the method is defined as follows.

```
public String getQuote (String shareSymbol)
```

The input and output strings are both in XML format, which is defined in the next section.

10.2.3 XML data format

In defining a XML schema, it is necessary to make a decision whether to use an element or attribute to represent the data under the root element. There are pros and cons for either way. As a general rule of thumb, attributes are used to describe the integral characteristics of an element, and typically the values of the attributes are short texts. One of the drawbacks of attributes is that it is hard to expand an attribute to include sub-contents, if the data structure has to be

amplified in the future. Such schema changes potentially require significant more code modifications than an element-based structure. For consistency, both request and response data in SQS use only elements in the data structure.

The request data submitted by the IAS as a .NET client to the SQS as a J2EE Web Service is composed of a list of share symbols in XML format. The data format is defined in the following XML schema.

Example 10-1

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Share">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="name" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="name" type="xsd:string" />
  <xsd:element name="ShareQuoteInquiry">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" ref="Share"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

A sample of the request data is illustrated as follows.

Example 10-2

```
<?xml version="1.0" encoding="UTF-8"?>
<ShareQuoteInquiry>
  <Share>
    <name>IBM</name>
  </Share>
  <Share>
    <name>MSFT</name>
  </Share>
  <Share>
    <name>Microsoft</name>
  </Share>
</ShareQuoteInquiry>
```

A sample of the request data is illustrated as follows.

Example 10-3

```
<?xml version="1.0" encoding="UTF-8"?>
<ShareQuoteInquiry>
  <Share>
    <name>IBM</name>
  </Share>
  <Share>
    <name>MSFT</name>
  </Share>
  <Share>
    <name>Microsoft</name>
  </Share>
</ShareQuoteInquiry>
```

The response data replied from the SQS to the IAS is composed of share prices and historical data in XML format. The data format is defined in the following XML schema.

Example 10-4

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Share">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="name" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="price" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="history" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="ShareQuoteResult">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" ref="Share"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="history">
    <xsd:complexType mixed="true">
      <xsd:choice maxOccurs="unbounded" minOccurs="0">
        <xsd:element ref="m1" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="m3" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="m6" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="m9" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="m12" minOccurs="1" maxOccurs="1"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```

</xsd:element>
<xsd:element name="m1" type="xsd:string"/>
<xsd:element name="m12" type="xsd:string"/>
<xsd:element name="m3" type="xsd:string"/>
<xsd:element name="m6" type="xsd:string"/>
<xsd:element name="m9" type="xsd:string"/>
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="price" type="xsd:string"/>
</xsd:schema>

```

A sample of the response data is illustrated as follows.

Example 10-5

```

<?xml version="1.0" encoding="UTF-8"?>
<ShareQuoteResult>
<Share>
  <name>IBM</name>
  <price>85</price>
  <history>
    <m1>1.5</m1>
    <m3>3.3</m3>
    <m6>6</m6>
    <m9>9.1</m9>
    <m12>11</m12>
  </history>
</Share>
<Share>
  <name>MSFT</name>
  <price>25</price>
  <history>
    <m1>1</m1>
    <m3>-3</m3>
    <m6>5.5</m6>
    <m9>5</m9>
    <m12>8</m12>
  </history>
</Share>
<Share>
  <name>Microsoft</name>
  <price>NA</price>
  <history>NA</history>
</Share>
</ShareQuoteResult>

```

10.3 Development

This section contains the design and implementation of the business logic in SQS. The persistence data storage, queue definitions, and build process are discussed.

10.3.1 Business logic implementation

The `getQuote()` method in `ShareQuote` class in SQS parses the input string using XPath to obtain the contents of all symbolic names, and then retrieves the stock information from the share data source. XQL may be used for sophisticated queries. In this demonstration application, a simple loop search via XPath is implemented. If the symbol is found in the data source, the data element is included in the response. Otherwise, "NA" is used as the content in both price and history sections to indicate an exception. For instance, the stock symbol is invalid, or the financial data for this symbol is unavailable.

There are a variety of ways to construct and parse XML data in a Java program. Coding in both the Simple API for XML (SAX) and the Document Object Model (DOM) can be tedious and bug-prone. JDOM provides a robust, light-weight means of reading and writing XML data without the complex and memory-consumptive options that the DOM and SAX APIs provide. It is a completely natural API for Java development, and it provides a low-cost entry point for manipulating XML in a fairly straightforward manner.

The source code of the `ShareQuote` class is shown below:

Example 10-6

```
import java.io.*;
import java.util.*;
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;
import org.jdom.xpath.*;

/**
 * @version 1.0
 * @author Tony Shan
 */
public class ShareQuote {

    protected static final String SHARE_DATA_SOURCE_FILE_NAME = "ShareDataSource.xml";
    protected static Document shareDataSource;

    static
    {
```

```

        try
        {
            shareDataSource = new SAXBuilder().build(new
FileInputStream(SHARE_DATA_SOURCE_FILE_NAME));
        }
        catch (Exception e)
        {
            e.printStackTrace(System.out);
            System.exit(-1);
        }
    }

    public static String getQuote (String shareSymbol) throws Exception
    {

        Document doc = new SAXBuilder().build(new StringReader(shareSymbol));

        Element root = new Element("ShareQuoteResult");
        Document out = new Document(root);

        List shareDataList = XPath.newInstance("//Share").selectNodes(shareDataSource);

        List symbols = XPath.newInstance("//Share").selectNodes(doc);

        Iterator i = symbols.iterator();

TONY:while (i.hasNext())
    {
        Element symbol = (Element) i.next();

        String name = symbol.getChild("name").getTextTrim();

        Iterator iter = shareDataList.iterator();
        while (iter.hasNext())
        {
            Element shareData = (Element) iter.next();
            if (shareData.getChild("name").getTextTrim().equalsIgnoreCase(name))
            {
                root.addContent((Element) shareData.clone());
                continue TONY;
            }
        }

        Element share = new Element("Share");

        share.addContent(new Element("name").addContent(name));
        share.addContent(new Element("price").addContent("NA"));
        share.addContent(new Element("history").addContent("NA"));
    }

```

```

        root.addContent(share);
    }

    StringWriter strout = new StringWriter();

    new XMLOutputter("\t", true).output(out, strout);

    return strout.toString();
}

public static void main(String[] args) throws Exception
{
    String input =
"<ShareQuoteInquiry><Share><name>IBM</name></Share><Share><name>MSFT</name></Share><Share><name>Microsoft</name></Share></ShareQuoteInquiry>";

    if (args.length > 0)
        input = args[0];

    System.out.println(ShareQuote.getQuote(input));
}
}

```

10.3.2 Persistent storage

The share data source in SQS stores all stock information. The live data is fed to the data source from another channel for periodical updates. The share data source is a native XML database, implemented either by a commercial product like Tamino XML Server and eXcelonís XIS, or an open source database like Xindice and eXist. For simplicity in this demonstration application, a XML file is used in place of the XML database. Sample data is shown as follows.

Example 10-7

```

<?xml version="1.0" encoding="UTF-8"?>
<ShareDataSource>
<Share>
  <name>IBM</name>
  <price>85</price>
  <history>
    <m1>1.5</m1>
    <m3>3.3</m3>
    <m6>6</m6>
  </history>
</Share>

```

```

        <m9>9.1</m9>
        <m12>12</m12>
    </history>
</Share>
<Share>
    <name>MSFT</name>
    <price>25</price>
    <history>
        <m1>1</m1>
        <m3>-3</m3>
        <m6>9</m6>
        <m9>5</m9>
        <m12>8</m12>
    </history>
</Share>
<Share>
    <name>CSCO</name>
    <price>18</price>
    <history>
        <m1>3</m1>
        <m3>1</m3>
        <m6>5</m6>
        <m9>10</m9>
        <m12>20</m12>
    </history>
</Share>
<Share>
    <name>DELL</name>
    <price>33</price>
    <history>
        <m1>3</m1>
        <m3>5</m3>
        <m6>8</m6>
        <m9>10</m9>
        <m12>15</m12>
    </history>
</Share>
<Share>
    <name>AMZN</name>
    <price>38</price>
    <history>
        <m1>2</m1>
        <m3>1</m3>
        <m6>5</m6>
        <m9>1</m9>
        <m12>3</m12>
    </history>
</Share>
<Share>

```

```

    <name>SUN</name>
    <price>10</price>
    <history>
      <m1>1</m1>
      <m3>2</m3>
      <m6>3</m6>
      <m9>4</m9>
      <m12>5</m12>
    </history>
  </Share>
  <Share>
    <name>YH00</name>
    <price>35</price>
    <history>
      <m1>8</m1>
      <m3>7</m3>
      <m6>6</m6>
      <m9>5</m9>
      <m12>4</m12>
    </history>
  </Share>
  <Share>
    <name>ORCL</name>
    <price>12</price>
    <history>
      <m1>1</m1>
      <m3>3</m3>
      <m6>0</m6>
      <m9>9</m9>
      <m12>6</m12>
    </history>
  </Share>
  <Share>
    <name>PEP</name>
    <price>44</price>
    <history>
      <m1>5</m1>
      <m3>8</m3>
      <m6>9</m6>
      <m9>12</m9>
      <m12>15</m12>
    </history>
  </Share>
  <Share>
    <name>GTW</name>
    <price>4</price>
    <history>
      <m1>3</m1>
      <m3>8</m3>

```

```

    <m6>1</m6>
    <m9>1</m9>
    <m12>5</m12>
  </history>
</Share>
</ShareDataSource>

```

10.3.3 WebSphere MQ definition

The IAS System forms a list of target companies for which share values and histories are required. This list is encapsulated into a request message using WebSphere MQ transport for SOAP and this message is then placed onto a local transmission queue. The message is sent over WebSphere MQ to the Share Quote System where it arrives in the queue SOAP.ShareQuote. The message queue definitions are illustrated in Figure 10-1.

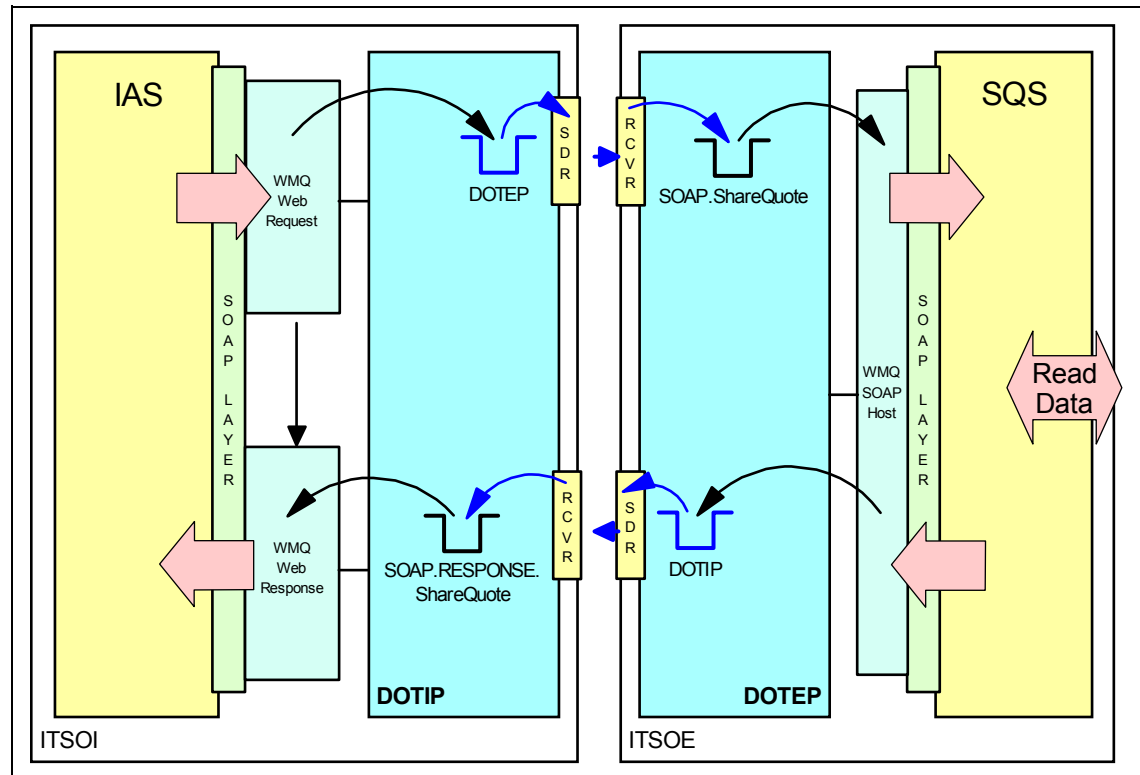


Figure 10-1 Message queue definitions

A JMS listener is provided with WebSphere MQ transport for SOAP which reads messages from a request queue and invokes the target service through the .NET

Web Services framework. The deployment process generates a DOS BAT file which calls this listener specifying the correct queue manager, in this case DOTIP, and the request queue as SOAP.ShareQuote. The Web Service determines the value of the required shares and constructs a response message. This is passed to the local transmission queue on the SQS system from where it is dispatched by WebSphere MQ back to the IAS system. This response placed in the queue SOAP.RESPONSE.ShareQuote from which the BSS application subsequently reads. Both the request and response messages are specifically tagged as non-persistent. This is because in the event of an interruption to the target SQS queue manager there may well have been a time lapse which renders the previously determined share prices invalid. Rather than use WebSphere MQ persistency here it is more accurate if the Banking system simply reposts the request.

10.3.4 Adding external classes to the CLASSPATH

If a service application uses external classes or libraries, the classpath must be set to include these dependent packages. In this case, the jar files in JDOM package have to be part of the classpath variable. Refer to 3.9.10, “Service code use of external classes” on page 59 for more detail.

10.4 Deployment

This section contains the specification of the runtime environment, and the deployment of the SQS Web Service.

10.4.1 Runtime environment

The following components and packages are required in the runtime environment.

- ▶ WebSphere MQ
- ▶ WebSphere MQ WebSphere MQ Transport for SOAP (SupportPac MA0R)
- ▶ Java Virtual Machine (JVM)
- ▶ .NET Framework redistributable
- ▶ .NET Framework SDK 1.1
- ▶ Service application binary (Java classes and libraries)

Since the server system does not contain Microsoft Visual Studio .NET, so it is necessary to install the Microsoft .NET Framework (V1.1) and the Microsoft .NET Framework SDK (V1.1). Note that it is necessary to install the framework first to

prevent obvious failure message when attempting to install the SDK even when it fails to install completely.

Although in this instance configuring a J2EE Web Service, WebSphere MQ transport for SOAP requires access to the .NET Framework because it uses the Global Assembly Cache and gacutil to register the DLLs that it makes use of.

10.4.2 ShareQuote service deployment

The ShareQuote service application is first developed as a simple standalone program. Once this functions correctly it is then turned it into a Web Service. This involves the use of the WebSphere MQ transport for SOAP deployment utility. Deployment is performed on the SQS system, which is the system on which the service is destined to be invoked.

IAS is itself a Web Service and services requests from the Banking System clients. As this is a non trivial application it necessitated testing our SQS service from a simple C# client. This is written from one of the sample programs provided in WebSphere MQ transport for SOAP (SQCS2Axis.cs). This client is built on the IAS system ITSOL. Having proved that the service could be invoked correctly from this simple client, the service is integrated into the main IAS application.

On the SQS system is placed the source code for the service in a new sub-directory and then the service is deployed with the deployWMQService utility:

Example 10-8

```
C:\j2eews\ma0r\tony_040703>\j2eews\Ma0raaaG0\ma0r\bin\setcp

C:\j2eews\ma0r\tony_040703>setcp

C:\j2eews\ma0r\tony_040703>set classpath=classes;C:\j2eews\ma0r\
lib\ma0r.jar;C:\j2eews\ma0r\lib\axis.jar;C:\j2eews\ma0r\
lib\jaxrpc.jar;C:\j2eews\ma0r\lib\saa.jar;C:\j2eews\ma0r\lib\
commons-logging.jar;C:\j2eews\ma0r\lib\commons-discovery.jar;C:\j2ee
ws\ma0r\lib\wsdl4j.jar;C:\j2eews\ma0r\lib\xerces.jar;C:\j2ee
ws\ma0r\lib\servlet.jar;C:\j2eews\ma0r\lib\com.ibm.mq.jar;C:
\j2eews\ma0r\lib\com.ibm.mqjms.jar;C:\j2eews\ma0r\lib\connec
tor.jar;C:\j2eews\ma0r\lib\jms.jar;C:\j2eews\ma0r\lib\jta.ja
r;C:\j2eews\ma0r\lib\jndi.jar;C:\j2eews\ma0r\lib\ldap.jar;C:
\j2eews\ma0r\lib\providerutil.jar;C:\j2eews\ma0r\lib\com.ibm
.mqbind.jar;lib\jdom.jar;lib\saxpath.jar;lib\jaxen-core.jar;lib\jaxen-jdom.jar;

C:\j2eews\ma0r\tony_040703>deployWMQService
-m DOTE -f ShareQuote.java
Package name: DefaultNamespace
```

```

Compiling service code...
Generating WSDL...
Serviceport: ShareQuote_Wmq
java org.apache.axis.wsdl.Java2WSDL --input helpers\ShareQuote_Wmq.wsdl --output
  helpers\ShareQuote_Wmq.wsdl --namespace ShareQuote_Wmq --location wmq:SOAP.Shar
eQuote@DOTE?connectQueueManager=DOTE? --bindingName ShareQuoteBindingSoap --ser
vicePortName ShareQuote_Wmq ShareQuote
Generating and deploying server wsdd file...
Target dir: C:\j2eews\ma0r\tony_040703\helpers\
Patching deploy.wsdd...
Patching undeploy.wsdd...
Removing temp.server directory...
Preparing listener...
Configuring MQ...
Generate and compile proxy code...
java com.ibm.mq.ma0r.tools.RunWSDL2Java --timeout -1 --output helpers -p Default
Namespace helpers\ShareQuote_Wmq.wsdl

C:\j2eews\ma0r\tony_040703>

```

The JMS listener is then started with the script automatically generated by the deployment utility in the helpers directory:

Example 10-9

```

C:\j2eews\ma0r\tony_040703\helpers>listen_SOAP.ShareQuote

C:\j2eews\ma0r\tony_040703\helpers>rem - generated by deployWMQService
.java at 15-Jul-03 10:16:12

C:\j2eews\ma0r\tony_040703\helpers>call C:\j2eews\ma0r\bin\s
etcp.bat

C:\j2eews\ma0r\tony_040703\helpers>call ..\setcp

C:\j2eews\ma0r\tony_040703\helpers>set classpath=classes;C:\j2eews\
ma0r\lib\ma0r.jar;C:\j2eews\ma0r\lib\axis.jar;C:\j2eews\
ma0r\lib\jaxrpc.jar;C:\j2eews\ma0r\lib\saa.jar;C:\j2eews\
ma0r\lib\commons-logging.jar;C:\j2eews\ma0r\lib\commons-discovery.jar
;C:\j2eews\ma0r\lib\wsdl4j.jar;C:\j2eews\ma0r\lib\xerces.jar
;C:\j2eews\ma0r\lib\servlet.jar;C:\j2eews\ma0r\lib\com.ibm.m
q.jar;C:\j2eews\ma0r\lib\com.ibm.mqjms.jar;C:\j2eews\ma0r\li
b\connector.jar;C:\j2eews\ma0r\lib\jms.jar;C:\j2eews\ma0r\li
b\jta.jar;C:\j2eews\ma0r\lib\jndi.jar;C:\j2eews\ma0r\lib\lda
p.jar;C:\j2eews\ma0r\lib\providerutil.jar;C:\j2eews\ma0r\lib
\com.ibm.mqbind.jar;lib\jdom.jar;lib\saxpath.jar;lib\jaxen-core.jar;lib\jaxen-jd
om.jar;

C:\j2eews\ma0r\tony_040703\helpers>cd /d C:\j2eews\ma0r\tony

```

_040703

```
C:\j2eews\ma0r\tony_040703>java com.ibm.axis.transport.wmq.SimpleJMSListener -u wmq:SOAP.ShareQuote@DOTEP?connectQueueManager=DOTEP -T 1
```

Starting Axis JMS listener.

listeners initialised.

```
Parameters: -u wmq:SOAP.ShareQuote@DOTEP?connectQueueManager=DOTEP -c . -f null
-s null -a false -T 1
```

```
*****
* Hit Enter to stop the listener and close this window *
*****
```

10.5 Testing

This section contains the unit test for the SQS Web Service.

10.5.1 Calling the service from the IAS client

Our IAS client application calls the ShareQuote service from the GetStockQuotes method. This method is located in the Portfolio class in the file Portfolio.asmx.cs. The invocation of the service is shown below:

Example 10-10 Invocation of the ShareQuote service from the Investment Advisory System

```
sqs proxySqs = new sqs();
tRes = proxySqs.GetQuotes();
```

The proxy object is not instantiated directly at this level but rather a separate class called sqs is developed to perform this task. This is done to separate the transport specific code from the logic of the business application. The key points in this class are the constructor and also the method GetQuotes(). These are shown below:

Example 10-11

```
public sqs()
{
    MQSOAP.MQWebRequest.Register();
}

public string GetQuotes()
{
```

```

string strSqsResponse = "";
XmlDocument xmlSqsRequest = new XmlDocument();
xmlSqsRequest.Load(@"C:\inetpub\wwwroot\YuBank\IASWS\requestSqs.xml");

try
{
    ShareQuoteService shareobj = new ShareQuoteService();

    shareobj.Url =
"wmq:SOAP.ShareQuote@DOTEP?connectQueueManager=DOTIP,replyToQueue=SOAP.RESPONSE.ShareQuote";

    strSqsResponse = shareobj.getQuote(xmlSqsRequest.OuterXml);
    return strSqsResponse ;
}
catch (System.Exception e)
{
    return e.GetType().ToString();
}
}

```

The WebSphere MQ transport must be registered in the client. In this case, the registration takes place in the sqs constructor. The GetQuotes() method then sets the required WebSphere MQ URI and then instantiates the proxy class ShareQuoteService. It then calls the method getQuote() from the proxy object to begin the process of invoking the service.

The proxy by default sets the URI to:

```
wmq:SOAP.Portfolio@DOTIP?connectQueueManager=DOTIP
```

This needs to be overridden to specify the correct request and response queues and also the local queue manager through which WebSphere MQ should communicate with the remote system. Consequently the URI in our sqs constructor is set to:

```
wmq:SOAP.ShareQuote@DOTEP?connectQueueManager=DOTIP,replyToQueue=SOAP.RESPONSE.ShareQuote
```

This URI specifies the target request queue is SOAP.ShareQuote on remote queue manager DOTEP, that the client should access the queue manager DOTEP via the local queue manager DOTIP and that the response queue is SOAP.RESPONSE.ShareQuote. The WebSphere MQ configuration scripts which sets up our environment (refer to “WebSphere MQ setup” on page 112 and Appendix A, “Scripts, source code and test data for YuBank” on page 317) have

already created the transmit and receive channels DOTE.P.TO.DOTIP and DOTIP.TO.DOTE.P that is needed on both the IAS and SQS system to establish communication with WebSphere MQ between the two systems.

The client call to the SQS system is integrated into the IAS system. Because this contains several classes and is more complicated than the simple demonstration presented earlier it is decided to build using Microsoft Visual Studio .NET rather than from the command line. This technique is documented in Chapter 8, "Messaging solution: .NET application to J2EE application" on page 139.

The proxy code in the helpers directory is generated by the deployment utility. It is only necessary to copy this code from the machine the service is deployed on back to the client system. Alternatively the deployment utility may be re-run on the client system, but this results in the request queue also being generated on the client which, although not a problem, is superfluous to requirements.

10.5.2 Test result

The demonstration program can now be invoked from the IAS client system:

Example 10-12

```
C:\$user\mon\ma0r\tony_040603>sqlclient IBM wmq:SOAP.ShareQuote@D0
TEP?connectQueueManager=DOTIP,replyToQueue=SOAP.RESPONSE.ShareQuote
XML: <?xml version="1.0" encoding="utf-16"?><ShareQuoteInquiry><Share><name>IBM<
/name></Share></ShareQuoteInquiry>
Using server bindings.
Sharequote reply is:
<?xml version="1.0" encoding="UTF-8"?>
<ShareQuoteResult>
  <Share>

    <name>IBM</name>

    <price>85</price>

    <history>

      <m1>1.5</m1>

      <m3>3.3</m3>

      <m6>6</m6>

      <m9>9.1</m9>

      <m12>12</m12>
```

```
</history>

</Share>
</ShareQuoteResult>

C:\$user\mon\ma0r\tony_040603>
```

The most important thing to note here is the URI. This specifies that the target request queue is SOAP.ShareQuote on remote queue manager DOTEPI, that the client should access the queue manager DOTEPI via the local queue manager DOTIP and that the response queue is SOAP.RESPONSE.ShareQuote. The MQ configuration scripts with which we set up our environment (refer to Chapter 5, “Solution design” on page 79) had already created the transmit and receive channels DOTEPI.TO.DOTIP and DOTIP.TO.DOTEPI that we needed on both the IAS and SQS system to establish communication with WebSphere MQ between the two systems.

10.6 Solution discussion

This chapter demonstrates the usage scenario of WebSphere MQ as the SOAP transport layer for the Web Service invocation from a .NET application to a Java application. The nature of this service invocation is a Remote Procedure Call (RPC) mechanism. This approach takes advantage of the asynchronous feature in the message queuing.

The service client in this demonstration is an application written in C#. It is obvious that applications programmed in other languages may consume the service, as long as these applications have the API access to the WebSphere MQ, and handle the SOAP marshalling and unmarshalling with a prebuilt package or their own codes. For example, a C/C++ program may use the API package provided by IBM for WebSphere MQ access to consume a Web Service implemented in SQS.

In contrast to SOAP over HTTP, this usage scenario requires additional queue setup and configuration. On the other hand, it enables the service invocations to be non-blocking, providing features like one-way call and notification for long-lived processes in an asynchronous mode.



System integration and functional test

This chapter describes the integration of the components and subsystems developed in the foregoing chapters and the end-to-end functional test.

11.1 Scope and objectives

At a high level, the system integration and functional test intends to prove that:

- ▶ The functionality delivered is as specified in the Business case scenario requirement.
- ▶ The software supports the intended business functions and achieves the standards required by the bank.
- ▶ The software delivered interfaces correctly with major Web browsers, including Microsoft Internet Explorer, Netscape Navigator, and Mozilla.
- ▶ The software implements the minimum exception handling.

Due to time limitation, the test plan described in this chapter is simplified for demonstration purpose only. The formal Quality Management (QM) process and Quality of Service (QoS) verification are deliberately streamlined. The following tests are excluded in this demonstration, but are highly recommended in the system development for your organization:

- ▶ Unit profiling test
- ▶ Regression test
- ▶ Performance test
- ▶ Stress test
- ▶ Penetration test
- ▶ Usability test
- ▶ Multi-user test
- ▶ Storage and network negative test
- ▶ Disaster recovery test
- ▶ Failover test
- ▶ Load-balancing test
- ▶ Compliance test
- ▶ SLA measurement test
- ▶ Operation acceptance test

11.2 System integration

The system integration is to ensure that all areas of the system, interface with each other correctly and that there are no gaps in the data flow. The integration test proves that the system works as an integrated unit when all the fixes are complete.

11.2.1 Runtime environment

The integration runtime environment is composed of all the subsystems developed in the previous chapters.

- ▶ The software installation and setup are documented in Chapter 6, “Environment setup” on page 89.
- ▶ The server configuration and setup for each subsystem are described in the respective chapters:
 - Chapter 7, “Messaging solution: .NET application to .NET application” on page 115
 - Chapter 8, “Messaging solution: .NET application to J2EE application” on page 139
 - Chapter 9, “Messaging solution: .NET client to .NET Web Services using WebSphere MQ SOAP transport” on page 163
 - Chapter 10, “Messaging solution: .NET client to J2EE Web Services using WebSphere MQ SOAP transport” on page 207

The inter-system communications are all based on WebSphere MQ Version 5.3. The queue managers, channels, and queues are pre-tested to ensure that the infrastructure is ready for integration.

11.2.2 Test data

Test data need to be pre-loaded to the subsystems for the integration testing. The Credit Check System (CCS) and Share Quote System (SQS) have test data loaded and unit-tested. The details of the test data are enclosed in Appendix A, “Scripts, source code and test data for YuBank” on page 317.

11.2.3 System build and deployment

The builds of all software components and packages are deployed to the respective subsystems, including the dependent libraries and packages. The details of the deployables and deployment are provided in each individual chapter:

- ▶ Chapter 7, “Messaging solution: .NET application to .NET application” on page 115
- ▶ Chapter 8, “Messaging solution: .NET application to J2EE application” on page 139
- ▶ Chapter 9, “Messaging solution: .NET client to .NET Web Services using WebSphere MQ SOAP transport” on page 163

- ▶ Chapter 10, “Messaging solution: .NET client to J2EE Web Services using WebSphere MQ SOAP transport” on page 207

11.2.4 System startup

The subsystems must be started and verified in a particular sequence, to ensure proper inter-system communications to be initiated correctly. All WebSphere MQ queue managers need to be started. All sender channels need to be started and in running state.

11.3 Functional test

The functional testing is to ensure that each element of the application meets the functional requirements of the business case scenario as outlined in Chapter 4, “Business case scenario” on page 73. It includes validation testing, which is intensive testing of the new front-end fields (HTML pages); layout standards; valid, invalid and limit data input; screen and field look and appearance, and overall consistency with the rest of the application. It also includes specific functional testing; low-level tests which aim to test the individual processes and data flows.

The tests applied, the data processed, the testing coverage and the expected results are documented in the following sections.

11.3.1 Entrance and exit criteria

The entrance criteria should be fulfilled before the functional test can commence.

- ▶ All developed code must be unit tested. Unit and link testing must be completed.
- ▶ All test hardware and environments must be in place, and free for functional test use.
- ▶ The acceptance tests must be completed, with a pass rate of not less than a certain percentage, for instance, 90%.

The exit criteria detailed below must be achieved before the final acceptance hand over.

- ▶ All high priority errors must be fixed and tested.
- ▶ If any medium or low priority errors are outstanding, the implementation risk must be acceptable by the business requirement.

11.3.2 Use case 1: Account opening

This section describes the test plan for the account opening use case.

General data flow

The diagram below shows the general data flow in use case 1 of the business case scenario.

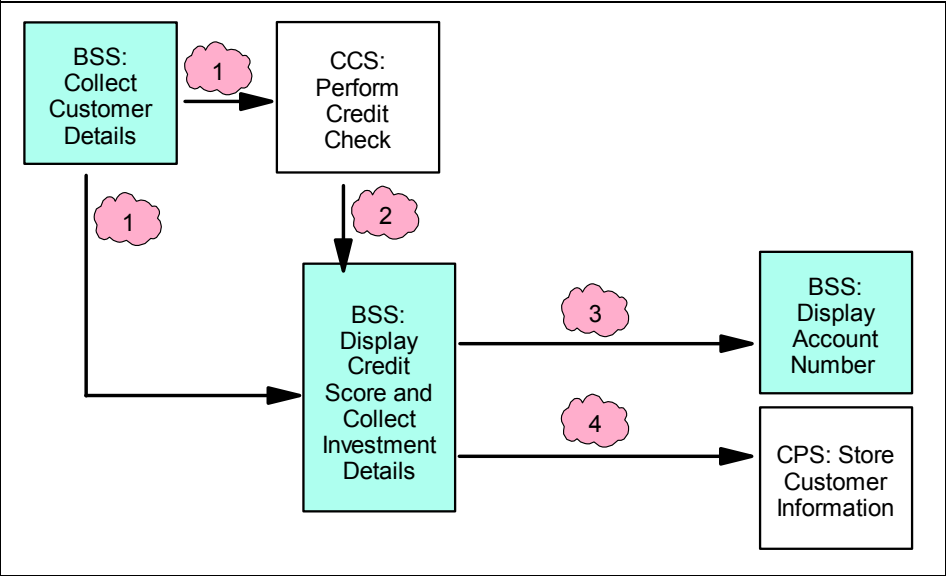


Figure 11-1 General data flow in use case 1

Number	Data
1	Customer details: Customer name, address and date of birth
2	Credit score and comments
3	Account number
4	Customer details, credit score and investment details: investment amount, investment period, return expectation, risk level and so on

Test Case 1: Successful account opening - full qualification

Step1: On starting the process, the customer is faced with the bank’s home page.

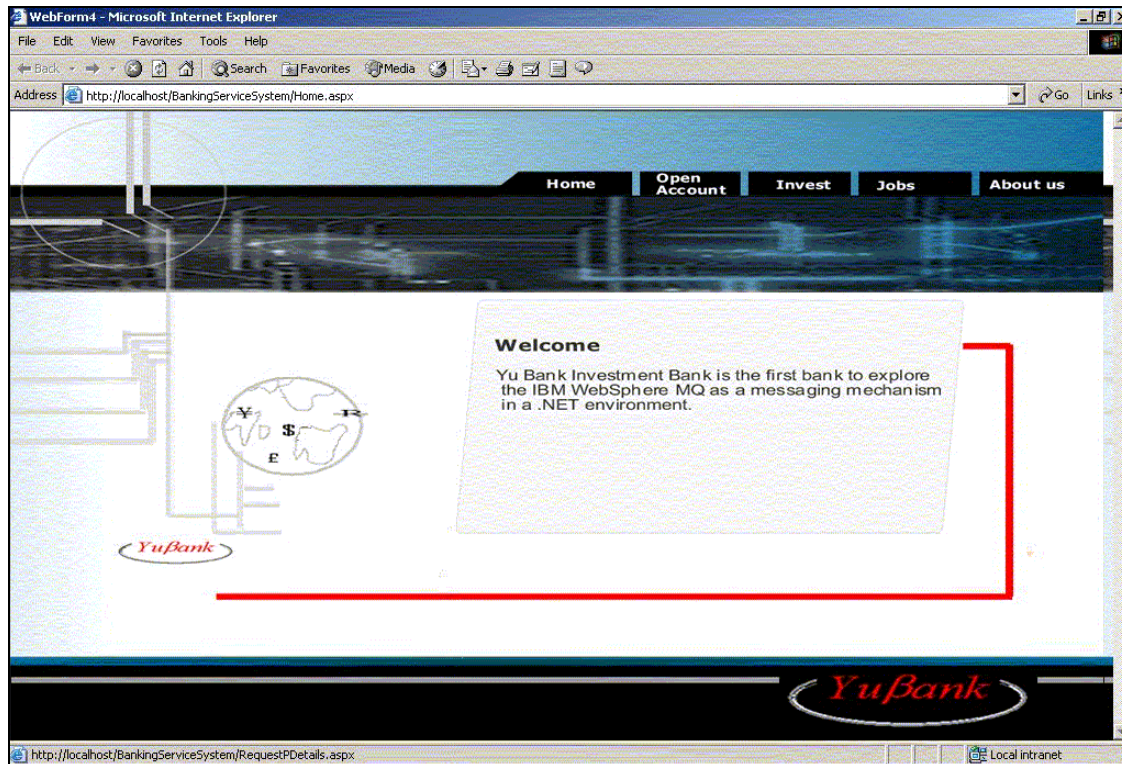


Figure 11-2 Home page

Step 2: The customer clicks the **Open Account** tab and is faced with a page requesting the customer details, the customer unique ID, type of ID, customer name, address and date of birth. Click the **Submit Personal Details** button as shown below:

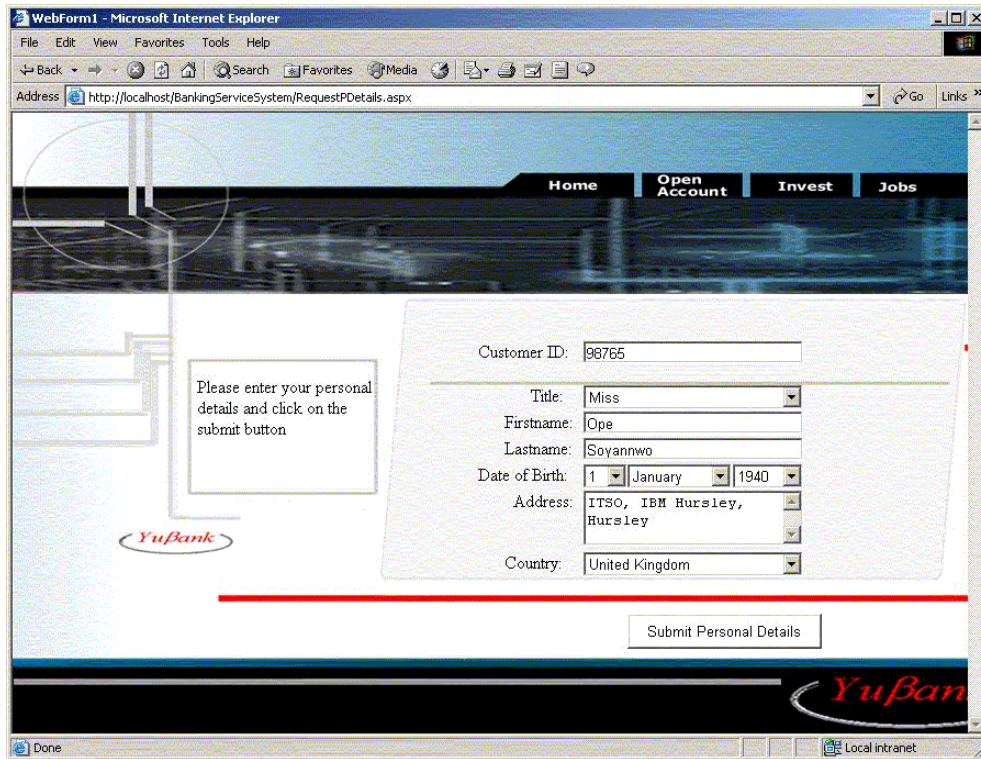


Figure 11-3 Open account: collect customer details

Step 3: On clicking the Submit Personal Details button, a credit check is done on the customer. The customer is faced with a page which displays the credit check results and requests investment details such as the initial deposit, risk level, existing assets, return expectation, family income, current debts as well as investment period. Enter investment details and click the **Submit Investment Details** button as shown below:

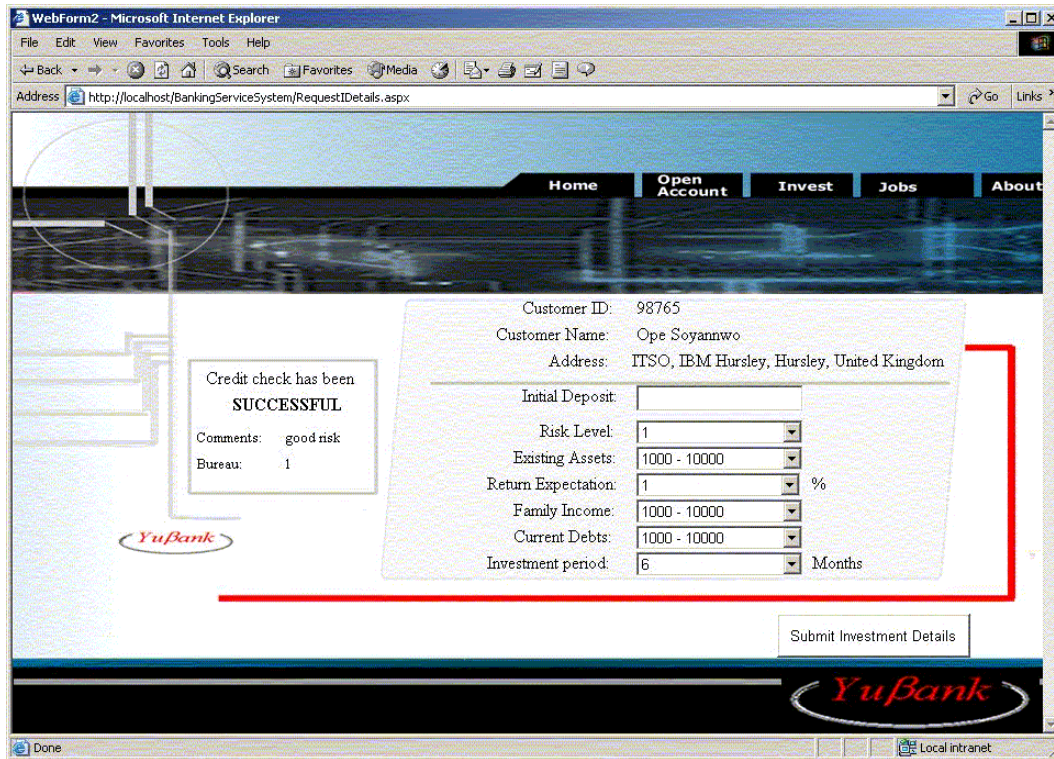


Figure 11-4 Open account: collect investment details

Step 4: On clicking the Submit Investment Details, the investment details, credit score and customer details are sent to an application for storing. The customer is given an account number for future investment application as shown below:

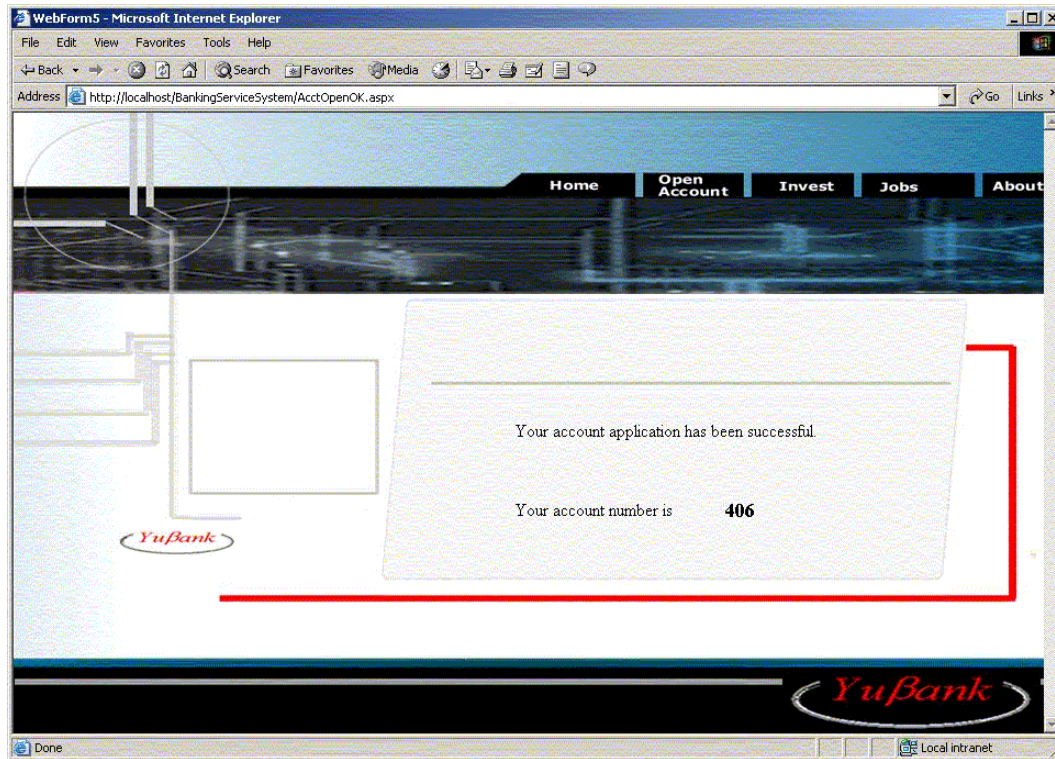


Figure 11-5 Account opened successfully

This test case verifies the normal path for the processing. A new investment account is created, along with a new customer profile, after the customer passes the credit check.

Test Case 2: Unsuccessful account opening - unqualified credit score

Repeat Step 1 and Step 2 from Test Case 1: Successful account opening above.

Step 3: On clicking the Submit Personal details button, a credit check is done on the customer. If the customer's credit score does not meet the bank requirement, the account opening process is terminated and the customer is faced with a page which displays the process result.

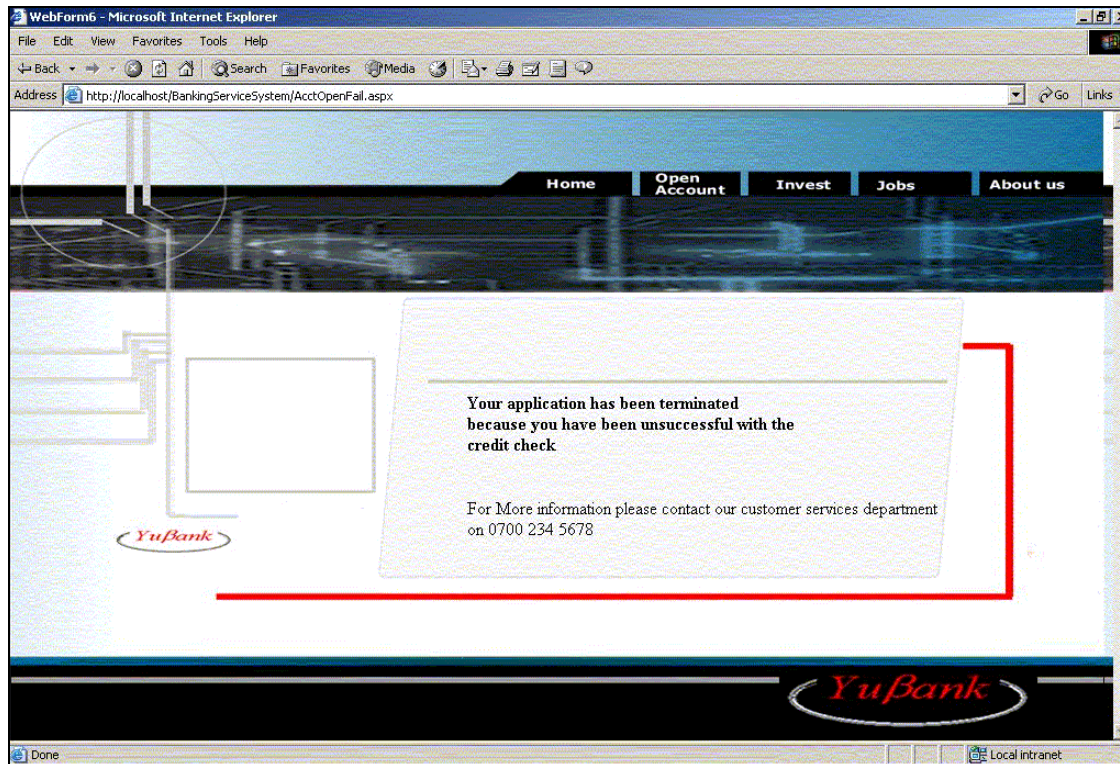


Figure 11-6 Account opening terminated

This test case verifies an alternative path for the processing. For customers who do not meet the credit requirement set by the bank, the account opening requests are declined.

Test Case 3: Negative tests - exception recovery and system unavailability

Account Opening service unavailable

Repeat Step 1 from Test Case 1: Successful account opening above.

Step 2: The customer clicks the **Open Account** tab. If the account opening service is unavailable, as a result of an inaccessible queue manager, the customer is faced with a page that looks like the one below:

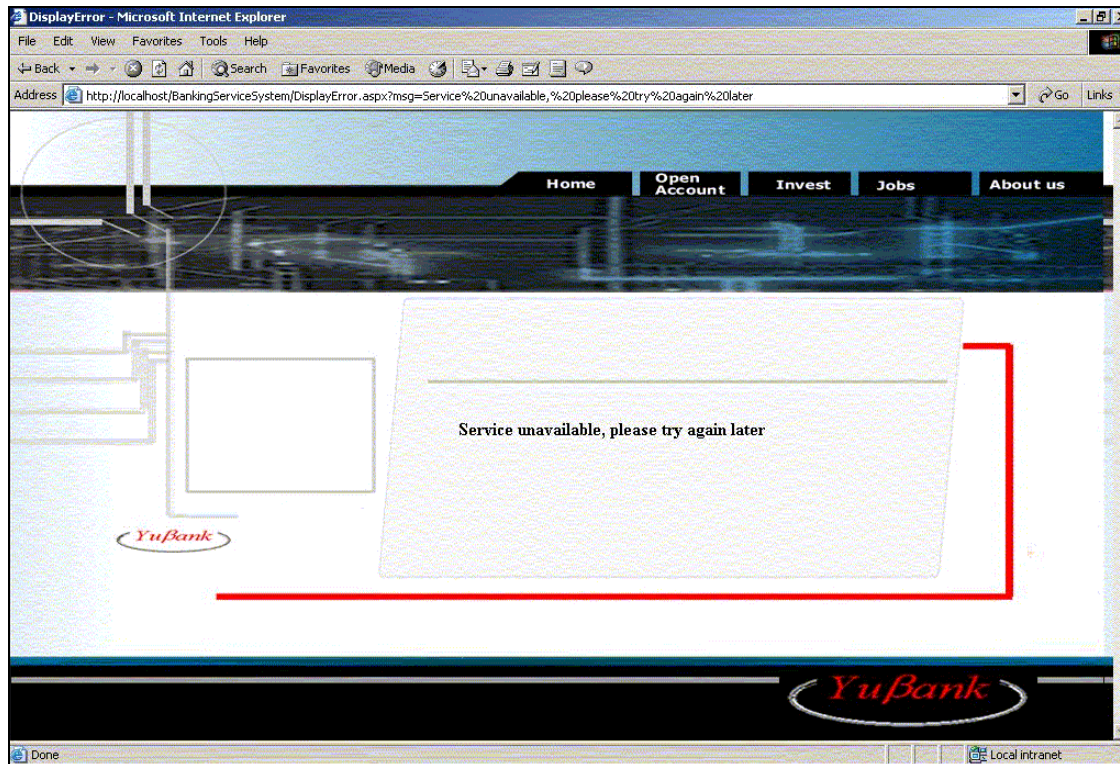


Figure 11-7 Service unavailable

Credit check service unavailable

Repeat Step 1 and Step 2 from Test Case 1: Successful account opening above.

Step 3: The customer clicks on the **Submit Personal details** button. If the credit check application is unavailable, as a result of an inaccessible queues, the customer gets feedback shown below:

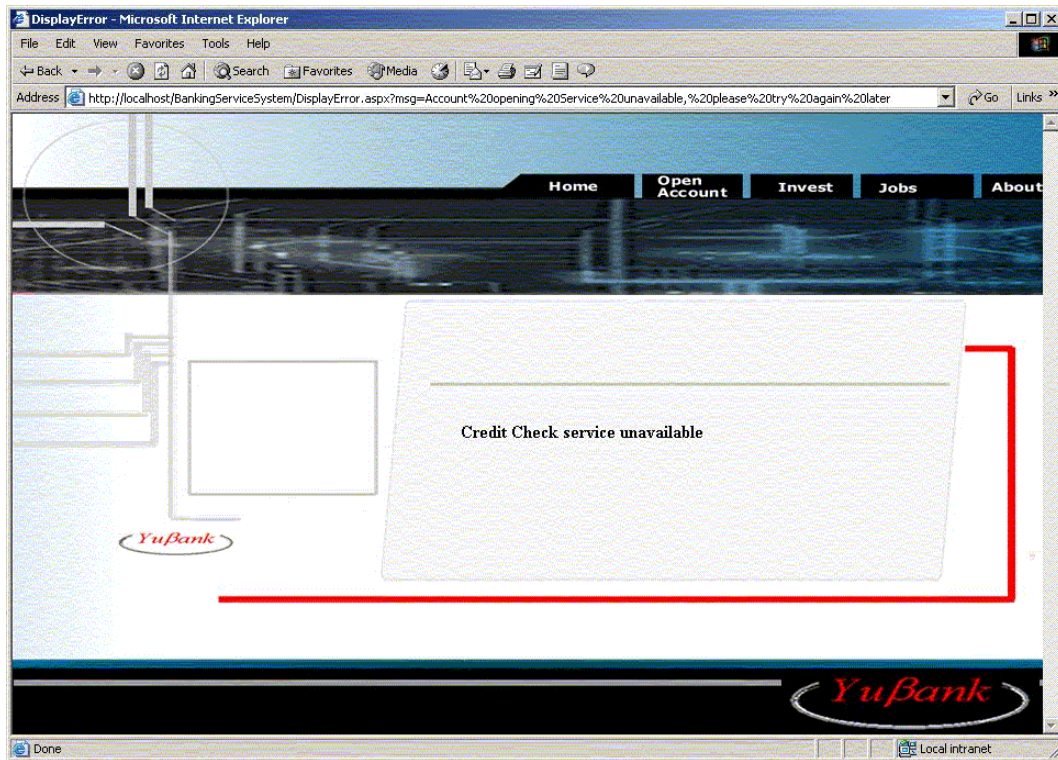


Figure 11-8 Credit Check service unavailable

Customer profile service unavailable

Repeat Step 1 to Step 3 from Test Case 1: Successful account opening above.

Step 4: The customer clicks the **submit investment details**. If the customer profile service is unavailable due to inaccessible queues or queue manager, the customer is still given an account number and faced with a page that looks like the one below:

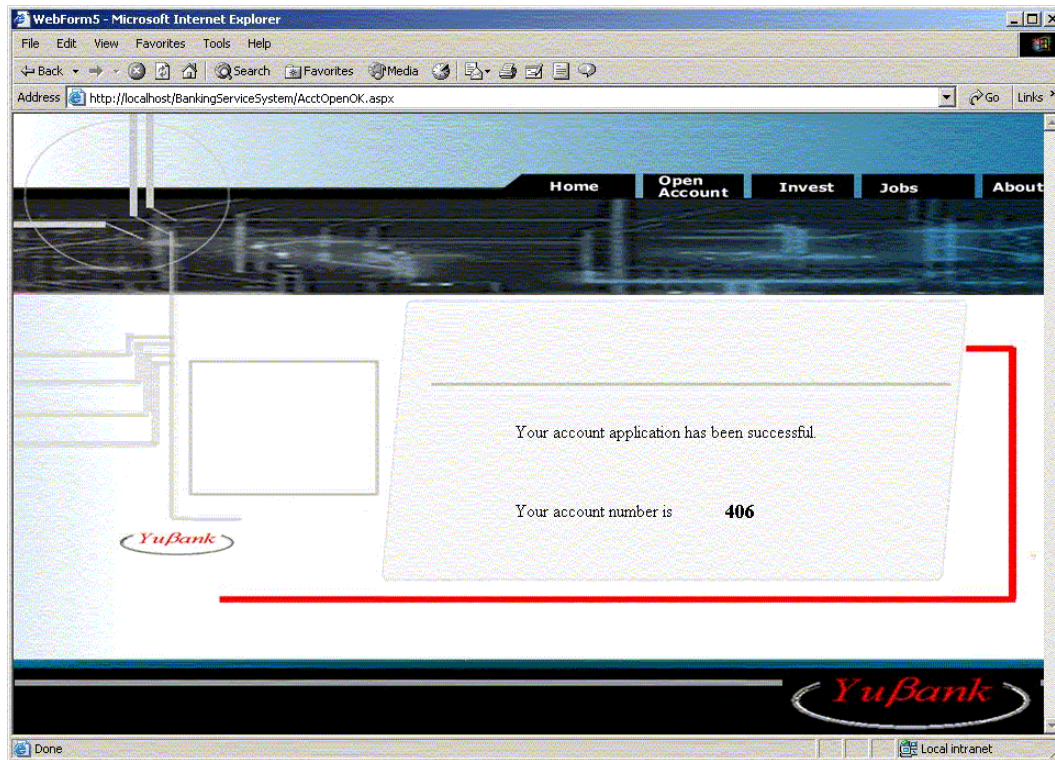


Figure 11-9 Account number created

Step 5: To verify that the customer profile service is unavailable and that the customer details haven't been processed:

- ▶ Click the **invest** tab, enter the customer account number created when customer profile service was unavailable, click **submit**.
- ▶ The investment advice service returns a "No match found" error as shown below:

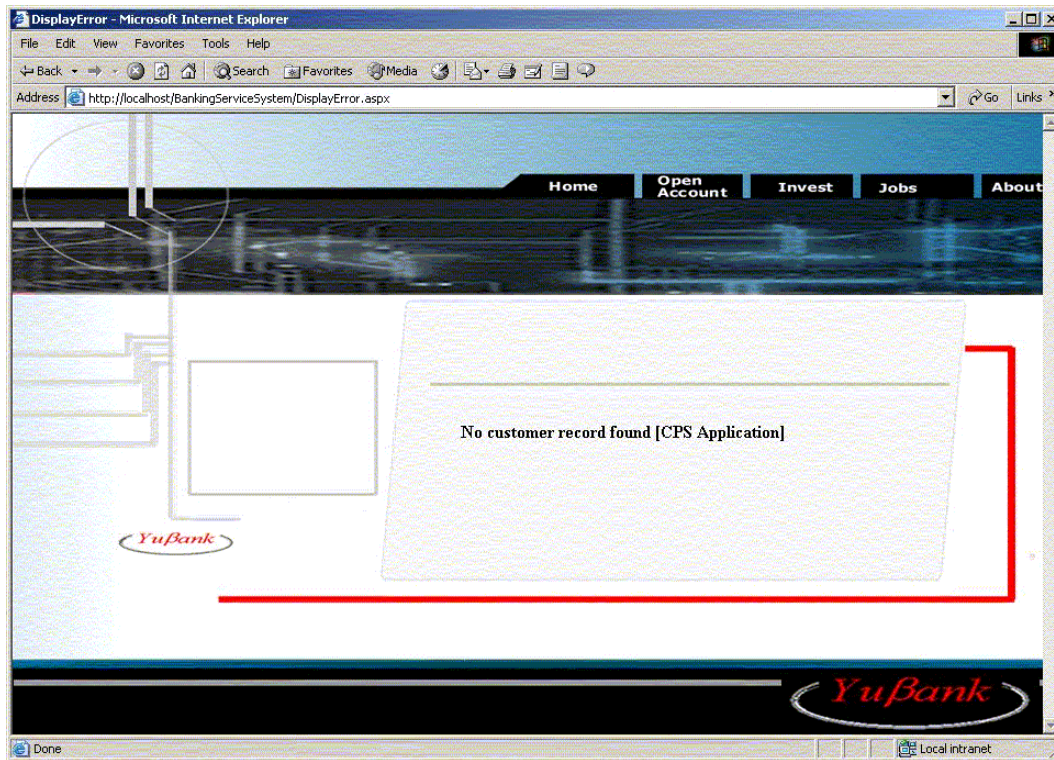


Figure 11-10 No match found

Explanation: This is the expected result. Even though the account is created, the details are not yet been stored, therefore, if investment is attempted, the customer details cannot be found.

Start the customer profile service and wait until all pending messages in the queue are processed. Then retry investing for the same customer account, the investment advice returns an advice as shown below:

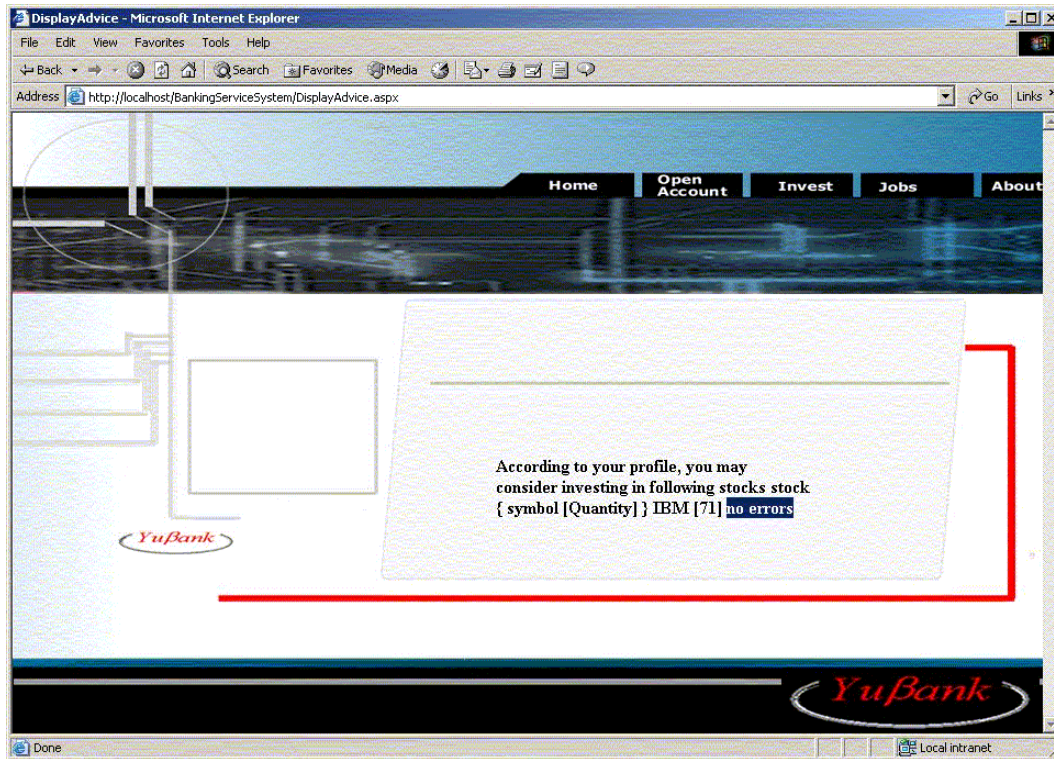


Figure 11-11 Advice results display

This test case demonstrates the asynchronous capability of WebSphere MQ. In contrast to a blocking call in a synchronous manner like HTTP, BBS uses the queuing feature in WebSphere MQ, and provides the service continuity to users even though one subsystem is temporarily out of service.

11.3.3 Use case 2: Investment advisory

This section describes the test plan for investment advisory use case 2.

General data flow

The diagram below shows the general data flow in use case 2 of the business case scenario.

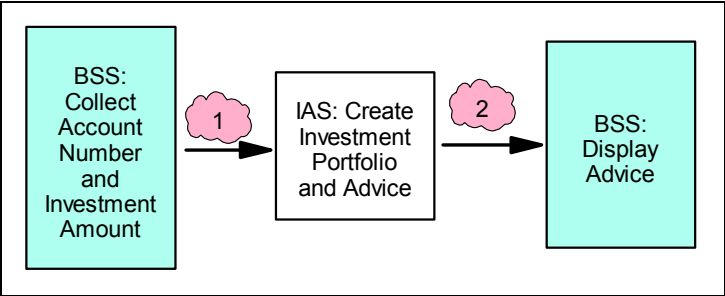


Figure 11-12 General data flow in use case 2

Number	Data
1	Customer account number and investment amount
2	Advice

Test Case 1: Successful investment advisory - portfolio recommendations

Step1: On starting the process, the customer is faced with the bank’s home page.

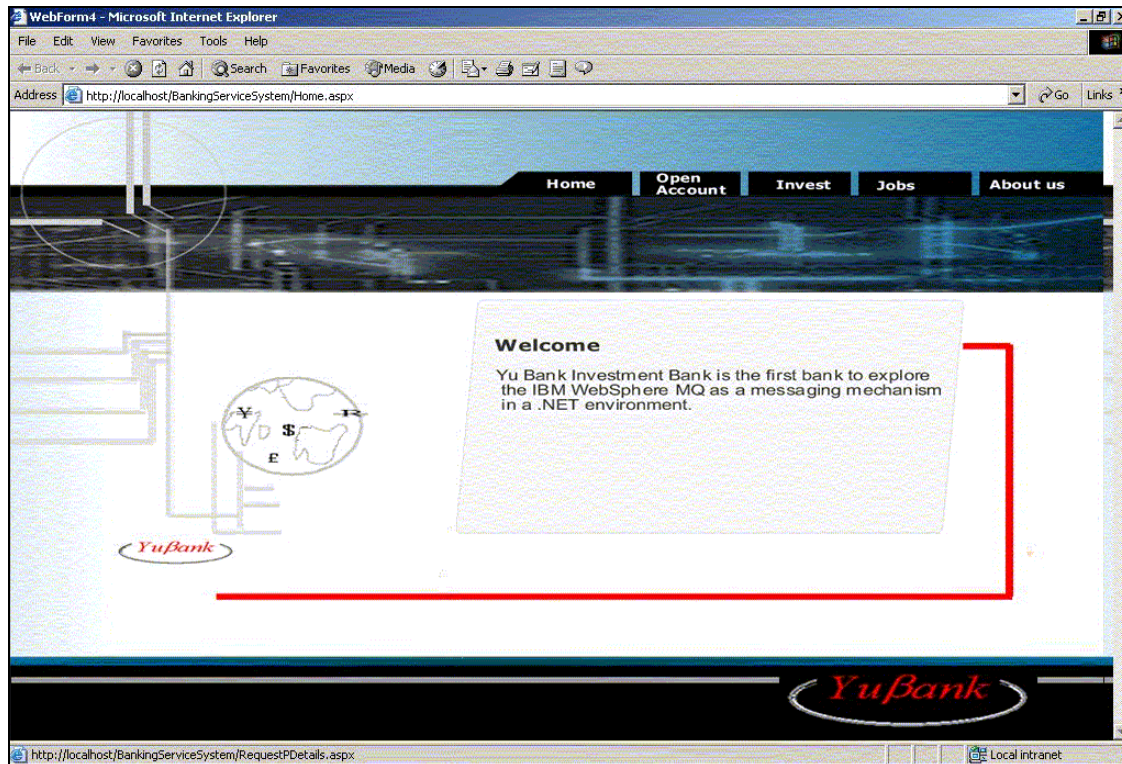


Figure 11-13 Home page

Step 2: The customer clicks the **Invest** tab and is faced with a page requesting the account number and investment amount as shown below:

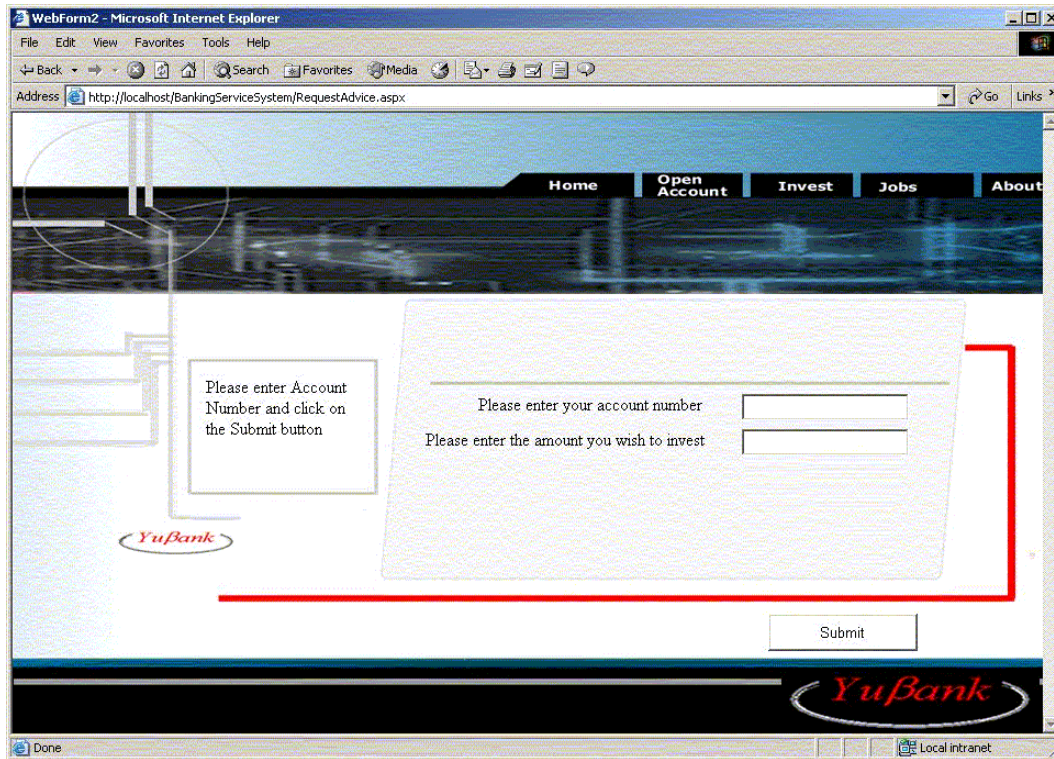


Figure 11-14 Request investment details

Step 3: The customer enters details and clicks the Submit button, a Web Service accesses the customer details from a database, based on the customer's details, a portfolio recommendation is produced. This recommendation is returned in the form of an advice to the customer on a page as shown next:

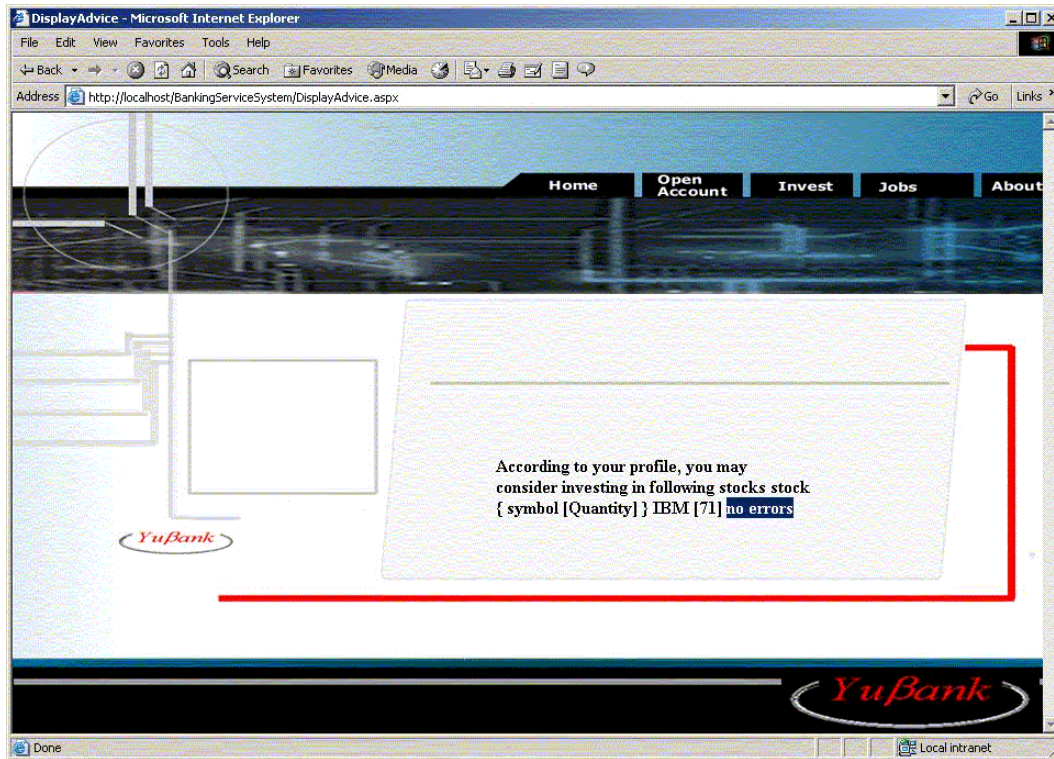


Figure 11-15 Advice results display

This test case verifies the normal path for the processing. A set of qualified shares are selected based on the customer criteria and intelligent analysis, and a portfolio structure is recommended.

Test Case 2: Successful investment advisory - no investment

Repeat Step 1 and Step 2 from Test Case 1: Investment advisory above.

Step 3: The customer enters details and clicks the **submit** button. If the investment service is unavailable, as a result of inaccessible queue manager onPageLoad, the customer is faced with a page shown next:

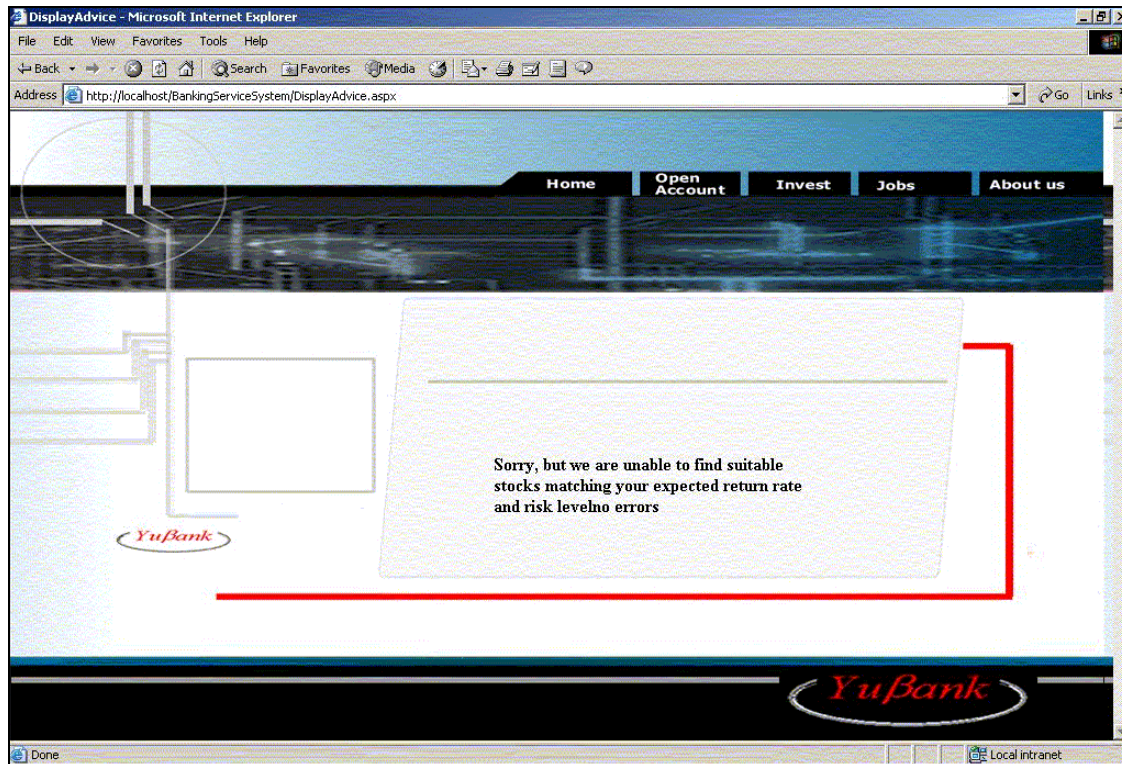


Figure 11-16 Result when no match is found

This test case verifies an alternative path for the processing. No portfolio is recommended because no qualified shares can be found to meet the customer's return expectations.

Test Case 3: Unsuccessful investment advisory - invalid customer data

Repeat Step 1 and Step 2 from Test Case 1: Investment advisory above.

Step 3: The customer's details are entered before applying Submit. A Web Service accesses the customer's details from a database in order to generate a portfolio recommendation. If no match is found for the customer then feedback to the customer is sent on the page shown next:

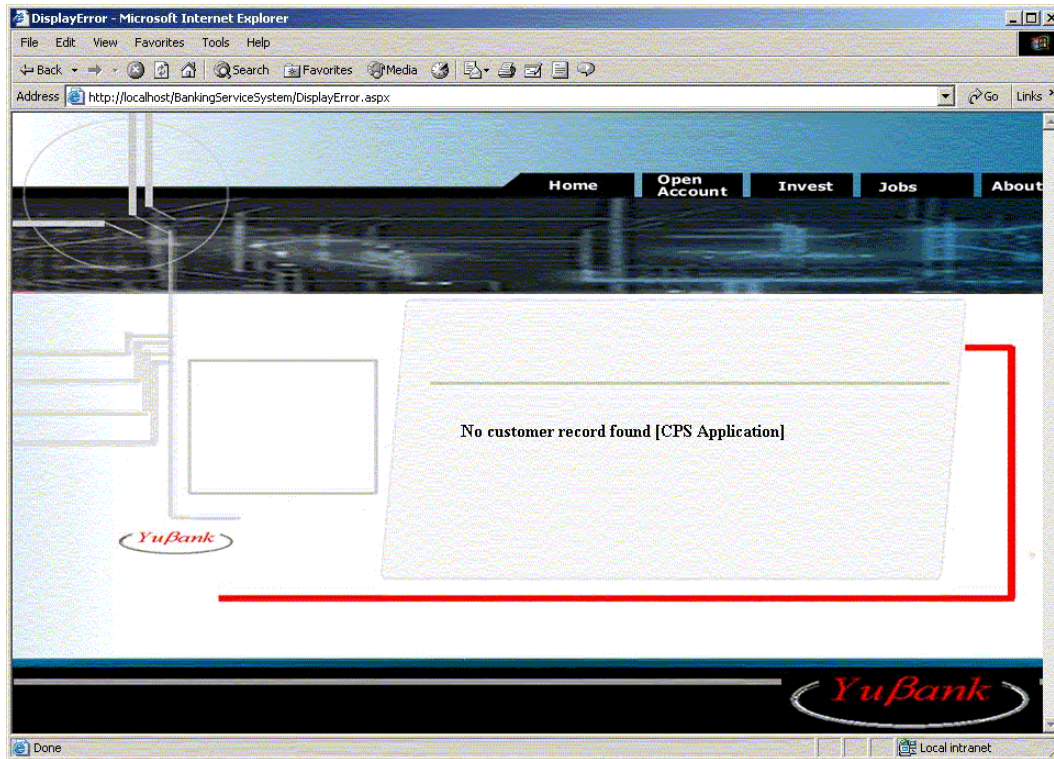


Figure 11-17 Customer record not found

This test case verifies an alternative path for the processing. The system can gracefully handle an invalid data entry by a customer.

Test Case 4: Negative tests - exception recovery and system unavailability

Repeat Step 1 and Step 2 from Test Case 1: Investment advisory above.

Step 3: The customer's details are entered before applying Submit. If the Web Service is inaccessible, the customer is faced with the page shown next:

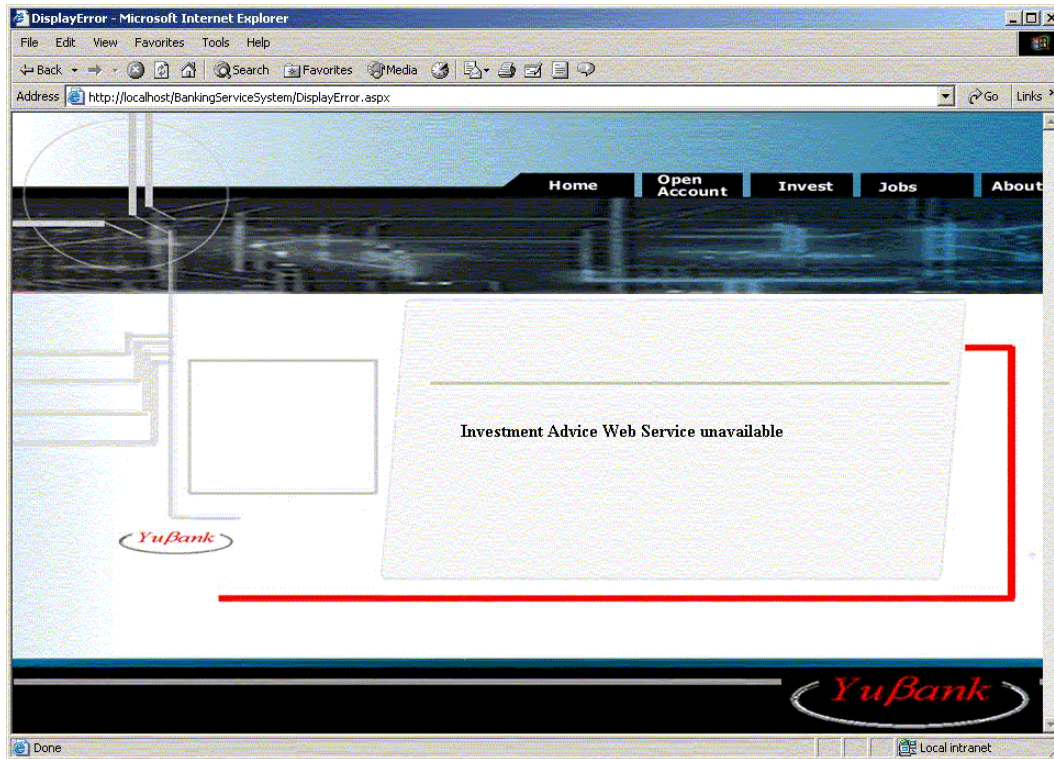


Figure 11-18 Web Service unavailable

This test case verifies an alternative path for the processing. If a subsystem necessary for the processing is down, a user is notified that the system has an unavailable service.

11.4 Summary

This chapter describes the system integration and functional test for the demonstration bank system developed in this book. The runtime environment, test data, deployment and system startup are discussed. Various test cases are created for the two use cases implemented in this bank system. The major functionality of the system are tested and verified, including successful, unsuccessful, and exception scenarios. One key feature demonstrated is that the system can provide the account opening service without interruption, even though one of the subsystems, CCS, is temporarily down.



Security

This chapter describes the techniques and implementation for securing the transportation of messages between applications and Web Services. It focuses mainly on the security issues related to WebSphere MQ in the business scenario discussed in the previous chapters.

The following topics are discussed in this chapter:

- ▶ Security concepts
- ▶ Planning the security services in use cases
- ▶ Cryptographic concepts
- ▶ Secure Sockets Layer (SSL) introduction
- ▶ WebSphere MQ SSL support
- ▶ WebSphere MQ working with SSL on Windows
- ▶ Deploy SSL support in use cases

12.1 Security concepts

Before beginning to discuss how to secure the transportation of messages between applications and Web Services, it is useful to consider some security problems that systems face. These key security problems include:

- ▶ Unauthorized access: occurs when an unknown user or application wants to read or write some critical information.
- ▶ Eavesdropping: occurs when unknown people understand the messages that are passed between the communicating parties.
- ▶ Tampering: involves someone intercepting a message and then changing it.
- ▶ Impersonating: occurs when a message is sent by someone other than the actual specified sender.

Figure 12-1 shows eavesdropping and tampering.

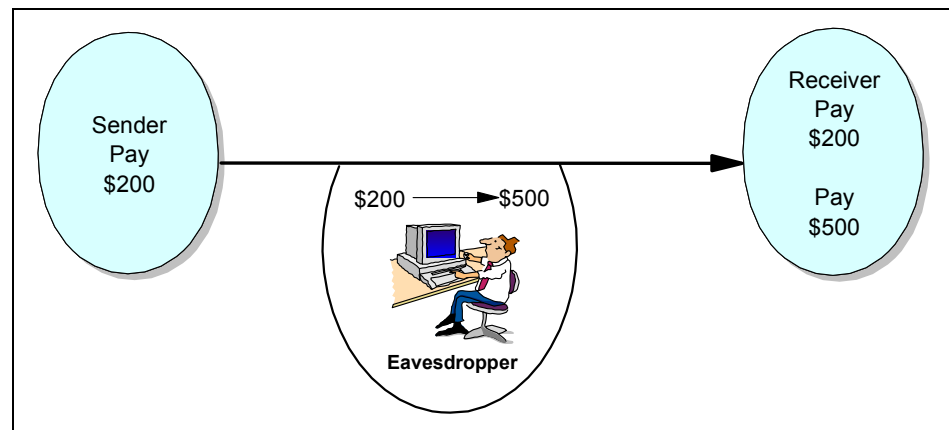


Figure 12-1 Example of eavesdropping and tampering

To avoid these security problems, a good system provides security services. The following section gives an overview of security services and the mechanisms to implement these services.

12.1.1 Security services

Security services are services within a computer system that protect its resources. There are mainly five security services that are identified in a security architecture.

- ▶ Identification and authentication
- ▶ Authority

- ▶ Confidentiality
- ▶ Data integrity
- ▶ Non-repudiation

Identification is being able to identify uniquely, a user of a system or an application that is running in the system. Authentication is being able to prove that a user or application is genuine and the one being used.

The authority service protects critical resources in a system by limiting access only to authorized users and their applications. It prevents unauthorized use of a resource or the use of a resource in an unauthorized manner.

The confidentiality service protects sensitive information from unauthorized disclosure.

The data integrity service detects whether there has been unauthorized modification of data. There are two ways in which data might be altered: accidentally, through hardware and transmission errors, or because of deliberate attacks. Many hardware products and transmission protocols now have mechanisms to detect and correct hardware and transmission errors. The purpose of the data integrity service is to detect a deliberate attack.

The non-repudiation service can be viewed as an extension to the identification and authentication service. In general, non-repudiation applies when data is transmitted electronically, for example, an order to a stock broker to buy or sell stock, or an order to a bank to transfer funds from one account to another. The overall goal is to be able to prove that a particular message is associated with a particular individual.

12.1.2 Security mechanisms

Security mechanisms are technical tools and techniques that are used to implement security services. A mechanism might operate by itself, or in conjunction with the others, to provide a particular service. Examples of common security mechanisms are:

- ▶ User database
- ▶ Authority database
- ▶ Cryptography
- ▶ Digital signatures

Refer to manual *WebSphere MQ Security*, SC34-6079 and redbook *WebSphere MQ Security in an Enterprise Environment*, SG24-6814 for further detail.

12.2 Planning the security services in use cases

This section covers the design of the security services in the business scenario discussed in the previous chapters.

A typical application system usually has several applications running on several computers and the intercommunication of these applications may cross various kinds of media, sometimes vulnerable media such as the Internet.

Security services in different scenario are different. For example, intercommunication between two applications running on the same machine may not worry about eavesdropping or tampering, and the authentication services on message transmission is different to the authentication services that protect local resources. Therefore, different security services are provided in different scenarios and from the application point of view, there is an identification service, an authentication service, and an authority service to entitle the requesting user access to resources. As from the message transmission point of view, there is identification service, authentication service, and authority service to verify whether the request is issued by the genuine user.

It is a good idea to separate the security services into several layers, and provide different security services in different layers. In this book, the security services are separated into two layers:

- ▶ The application layer security services
- ▶ The transmission layer security services

The subsequent sections details the two layer of services.

12.2.1 Application layer security services

In the application layer, the security services considers the end-to-end data security, including:

- ▶ Identification service and authentication service
- ▶ Authority service

The identification service and authentication service can be implemented by a user database, such as the system user database or a user database created by the customer.

The authority service can be implement by the attribute of data objects.

WebSphere MQ provide a good security solution for these security services. Refer to manual *WebSphere MQ Security*, SC34-6079 and redbook *WebSphere MQ Security in an Enterprise Environment*, SG24-6814 for further detail.

12.2.2 Transmission layer security services

In the transmission layer, the security services considers the following services:

- ▶ Identification and authentication services
- ▶ Confidentiality service
- ▶ Data integrity service
- ▶ Non-repudiation

In the business scenario discussed in the previous chapters, there are five applications. The five applications intercommunicate through WebSphere MQ queue manager and the client. Refer to Chapter 5, “Solution design” on page 79 for details.

The Bank Service System (BSS) communicates with the Credit Check System (CCS) and Investment Advisory System (IAS) through a WebSphere MQ client across the Internet. The Customer Profile System (CPS), Investment Advisory System (IAS), and the Share Quote System (SQS) are applications within the YuBank, and they communicate to each other through WebSphere MQ queue managers in the intranet.

Among many techniques to implement these security services, Secure Sockets Layer (SSL) is a good solution. WebSphere MQ provides good SSL support in version 5.3.

There are other good solutions such as Web Services Security (WS-Security). The strength of the Web Services framework is its ability to work as an integration platform which brings heterogeneous systems and applications together. This also imposes certain security requirements on the Web Services framework. The communication between Web Services, between a Web Service and other applications often requires certain level of security. To achieve this, Web Services framework requires a strong security model that brings together incompatible security technologies such as Public Key Infrastructure (PKI), Kerberos, and others.

IBM, Microsoft and VeriSign have proposed Web Services security specification to cover broader security issues relating to messages in Web Services communication. WS-Security defines enhancements to Simple Object Access Protocol (SOAP) for protecting the integrity and confidentiality of a message. WS-Security also provides a generic mechanisms for integrating security tokens with the messages and procedures for encoding binary security.

As all inter-communications in the business case scenario discussed in this redbook are not based on Web Services, the WS-Security is not implemented on all the message transmission in the use cases. For complete information about WS-Security, refer to Specification: Web Services Security at:

<http://www-106.ibm.com/developerworks/webservices/library/ws-secure/>

This redbook does not implement SSL security services between IAS and CPS, and between IAS and SQS for the following reasons:

- ▶ In the solution of YuBank, the three application, IAS, CPS, and SQS, are all within the intranet of YuBank.
- ▶ The message flows among these applications are conveyed by WebSphere MQ queue manager. The implementation of SSL security services between queue managers are well described and demonstrated in manual *WebSphere MQ Security*, SC34-6079 and redbook *WebSphere MQ Security in an Enterprise Environment*, SG24-6814.

However, two message links below needed to be made secure in the business case scenario:

- ▶ The BSS to the CCS
- ▶ The BSS to the IAS

Bank Service System (BSS) communicates with the Credit Check System (CCS) and Investment Advisory System (IAS) through a WebSphere MQ client across the Internet.

Security is also implemented on the three applications involved, with security services discussed above. The following sections indicate how SSL is implemented on these three applications.

12.3 Cryptographic concepts

- ▶ Before implementing SSL in the use cases, it is useful to know some concepts of cryptography. This section gives an introduction to some cryptography concepts. Some of the knowledge in this section comes from X.509 specification. For more information, refer to Internet X.509 Public Key Infrastructure at:

<http://www.ietf.org/internet-drafts/draft-ietf-pkix-logotypes-10.txt>

12.3.1 Cryptography

Cryptography is the process of between converting readable text, called *plaintext*, and an unreadable form, called *ciphertext*.

1. The sender converts the plaintext message to ciphertext. This part of the process is called encryption (sometimes encipherment).
2. The ciphertext is transmitted to the receiver.

3. The receiver converts the ciphertext message back to its plaintext form. This part of the process is called decryption (sometimes decipherment). As shown in Figure 12-2.

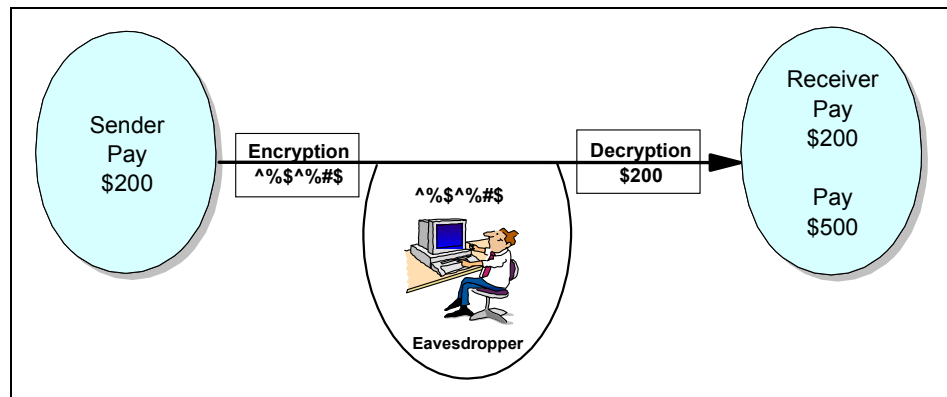


Figure 12-2 Puzzled eavesdropper

The conversion involves a sequence of mathematical operations that change the appearance of the message during transmission but does not affect the content. Cryptographic techniques can ensure confidentiality and protect messages against unauthorized viewing (eavesdropping), because an encrypted message is not understandable. Digital signatures, which provide an assurance of message integrity, use encryption techniques. See 12.3.3, “Digital signature” on page 257 for more information.

Cryptographic techniques involve a general algorithm, made specific by the use of keys. There are two classes of algorithm:

- ▶ Symmetric key algorithm
- ▶ Asymmetric key algorithm

Symmetric key cryptography requires both parties to use the same secret key as shown in Figure 12-3.

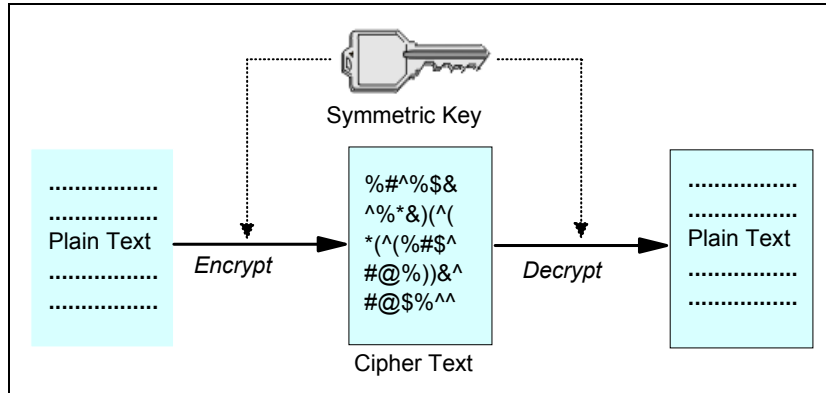


Figure 12-3 Symmetric key cryptography

Asymmetric key cryptography uses one key for encryption and a different key for decryption. One of these must be kept secret but the other can be public. Figure 12-4 illustrates how asymmetric key cryptography works. Asymmetric key cryptography is also known as public key cryptography.

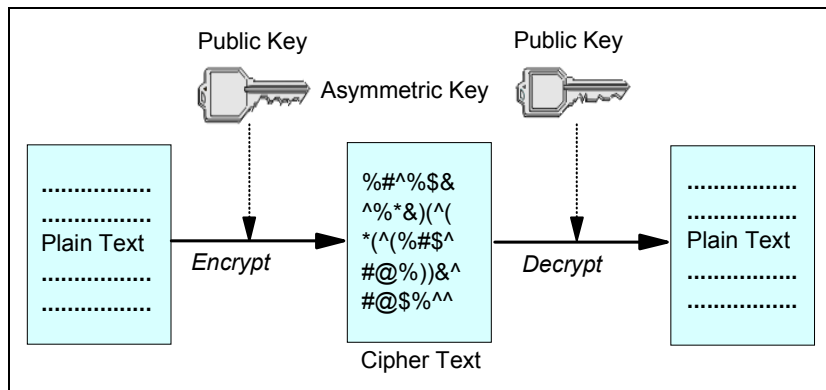


Figure 12-4 Asymmetric key cryptography

The encryption and decryption algorithms used can be public but the shared secret key and the private key must be kept secret.

12.3.2 Message digest

Data integrity and tampering can be addressed by a message digest, which is a fixed size numeric representation of the contents of a message. The message digest (also called the message authentication code) is computed using a hash function. The hash function meets two criteria:

- ▶ The hash function is one-way. It is not possible to reverse the function to find the message corresponding to a given message digest, other than by testing all possible messages.
- ▶ It is computational infeasible to find two messages that hash to the same digest.

A message digest is also known as a Message Authentication Code (MAC), because it can provide assurance that the message has not been modified. Once computed, the digest is transmitted along with the message. The receiver then invokes the same hash function on the received message and compares the generated digest to the one received. If the digests are the same, then the message was not tampered with en route. Any tampering with the message during transmission almost certainly results in a different message digest. The sender and receiver must agree on the hash function that to be used before transmission begins for this process to work.

12.3.3 Digital signature

A digital signature is formed by encrypting a representation of a message. The encryption uses the private key of the signatory and, for efficiency, usually operates on a message digest rather than the message itself. See 12.3.2, “Message digest” on page 256 for more information.

Digital signatures vary with the data being signed, unlike handwritten signatures, they do not depend on the content of the document being signed. If two different messages are signed digitally by the same entity and the two signatures differ, both signatures can be verified with the same public key, that is, the public key of the entity that signed the messages.

The steps of the digital signature process are as follows:

1. The sender computes a message digest and then encrypts the digest using the sender's private key, forming the digital signature.
2. The sender transmits the digital signature with the message.
3. The receiver decrypts the digital signature using the sender's public key, regenerating the sender's message digest.
4. The receiver computes a message digest from the message data received and verifies that the two digests are the same.

If the digital signature is verified, the receiver knows that:

- ▶ The message has not been modified during transmission.
- ▶ The message was sent by the entity that claims to have sent it.

Digital signatures are part of integrity and authentication services. Digital signatures also provide proof of origin. Only the sender knows the private key, which provides strong evidence that the sender is the originator of the message.

12.3.4 Digital certificate

Digital signatures are used to verify that the message is sent by the sender; they combine the use of message digests and public key cryptography. The process involves the sender generating the message digest, then encrypting the digest using its private key to create the digital signature. The receiver then decrypts the message digest using the sender's public key, confirming that the message is indeed sent by the sender. Comparing the digest with the one the receiver generates further confirms that the message has not been changed since it was signed.

However, how does the receiver know that the public key can be trusted? Digital certificates provide protection against impersonation, because a digital certificate binds a public key to its owner, whether that owner is an individual, a queue manager, or some other entity.

The rest of this section introduces digital certificate in detail.

What is a digital certificate?

Digital certificates are also known as public key certificates, because they give assurances about the ownership of a public key when used as in an asymmetric key scheme. A digital certificate contains the public key for an entity and is a statement that the public key belongs to that entity. The digital certificate is typically issued by a trusted third party called a Certification Authority (CA). When the certificate is for an individual entity, it is called a *personal certificate* or *user certificate*. When the certificate is for a CA, the certificate is called a *CA certificate* or *signer certificate*. For a fee, the CA generates a digital certificate that contains:

- ▶ The owner's public key
- ▶ The owner's Distinguished Name
- ▶ The Distinguished Name of the CA that is issuing the certificate
- ▶ The date from which the certificate is valid
- ▶ The expiry date of the certificate
- ▶ A version number
- ▶ A serial number

Before issuing a certificate, Certification Authorities run appropriate background checks on the requestor to verify that the requestor is who it says it is. All certificates issued by a CA are digitally signed by the CA, and can be verified using the CA certificate (which contains the CA's public key). The exchange

between the sender and receiver now takes on an additional verification step. Instead of simply sending its public key to the receiver, the sender sends its digital certificate issued by a CA. The receiver uses the CA's certificate to verify the sender's certificate, ascertaining that the public key contained in the certificate truly belongs to the owner. During the lifetime of a digital certificate, the issuing CA might determine that the certificate is no longer trustworthy. Such certificates are published to a Certificate Revocation List (CRL), against which both the sender and receiver can choose to check the received certificates.

Certification Authorities

A Certification Authority (CA) is an independent and a trusted third party that issues digital certificates to provide an assurance that the public key of an entity truly belongs to that entity. The roles of a CA are:

- ▶ On receiving a request for a digital certificate, to verify the identity of the requestor before building, signing and returning the personal certificate
- ▶ To provide the CA's own public key in its CA certificate
- ▶ To publish lists of certificates those are no longer trusted in a Certificate Revocation List (CRL). For more information, refer to Internet X.509 Public Key Infrastructure at:

<http://www.ietf.org/internet-drafts/draft-ietf-pkix-logotypes-10.txt>

Distinguished Names

The Distinguished Name (DN) uniquely identifies an entity in an X.509 certificate. The following attribute types are commonly found in the DN:

- ▶ CN: Common Name
- ▶ T: Title
- ▶ Organization name
- ▶ OU: Organizational Unit name
- ▶ L: Locality name
- ▶ ST (or SP or S): State or Province name
- ▶ C: Country (or region)

The X.509 standard defines other attributes that do not usually form part of the DN but can provide optional extensions to the digital certificate.

The X.509 standard provides for a DN to be specified in a string format. For example: CN=John Smith, O=IBM, OU=Test, C=GB

The Common Name (CN) can describe an individual user or any other entity, for example a Web server.

The DN can contain multiple OU attributes, but one instance only for each of the other attributes is permitted. The order of the OU entries is significant: the order specifies a hierarchy of Organizational Unit names, with the highest-level unit first.

How digital certificates work

Obtain a digital certificate by sending information to a CA. The X.509 standard defines a format for this information, but some CAs have their own format. Certificate requests are usually generated by the certificate management tool the operating system uses, for example the iKeyman tool on UNIX® systems. The information comprises the Distinguished Name and is accompanied by the public key. When the certificate management tool generates a certificate request, it also generates a private key, which must be kept secure. Never distribute the private key.

When the CA receives the request, the authority verifies the requester's identity before building the certificate and returning it to the requester as a personal certificate.

Obtaining personal certificates

A digital certificate can be obtained by either requesting a digital certificate from a CA, or generate a self-signed certificate with environment-specific tools. With a self-signed certificate, the user acts as its own CA. Self-signed certificates can be useful for test environments because users can generate them locally and do not have to pay fees to a CA. However, many Certification Authorities offer a demo facility that can generate demo (test) certificates at no charge. Also, in some environments, users can request a CA to generate a CA certificate for a self-signed certificate, which validates and resigns the previously generated certificate. Figure 12-5 illustrates how to obtain a personal certificate.

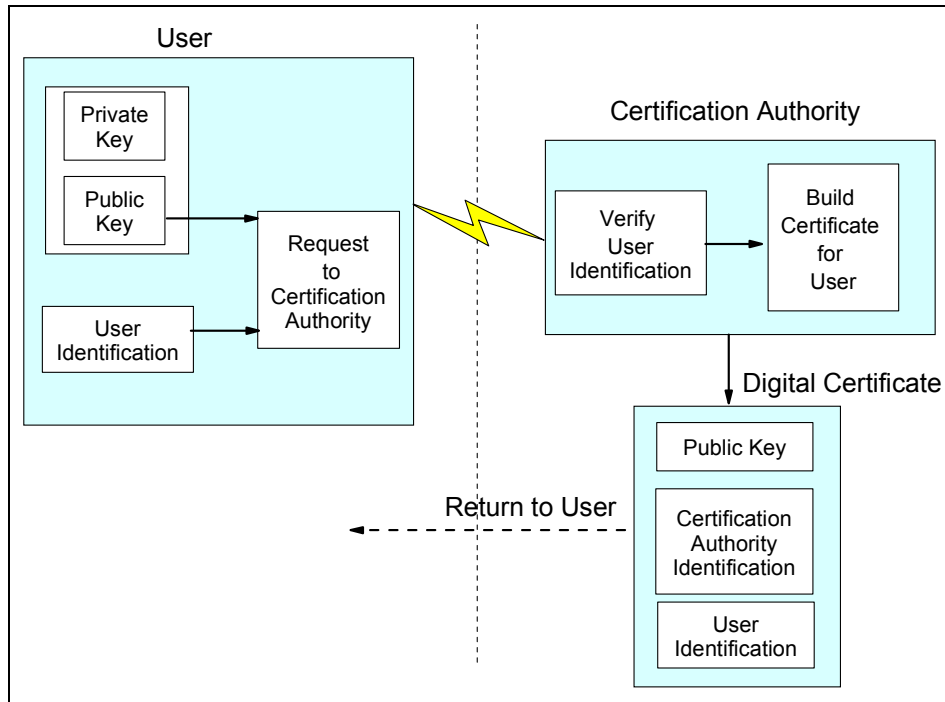


Figure 12-5 Obtaining a personal certificate

Certificate chain

When an application receives a certificate for another entity, it needs to use a certificate chain to obtain the root CA certificate. The certificate chain, also known as the certification path, is a list of certificates used to authenticate an entity. The chain or path, begins with the certificate of that entity, and each certificate in the chain is signed by the entity identified by the next certificate in the chain. The chain terminates with a root CA certificate. The root CA certificate is always signed by the CA itself. The signatures of all certificates in the chain must be verified until the root CA certificate is reached.

When certificates are no longer valid

Digital certificates are issued for a fixed period and are not valid after their expiry date. Certificates can also become untrustworthy for various reasons, including:

- ▶ The owner has moved to a different organization.
- ▶ The private key is no longer secret.

- ▶ A Certification Authority can revoke a certificate that is no longer trusted by publishing it in a Certificate Revocation List (CRL). For more information, refer to Internet X.509 Public Key Infrastructure at:

<http://www.ietf.org/internet-drafts/draft-ietf-pkix-logotypes-10.txt>

12.3.5 Public Key Infrastructure (PKI)

A Public Key Infrastructure (PKI) is a system of facilities, policies, and services that support the use public key cryptography for authenticating the parties involved in a transaction. There is no single standard that defines the components of a Public Key Infrastructure, but a PKI typically comprises Certification Authorities and other Registration Authorities (RAs) that provide the following services:

- ▶ Issuing digital certificates
- ▶ Validating digital certificates
- ▶ Revoking digital certificates
- ▶ Distributing public keys

Refer to 12.3.4, “Digital certificate” on page 258 for more information about digital certificates and Certification Authorities (CAs). RAs verify the information provided when digital certificates are requested. Once the RA verifies that information, the CA can then issue a digital certificate to the requester.

- ▶ A PKI might also provide tools for managing digital certificates and public keys. A PKI is sometimes described as a trust hierarchy for managing digital certificates, but most definitions include additional services. Some definitions include encryption and digital signature services, but these are not essential to the operation of a PKI. For more information about PKI, refer to Internet X.509 Public Key Infrastructure at:

<http://www.ietf.org/internet-drafts/draft-ietf-pkix-logotypes-10.txt>

12.4 Secure Sockets Layer (SSL) introduction

SSL is a very common used security protocol in the intercommunication of critical data on the internet. This section discusses some concepts of SSL.

12.4.1 Secure Sockets Layer(SSL) concepts

The Secure Sockets Layer (SSL) provides an industry standard protocol for transmitting data in a secure manner over an insecure network. The SSL protocol is widely deployed in both Internet and intranet applications. SSL defines methods for authentication, data encryption, and message integrity for a reliable transport protocol, usually Transmission Control Protocol/Internet Protocol

(TCP/IP). SSL uses both asymmetric and symmetric cryptography techniques. For complete description of SSL, refer to SSL 3.0 specification at:

<http://wp.netscape.com/eng/ss13/>

An SSL connection is initiated by the caller application, which becomes the SSL client. The responder application becomes the SSL server. Every new SSL session begins with an SSL handshake, as defined by the SSL protocol.

The following issues should be considered in SSL:

- ▶ SSL handshake
- ▶ SSL authentication
- ▶ SSL confidentiality
- ▶ SSL integrity

For detail information about these issues, refer to manual *WebSphere MQ Security*, SC34-6079, or SSL 3.0 specification at:

<http://wp.netscape.com/eng/ss13/>

12.4.2 CipherSuites and CipherSpecs

A CipherSuite is a suite of cryptographic algorithms used by an SSL connection. A suite comprises three distinct algorithms:

- ▶ The key exchange and authentication algorithm, used during the SSL handshake
- ▶ The encryption algorithm, used to encipher the data
- ▶ The MAC (Message Authentication Code) algorithm, used to generate the message digest

For complete information about CipherSuites and CipherSpecs, refer to manual *WebSphere MQ Security*, SC34-6079, or SSL 3.0 specification at:

<http://wp.netscape.com/eng/ss13/>

12.5 WebSphere MQ SSL support

Message channels and Message Queue Interface (MQI) channels can use the SSL protocol to provide link level security. A caller Message Channel Agent (MCA) is an SSL client and a responder MCA is an SSL server. WebSphere MQ supports Version 3.0 of the SSL protocol. The cryptographic algorithms that are used by the SSL protocol are supplied by the CipherSpec part of the channel definition.

At the server end of an MQI channel and at each end of a message channel, the MCA acts as a security service on behalf of the queue manager to which it is connected. During the SSL handshake, the MCA sends the digital certificate of the queue manager to its partner MCA at the other end of the channel. The WebSphere MQ code at the client end of an MQI channel acts on behalf of the user of the WebSphere MQ client application. During the SSL handshake, the WebSphere MQ code sends the user's digital certificate to the MCA at the server end of the MQI channel.

Digital certificates are stored in a key repository. The queue manager attribute `SSLKeyRepository` specifies the location of the key repository that holds the queue manager's digital certificate. On a WebSphere MQ client system, the `MQSSLKEYR` environment variable specifies the location of the key repository and the key (without the `.STO` suffix) that holds the user's digital certificate. Alternatively, a WebSphere MQ client application can specify its location in the `KeyRepository` field of the SSL configuration options structure, `MQSCO`, on an `MQCONN` call.

For complete information of WebSphere MQ SSL support, refer to manual *WebSphere MQ Security*, SC34-6079.

12.6 WebSphere MQ working with SSL on Windows

This section describes how to set up WebSphere MQ queue manager and WebSphere MQ client to work with the Secure Sockets Layer (SSL) on Windows systems.

The following tasks must be performed to implement SSL between a queue manager and a client:

- ▶ Setup a key repository
- ▶ Work with a key repository
- ▶ Obtain personal certificates
- ▶ Add personal certificates to a key repository
- ▶ Manage digital certificates
- ▶ Map DNs to user IDs

On Windows 2000 and XP, SSL support is integral to the operating system. Microsoft Internet Explorer provides the SSL support on other Windows platforms. Windows SSL support is documented in the Microsoft Developer Network (MSDN) library at:

<http://msdn.microsoft.com/library/default.asp>

Refer to *WebSphere MQ Security*, SC34-6079 for information about how to complete the tasks above.

12.7 Deploy SSL support in use cases

This section discusses how to implement SSL support on the applications used in the business case scenario.

12.7.1 Obtaining certificates

To deploy SSL support in the use cases, the first job is to obtain certificates for the applications. Certificates can be requested from a CA or generate by a system specific tool.

This redbook uses a CA's demo facilities to generate certificates for the applications.

Certificates are stored in repositories that vary, based on the operating system and the tools used to access them. On the Windows platform, certificates are stored in Microsoft Certificate Stores (MCS) and can be viewed and manipulated using Internet Explorer.

Obtaining certificate from CA

1. In Internet Explorer browser, launch the URL:
http://www.digsigtrust.com/prod_serv/index.html
2. Click **Get a TrustID Demo Certificate**. (Refer to Figure 12-6.)

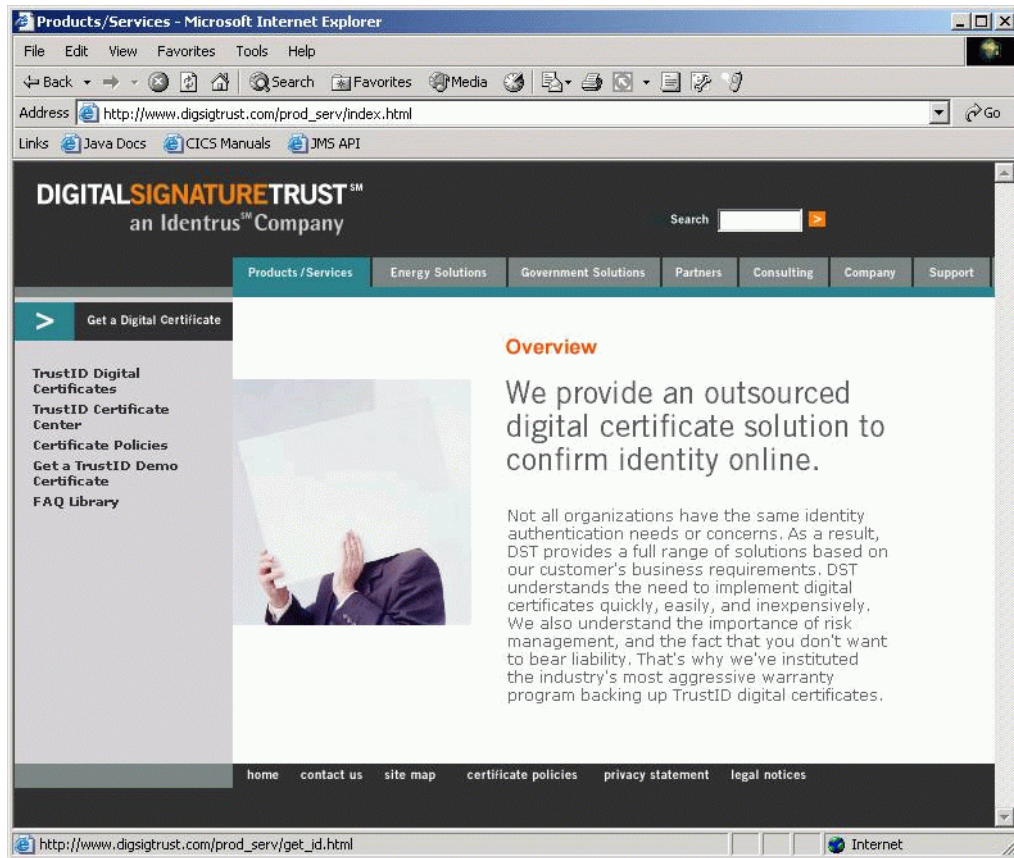


Figure 12-6 Get a TrustID Demo Certificate

3. Fill in the identification form. The certificate is based on the name, organization, and location details. In Figure 12-7, the name BSS REDBOOK is used to identify the BSS application certificate. The certificate is sent to the e-mail address provided. The passphrase provided is used to retrieve the certificate. Click **Continue**.

DST TrustSource - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Media Print Mail

Address <https://secure.digistrust.com/ts/demo-register.html> Go

Links Java Docs CICS Manuals JMS API

[privacy statement](#) for more information).

FIRST NAME

MIDDLE INITIAL

LAST NAME

ORGANIZATION (optional)

CITY

STATE

E-MAIL

DAYTIME PHONE EXT.

Please enter a passphrase. You will use your passphrase to retrieve your demo certificate. Your passphrase should be between eight and thirty characters long and can include letters and/or numbers.

ENTER PASSPHRASE

RE-ENTER PASSPHRASE

Done Internet

Figure 12-7 Server Identification

4. Review the information and click **Continue**.
5. Select **Browser** as the option for storing the certificate (see Figure 12-8). Click **Continue**.

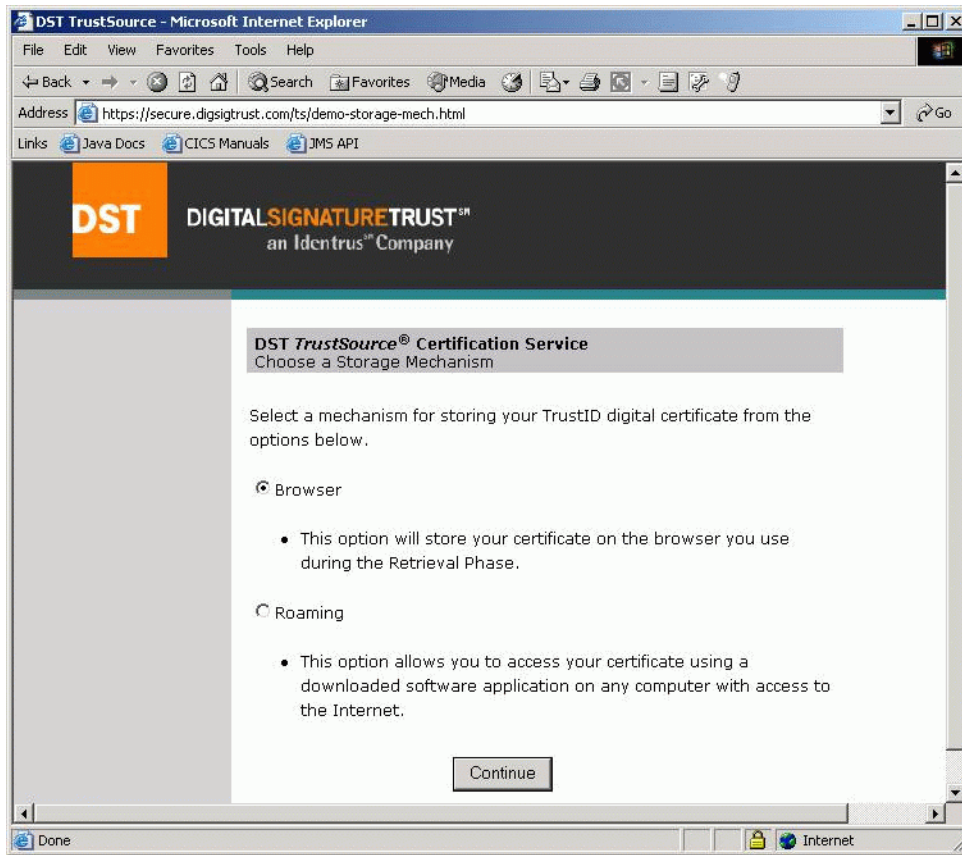


Figure 12-8 Choose storage mechanism

6. Click **Accept** on the **Certificate Agreement**.
7. Accept the defaults: **1024** and **Microsoft Enhanced Cryptographic Provider v1.0**. These values are used to generate public/private key pair (see Figure 12-9). Click **Continue**.

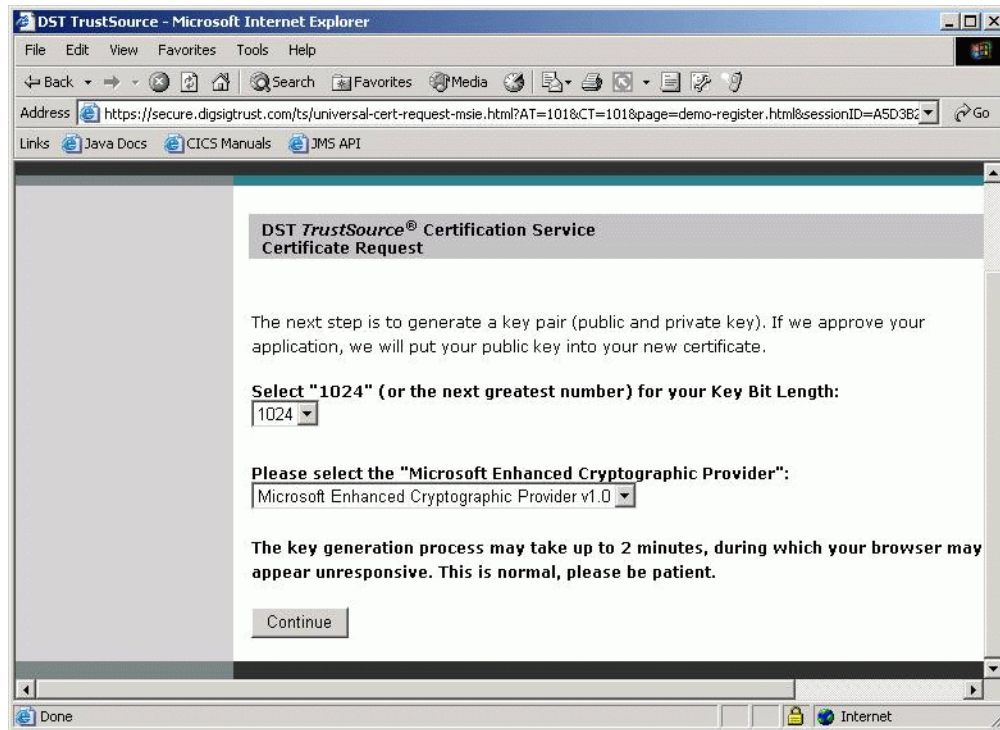


Figure 12-9 Generating the key pair

8. If warning messages are prompted from Internet Explorer indicating that a site is generating a certificate on your behalf, select **Yes** on the message dialogs.
9. The next window confirms that the certificate is being generated. It also prompts for downloading the Digital Signature Trust (DST) Root Certificate. This step is not required because Windows 2000 comes pre-supplied with various Root Certificates from different CAs, including the DST Root Certificate.
10. Check the e-mail for a note from the Digitrust; it contains a URL and activation code as shown in following Example 12-1.

Example 12-1 Certificate mail

Dear BSS REDBOOK,

Your Demo DST TrustSource Certificate request has been approved. Please visit the following web address to retrieve your new certificate. You will be asked to use the following Activation Code and enter the passphrase

you selected when you submitted your request.

<https://secure.digsigtrust.com/ts/retrieve.html?act=3582187993>

Activation Code: 3582187993

TrustSource Certification Service
Digital Signature Trust Co.
Salt Lake City, Utah
trustsource@digsigtrust.com

11. Access the supplied URL using Internet Explorer; supply the passphrase (refer to step 3 above), and click **Retrieve**.
12. Internet Explorer now prompts to add a certificate. Click **Yes** on this message dialog.
13. The resulting status screen confirms that the certificate has been added. Click **Continue**.
14. To view the certificate in Internet Explorer, select **Tools => Internet Options**. On the resulting window, click the **Contents** tab, and then click the **Certificates** button. The Certificates window displays all of the certificates available in the Microsoft Certificate Store. The **Personal** tab view lists in alphabetical order the personal certificates, including the ones just generated (see Figure 12-10).

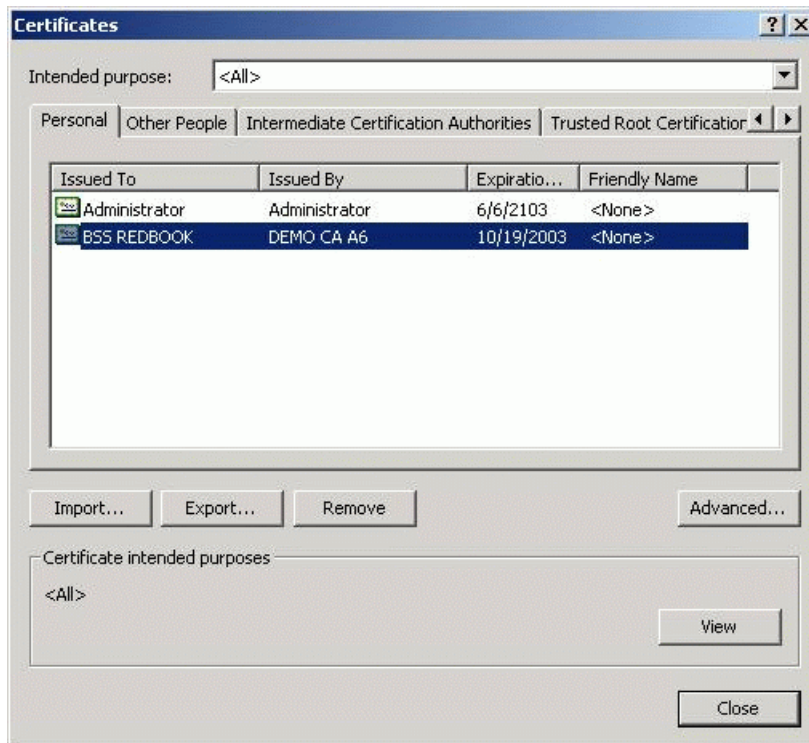


Figure 12-10 Microsoft certificate store

Now the certificate for BSS application is ready. Repeat the steps above, and get certificates for CCS and IAS. The following sections discusses complete deployment of SSL for the business case scenario.

To configure the certificate for a queue manager on the same machine that the certificate was requested, just leave the certificate in the system repository. To use this certificate in another system, click the **Export** button to export the certificate to a file with a password, and import it on the other system. For the convenience of the subsequent sections, the certificate for BSS is exported to a file named *BSS_Cert.pfx* with password *redbook*.

12.7.2 Deploying SSL support in CCS

To deploy SSL support to WebSphere MQ queue manager, requires two steps:

- ▶ Assign a certificate to the certificate repository of the queue manager.
- ▶ Setup the attribute of the channel to be used in intercommunication.

Assign a certificate to the certificate repository of a queue manager

1. Open **WebSphere MQ Services**, extend **WebSphere MQ Services**, right-click queue manager **DOTDP** and choose **Manage SSL Certificates** as shown in Figure 12-11.

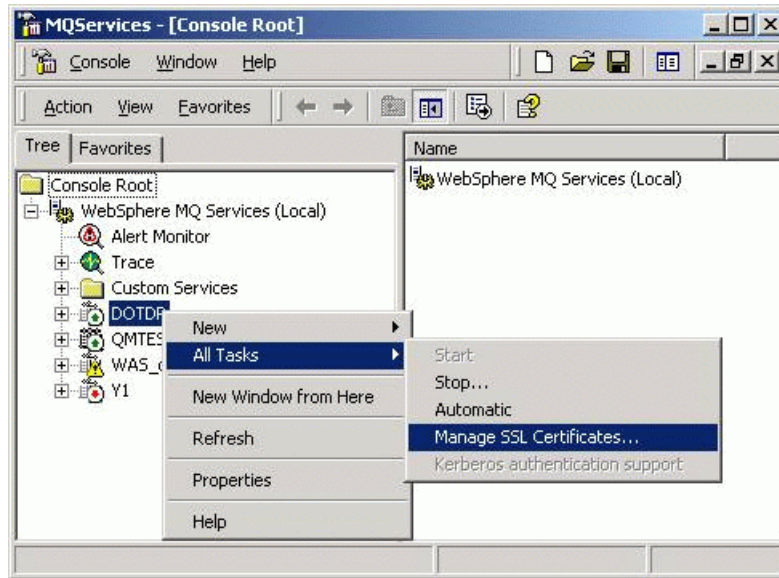


Figure 12-11 Manage SSL certificate

2. Click **Add** at the **Manage SSL Certificates - DOTDP** window, and the **Add Certificate - DOTDP** window appears as shown in Figure 12-12.

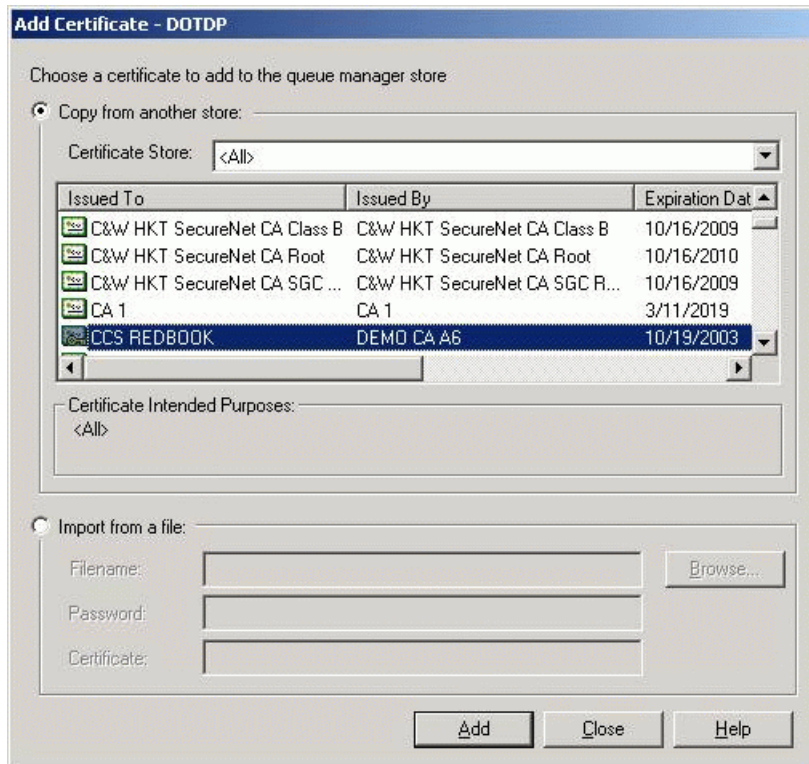


Figure 12-12 Add certificate

3. Select the **CCS REDBOOK** in the certificate list, and click **Add** in the **Add Certificate** window. Then the certificate is added to the queue manager.

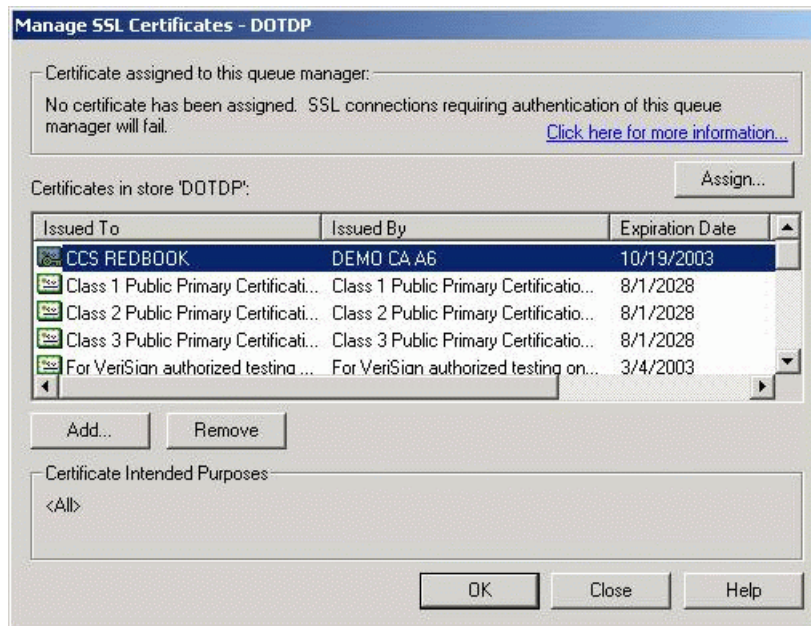


Figure 12-13 Certificate added

- Now, select the certificate from the certificate list and click **Assign** in the window.
- Click **Assign** in the **Assign Queue Manager Certificate** window.

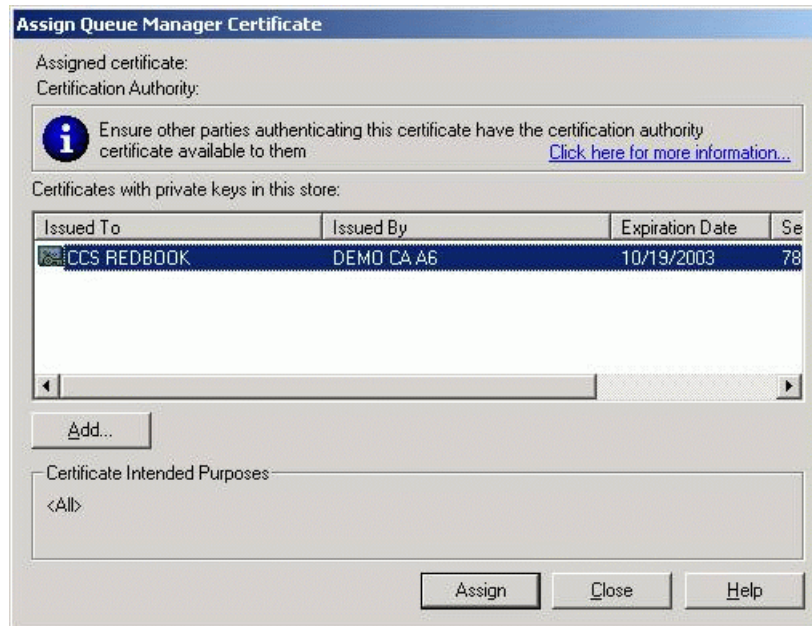


Figure 12-14 Assign queue manager certificate

6. The certificate is now successfully assigned to the queue manager. Click **OK** to exit.

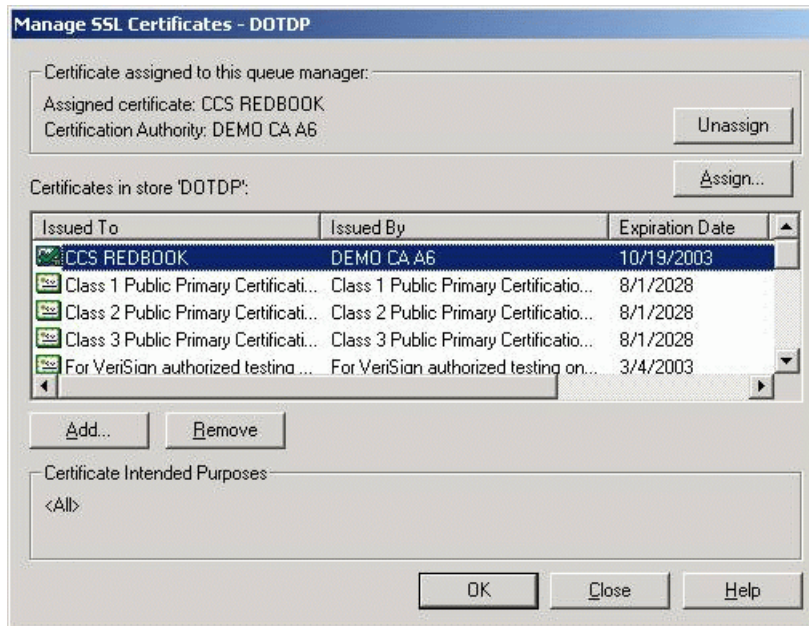


Figure 12-15 Certificate assigned

It is possible that the certificate is not in the system repository, but in a file copied from another machine. Then, in the step 3, choose **Import from file**, provide the path name as well as the password of the certificate file, then click **Add** as shown in Figure 12-16. The remaining steps are the same as above.

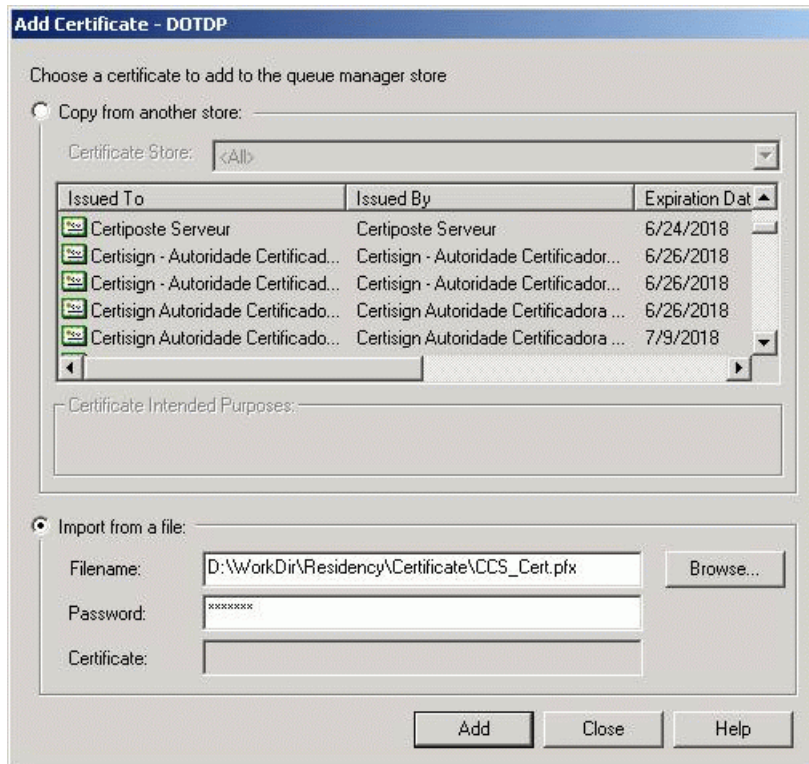


Figure 12-16 Import certificate from file

Equivalent command line steps are as follows:

1. Use the following command to view the list of the personal certificates in the system repository, if the certificate is in the system repository.

```
amqmcert -l -k MY
```

The output should contain:

```
14001: * CCS REDBOOK, DEMO CA A6
```

2. Run the following command to add the certificate to the queue manager:

```
amqmcert -a 14001 -m DOTDP
```

3. If the certificate is not in the system repository, but in a certificate file such as *CCS_Cert.pfx* with password *redbook*, run the following command to add it to the repository of queue manager:

```
amqmcert -a -m DOTDP -p CCS_Cert.pfx -z redbook
```

4. Get the handle of the certificate in store of queue manager by the following command:

```
amqmcert -l -m DOTDP
```

The output should contain:

```
00017: * CCS REDBOOK, DEMO CA A6
```

5. Assign the certificate with

```
amqmcert -d 00017 -m DOTDP
```

For usage of command `amqmcert`, refer to manual *WebSphere MQ System Administration Guide*, SC34-6068.

Setup the attribute of the channel to be used in intercommunication

1. Start a command prompt and run

```
c:\>runmqsc DOTDP
```

2. Run the following script command in the command prompt:

```
alter channel(TO.DOTDP) chltype(SVRCONN) +  
sslcauth(OPTIONAL) sslciph(NULL_MD5)
```

It is possible to use WebSphere MQ Explorer to set the attribute of the server connection channel TO.DOTDP. Refer to *WebSphere MQ System Administration Guide*, SC34-6068 for detail on how to set the attribute of channels.

12.7.3 Deploying SSL support in IAS

The deployment of SLL support on IAS is similar to that on CCS, except for the following differences:

- ▶ The name of the certificate
- ▶ The name of the queue manager
- ▶ The name of the channel

12.7.4 Deploying SSL support in BSS

To deploy SSL support in BSS is a little bit complex. There are three tasks to be performed:

- ▶ Set up the environment
- ▶ Access CCS with SSL support
- ▶ Access IAS with SSL support

Setup the environment

For a client application to use SSL, it must have its own key repository and for it to generate a key repository, a certificate is required. A command utility to

generate a key store is run. The location of the key store is determined by an environment variable MQSSLKEYR. This environment variable is setup as follows:

```
set MQSSLKEYR=C:\SSLCertificates\key
```

A certificate for BSS had been obtained (refer to Obtaining certificates). To add the certificate to that key repository, use the following command:

```
amqmcert -a -p BSS_Cert.pfx -z redbook
```

If the certificate is in the system certificate repository, run

```
amqmcert -l -k MY
```

to get the handle of the certificate. And then run

```
amqmcert -a handle
```

to add the certificate to the key store of queue manager.

The environment variable MQSSLKEYR caused the following key store to be generated:

```
C:\SSLCertificates\key.sto
```

Verify the repository by command:

```
amqmcert -l
```

And it gives the following output:

Example 12-2

```
C:\SSLCertificates>amqmcert -l
5724-B41 (C) Copyright IBM Corp. 1994, 2002.  ALL RIGHTS RESERVED.
Using CURRENT_USER for default system stores.
AMQ4809: No certificate has been assigned to this WebSphere MQ client.
Enumerating Certificate Stores:
```

```
WebSphere MQ Client Store (C:\SSLCertificates\key):
```

```
-----
02001:  Secure Server Certification Authority, Secure Server Certification
Authority
02002:  Thawte Personal Basic CA, Thawte Personal Basic CA
02003:  Thawte Personal Freemail CA, Thawte Personal Freemail CA
02004:  Thawte Personal Premium CA, Thawte Personal Premium CA
02005:  Thawte Premium Server CA, Thawte Premium Server CA
02006:  Thawte Server CA, Thawte Server CA
02007:  VeriSign Class 1 CA Individual Subscriber-Persona Not Validated,
Class 1 Public Primary Certification Authority
```

```
02008: Class 1 Public Primary Certification Authority, Class 1 Public
Primary Certification Authority
02009: VeriSign Class 2 CA - Individual Subscriber, Class 2 Public Primary
Certification Authority
02010: Class 2 Public Primary Certification Authority, Class 2 Public
Primary Certification Authority
02011: www.verisign.com/CPS Incorp.by Ref. LIABILITY LTD.(c)97 VeriSign,
Class 3 Public Primary Certification Authority
02012: Class 3 Public Primary Certification Authority, Class 3 Public
Primary Certification Authority
02013: For VeriSign authorized testing only. No assurances (C)VS1997, For
VeriSign authorized testing only. No assurances (C)VS1997
02014: * BSS REDBOOK, DEMO CA A6
```

Notice from the last line of the output that the certificate has been imported to the repository, and its handle is *02014*.

After add the certificate to the repository, it is required to assign it to the client with the following command:

```
amqmcert -d 02014
```

The certificate is now successfully assigned to the WebSphere MQ Client. The BSS can now take advantage of SSL in the code.

Access CCS with SSL support

The BSS communicates with CCS via WebSphere MQ classes for Microsoft .NET. To connect to the queue manager DOTDP with SSL support, the client code (in C#) accesses this key store with the following code:

The hash method is:

```
ConnectOptions.Add(MQC.SSL_CERT_STORE_PROPERTY,@"C:\SSLCertificates\key");
```

The MQEnvironment method is:

```
MQEnvironment.SSLKeyRepository = @"C:\SSLCertificates\key"
```

Note: In both cases, the extent name of the file of the certificate store, .sto, is omitted.

There are two classes to be added to the C# code to enable SSL support.

```
Using MQEnvironment
Using Hashtable
```

The MQEnvironment typically requires the following code prior to creating the queue manager.

Example 12-3

```
//  
// Connect using Environment and Constructor  
//  
MQEnvironment.Hostname = "ITSOD"; // The machine that the CCS reside  
MQEnvironment.Port = 1414;  
MQEnvironment.Channel = "TO.DOTDP";  
MQEnvironment.SSLKeyRepository = @"C:\SSLCertificates\key";  
MQEnvironment.SSLCipherSpec = "NULL_MD5";  
  
MQQueueManager qm = new MQQueueManager("DOTDP");
```

The HashTable typically requires the following code prior to creating the queue manager.

Example 12-4

```
//  
// Connect using HashTable for connection options  
//  
Hashtable ConnectOptions = new Hashtable();  
ConnectOptions.Add(MQC.CHANNEL_PROPERTY, "TO.DOTDP");  
ConnectOptions.Add(MQC.HOST_NAME_PROPERTY, "ITSOD");  
ConnectOptions.Add(MQC.PORT_PROPERTY, "1414");  
ConnectOptions.Add(MQC.SSL_CERT_STORE_PROPERTY, @"C:\SSLCertificates\key");  
ConnectOptions.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "NULL_MD5");  
  
MQQueueManager qm = new MQQueueManager("DOTDP", ConnectOptions);
```

Access IAS with SSL support

The BSS communicates with IAS via WebSphere MQ Transport for SOAP. To take advantage of SSL in communication with IAS via WebSphere MQ Transport for SOAP, the client application adds the following options into the WebSphere MQ URL:

```
SSLKeyRepository=<key-repository>,SSLCipherSpec="NULL_MD5"
```

For example:

```
wmq:SOAP.StockQuoteDotNet?clientConnection=ITSOI,clientChannel=TO.DOTIP,SSL  
CipherSpec="NULL_MD5",SSLKeyRepository="C:\SSLCertificates\key"
```




Transactions

This chapter explains the fundamental concepts of transactions and how to incorporate transaction support in applications and services. This chapter is structured in three parts.

- ▶ **Local transactions:** This section introduces basic transaction concepts and programming support for basic transactional activities.
- ▶ **Distributed transactions:** This section describes the nature and complexities of transactions in distributed environment.
- ▶ **Web Service transactions:** Standards and protocols are being evolved to support Web Services in more loosely coupled manner. This section gives a glimpse of the future support for distributed transactions.

Discussions in these sections are focused on .NET and IBM WebSphere MQ technologies and standards. The YuBank scenario has been extended to explain practical aspects of transaction management.

Introduction

A business transaction is a set of tasks that either succeed or fail as a unit. Although tasks associated with a transaction are operationally independent; they indeed share a common intent as a unit. By performing only a subset of these operations, the system could compromise the overall intent of the transaction.

To guarantee the “all or nothing” aspect of transactions, the IT systems need to handle participating operations in transactional context within the overall scope. If

a single participant of the transaction fails, all changes to data within the scope of the transaction are rolled back to a previous stage. It is required that all participating operations must guarantee that any change to data is permanent and persist despite system crashes or other unanticipated events.

13.1 Local transactions

Transactions are a very commonly used phenomenon in IT supported systems. For example, in an online order processing system there could be multiple tasks that are logically bound together as one business transaction. Such a transaction in the order processing scenario may include tasks such as selection of item, address validation, payment verification, customer register update, inventory update and so on. If any one of these tasks fails to commit then the changes made to the resources (for example, database, message queues, and file store) must be rolled back to their original state.

The following figure depicts typical order processing transactions involved in an online purchasing operation.

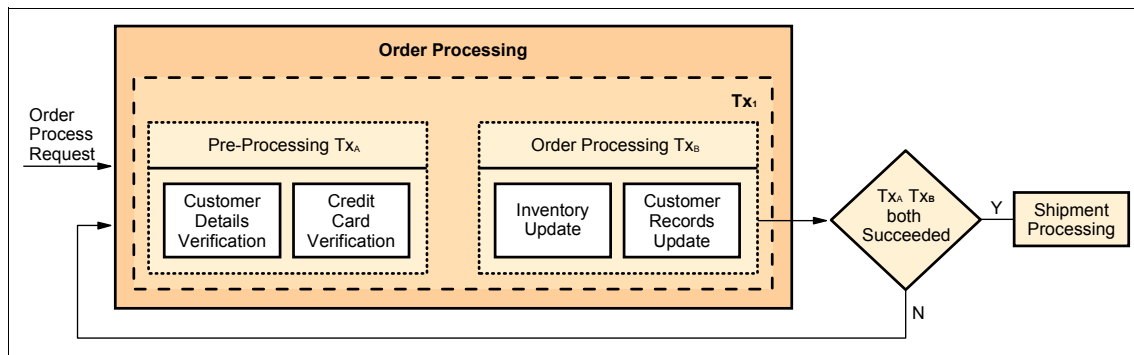


Figure 13-1 Local transaction

13.1.1 ACID properties of a transaction

Transaction behavior properties are known as ACID properties. The term ACID stands for atomicity, consistency, isolation, and durability. The table below portrays a short description of these properties.

Table 13-1 ACID properties

Property	Description
Atomicity	Atomicity property of a transaction guaranties that the transaction executes exactly once and all the work required by its participants is done or none of it is.
Consistency	The semantics, integrity and consistent state of data must be persevered during transformation from pre-transaction to post-transaction phase. (For example travel estimate transaction should not arbitrarily move decimal point of prices quoted by other services).Thanks
Isolation	In case of concurrent transaction processing, Isolation guaranties that the outcome obtained from a set of concurrent transactions are identical to the outcome obtained by running each transaction serially.
Durability	A durable transaction assures that on successful completion, all the updates to the resources made by its operations persists, even if the computer crashes immediately after the commit.

A local transaction allows the local transaction manager to manage resources using their internal resource managers. In other words, if all participating resources (databases, queues, and so on) are running locally then it is possible to manage the transactions using the transaction manager which is internal to the local system. For example, transactions in a COM+ component are managed by the Windows Component Services running on the same server.

The following section describes how to manage transactions locally using API provided by application frameworks.

13.1.2 Programming local transactions

Both .NET and J2EE infrastructures provide libraries for supporting transaction management. Microsoft offers transaction management support in MTS, COM+ and Common language runtime technologies. .NET Framework provides following APIs for handling transactions in common language runtime environment.

```
System.EnterpriseServices
System.Messaging.MessageQueueTransaction
ADO.NET connection objects.
```

Java Transaction API (JTA) provides libraries to handle transactions in Java environment. More information about JTA can be found in the following location:

<http://java.sun.com/products/jta/>

In this section we consider programming local transaction in .NET environment.

Transaction Models in .NET Framework

Microsoft .NET Framework supports two transaction models for objects registered with COM+.

Manual Transaction: In this case transactions are managed explicitly by using API instructions. The developer is responsible for proper handling of transaction related tasks such as begin, commit/rollback, end and so on. Manual transaction model enables total control over the behavior of transaction which is a useful feature in cases like managing nested transaction and linked transactions.

Automatic Transaction: Automatic transactions offer declarative model for managing transactions. Class marked with transaction attribute can participate in an existing transaction, request a new transaction, be the root of a new transaction or never participate in a transaction. One of the advantages of automatic transaction is that the transaction associated with the object automatically flows to the appropriate resource manager (such as ODBC). However, automatic transaction model is not suitable for handling nested transactions.

In this section we explain how to manage local transaction using .NET Framework APIs.

Consider Usage scenarios: .NET application to .NET application explained in Chapter 2., “Overview” on page 3. YuBank is planning to provide an investment service to its customers. One of the application processing steps requires collaboration with credit rating bureau. A credit rating bureau provides credit score based on the applicant's record in the credit history data source. This service is called Credit Check Service (CCS). The CCS involves three major steps:

1. Receive credit check requests in CreditCheck queue on CCS server.
2. Calculate the score based on information stored in credit history database also running on the CCS server.
3. Put credit score reply in CreditScore queue on the same server.

YuBank has access to CreditCheck and CreditScore queues on the CCS server. The credit rating bureau wants to ensure that all these steps occur in a transactional mode. Developers are told to make sure that if any one of these steps fails to complete then the credit check request must be made available back into the queue without causing any change in other systems.

Code example

The following code examples describe the solution. This example shows the manual transaction management.

In our business case scenario, the database is currently a XML file. Whenever the CCS application reads a request message it reads the file and puts a reply to the reply queue. The file is not changed. So WebSphere MQ can be used as a transaction manager for its own resource changes.

The first step in the transaction is getting the request from the input queue. When the get fails, the transaction is rolled back and the request is available on the queue again.

Example 13-1 Getting the request under sync point

```
MQMessage mqMsg;
MQGetMessageOptions mqGetMsgOpts;
mqMsg = new MQMessage();
mqGetMsgOpts = new MQGetMessageOptions();
mqGetMsgOpts.Options = MQC.MQGMO_WAIT + MQC.MQGMO_FAIL_IF_QUIESCING +
    MQC.MQGMO_SYNCPOINT;
try
{
    mqQueue.Get( mqMsg, mqGetMsgOpts );
    ...
}
catch (MQException mqe)
{
    ...
    mqMgr.Backout();
}
```

The next step in the transaction is reading the XML file. If the file is not accessible, the unit of work is rolled back, so the request is available on the queue and can be processed again.

Example 13-2 Reading the XML file

```
try
{
    DatabaseDOM = new XmlDocument();
    DatabaseDOM.Load(xmlFile);
}
catch (Exception ex)
{
    ...
    mqMgr.Backout();
}
```

The third and final step in the transaction is putting the reply on the reply queue. This step has two possible errors. Both the open of the reply queue and the put can fail. If the put is successful we commit the transaction.

Example 13-3 Putting the reply under sync point

```
try
{
    mqQueue = mqMgr.AccessQueue(QNameInComing, MQC.MQOO_OUTPUT +
        MQC.MQOO_FAIL_IF_QUIESCING ,QMgrIncoming,"","");
}
catch (MQException mqe)
{
    ...
mqMgr.Backout();
}
...
MQPutMessageOptions mqPutMsgOpts = new MQPutMessageOptions();
mqPutMsgOpts.Options=MQC.MQPMO_SYNCPOINT;
try
{
    mqQueue.Put( mqMsg, mqPutMsgOpts );
    mqMgr.Commit();
}
catch (MQException mqe)
{
    ...
mqMgr.Backout();
}
```

If the database file or the output queue is not available, the transaction is rolled back and the request is available again. This happens as long as the problem exists and the CCS application is starting and rolling back the transaction for a long time. Therefore, the application has to check whether the request already has been backed out. When the message backout count equals the backout threshold value of the input queue, it has to put it in the backout requeue queue specified. After putting this message into the backout requeue, the transaction is committed.

Example 13-4 Requeue messages

```
if (mqMsg.BackoutCount >= mqQueue.BackoutThreshold)
{
    MQQueue requeueQ;
    MQPutMessageOptions pmo = new MQPutMessageOptions();
    try
    {
        requeueQ = mqMgr.AccessQueue(mqQueue.BackoutRequeueName,
```

```

        MQC.MQOO_OUTPUT);
    pmo.Options = MQC.MQPMO_SYNCPOINT;
    requeueQ.Put(mqMsg, pmo);
    mqQMgr.Commit();
}
catch (MQException mqe)
{
    ...
    mqQMgr.Backout();
}
}

```

The input queue was changed and a requeue queue was created using the following script to support the requeuing mechanism.

Example 13-5 Alter backout parameters

```

define qlocal('TxQueue') replace
alter qlocal('CreditCheck') boqname('TxQueue') bothresh(10)

```

If the business scenario is changed, so that the XML file is updated whenever a request is processed, this solution cannot be used. When the changing of the file fails the whole transaction could still be rolled back. But if the last step of the transaction, the put of the reply, fails the changes to the file could not be rolled back.

If there is the requirement to update the data within the unit of work, this solution is no longer sufficient. There are two possible ways to solve this problem:

- ▶ Use the data base resource manager within the context of WebSphere MQ transaction. This allows the WebSphere MQ transaction manager to coordinate changes to WebSphere MQ and to the database. On Microsoft Windows the following transaction monitors can be used to handle transactions that include WebSphere MQ resources: On Microsoft Windows the following databases can participate in a transaction that is controlled by WebSphere MQ:
 - Oracle
 - DB2® UDB
 - Sybase

For the latest list of supported transaction monitors and databases on Microsoft Windows see the following link:

http://www-3.ibm.com/software/integration/mqfamily/platforms/supported/wsmq_for_winnt2000_5_3.html

- Design the solution in such a way that your database transaction becomes the root transaction. Transaction manager handling the root transaction coordinates the subsequent transactions invoked in the context. Interoperability between resource managers is also an important issue in designing such solutions.

13.2 Distributed transactions

Transactions can also span resources on multiple systems. Distributed transactions allow you to incorporate several distinct operations occurring on different systems into a single transaction.

Distributed transaction processing systems are designed to facilitate transactions that span multiple data sources. Therefore, in a transactional unit you can possibly combine a number of diverse activities such as retrieving a message from a WebSphere MQ queue, storing the message in a database, and forwarding the request to a Web Service.

Note: In the WebSphere MQ manuals the terms “local units of work” and “global units of work” are used. The terms “local transaction” and “distributed transaction” used in DTC context however mean something slightly different.

► **WebSphere MQ terms**

Local unit of work: A local unit of work only updates resources of the WebSphere MQ queue manager.

Global unit of work: A global unit of work updates resources of the WebSphere MQ queue manager and of other resource managers, like databases. These global units of work can be coordinated by the queue manager or by an external transaction manager.

► **DTC terms**

Local transaction: A local transaction updates resources of one or more resource manager, like WebSphere MQ or DB2 UDB. All resource managers run on the same machine.

Distributed transaction: A distributed transaction updates resources of several resource managers. The resources managers run on different machines.

So a local transaction is also a local unit of work, as long as the WebSphere MQ queue manager is the only resource manager involved. A local transaction is a global unit of work, when also other resource managers take part in the transaction.

13.2.1 Transaction support under Windows 2000

Distributed transactions work across multiple systems and data sources. The job of enforcing ACID properties therefore requires special infrastructure.

Under Windows 2000, Microsoft Component Services provide such an infrastructure for transaction management. Component Services are comprised of two services; COM+ and Microsoft Distributed Transaction Coordinator (DTC).

The DTC is the Transaction Processing (TP) monitor for Microsoft Windows 2000. DTC operates between a transaction-aware (COM+) application and a set of resources. The role of DTC is to streamline network communications and to connect multiple clients to multiple applications that potentially access multiple data resources.

In a distributed transaction, each participating resource (for example SQL database, WebSphere MQ) has a Transactional Resource Manager (RM) running to track incoming and outgoing transactions on the resource. When a

resource participates in a transaction the associated resource manager enlists itself in a transaction with Transaction Manager (TM).

The following figure explains how DTC works with resource managers.

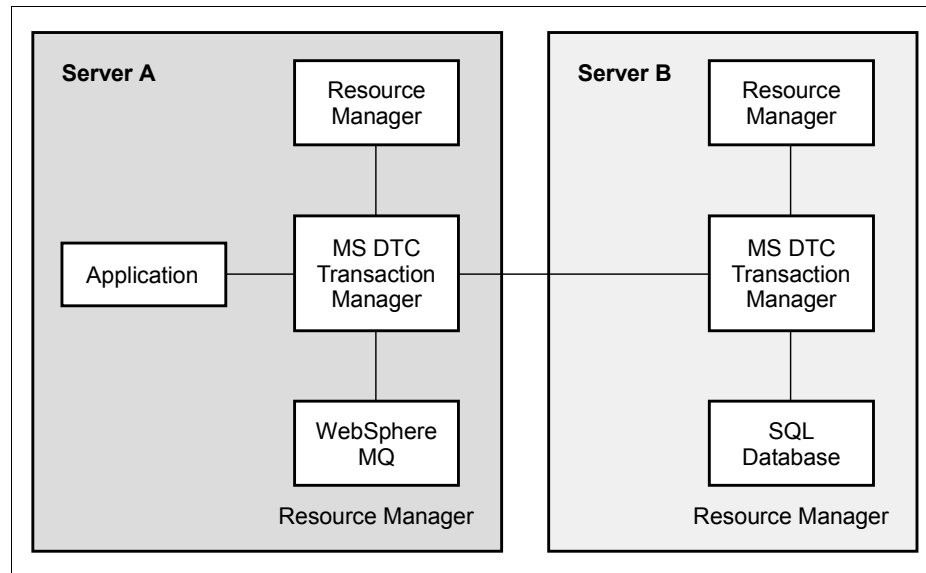


Figure 13-2 Coordination of distributed resource managers using DTC

Microsoft DTC transaction manager coordinates the resource managers. On receiving a transaction request from an application, the transaction manager initiates a transaction and enlists the resource managers in a transaction during the scope of transaction. In distributed applications where resource managers are running on different servers, Microsoft DTC coordinates transactions using two phase commit protocol (described later in this section).

The following figure shows Microsoft Management Console (MMC) snap in for Component Services Administration which displays details of running local and distributed transactions.

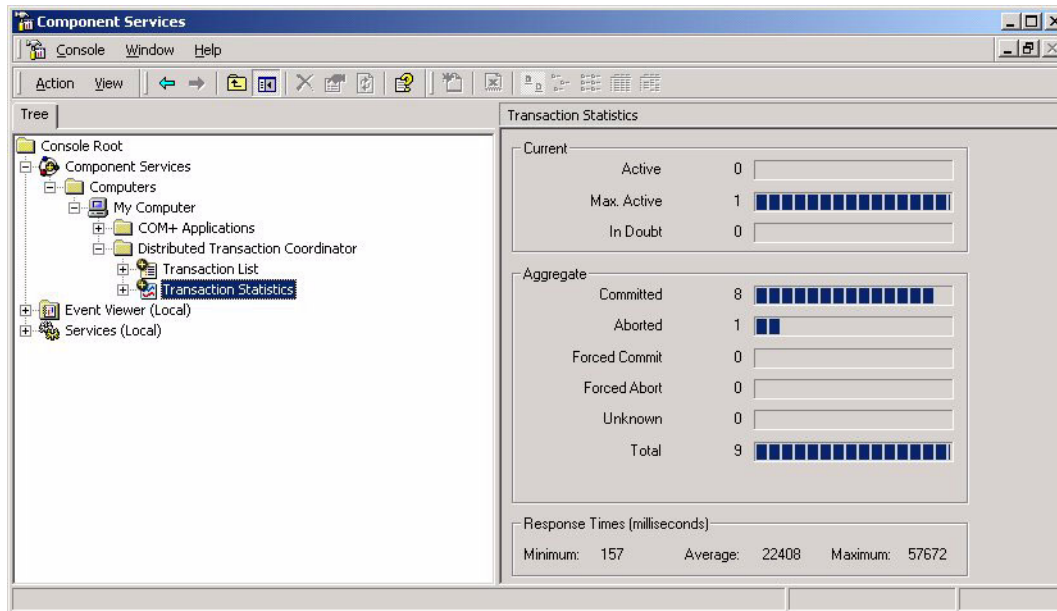


Figure 13-3 Distributed transactions management using component administration MMC in Windows 2000

To work with Microsoft DTC transaction manager various resource vendors are required to provide a compliant Resource Manager for handling data persistence, durability and recovery associated with the resource. IBM provides DTC compatible resource managers for DB2, WebSphere MQ, CICS®, and so on.

WebSphere MQ applications can participate in transactions managed by Microsoft transaction managers. The application either has to be directly connected to a queue manager, or it has to use the extended transactional client function when running on a remote machine.

More information about WebSphere MQ transactional client can be found in the manual *WebSphere MQ Extended Transactional Clients*, SC34-6275 and under:

<http://www-3.ibm.com/software/integration/wmq/transclient.html>

What is two-phase commit?

The intention of this discussion is to give brief understanding of two phase commit (2PC) protocol, one of key concepts in distributed transaction management. You can safely ignore this subsection as 2PC is managed internally by tools described in this redbook, and it is unlikely to affect any code that you may require to write.

In a local transaction where there is only one resource manager involved in the process, the resource manager (for example a DB2 UDD) handles commit or roll back processes of a transaction.

Distributed transactions, however involve two or more resource managers (for example two databases and a WebSphere MQ queue). To ensure atomicity of the transaction within and across resources, the transaction managers need to coordinate all resource managers requested by the application.

Transaction Process Managers (TPM) like Microsoft DTC achieve this task by using two phase commits (2PC) protocol.

In the first phase of this protocol, the transaction manager sends a "Prepare" message to each resource manager, asking if it is ready and able to commit the transaction;. At this stage the coordinator enters in the "wait" state. This message also contains a unique Transaction ID (TID), which is used in all further messages in this protocol when run. If the coordinator receives an affirmative reply from all the resource managers, it results into a commit vote in the log.

After the coordinator has received responses from all resource managers, it decides whether to commit or abort according to the global commit rule, and writes this decision in the log.

This area of distributed transactions has well matured over the years and all major transaction managers including Microsoft DTC, IBM CICS adhere to this de-facto standard. Developers do not need to write any code to implement the two phase commit protocol in their distributed application.

13.2.2 Programming distributed transactions: Credit Check Service

Consider the same credit check application described in the previous section. The credit rating bureau has expanded its operations in recent years. It now serves more than two hundred customers nationwide. A one to one integration with their WebSphere MQ is therefore no longer a feasible solution. The company decides to setup a dedicated WebSphere MQ server for receiving credit check requests from its customers. Also, to meet the new privacy regulations, the company decides to utilize a third party service for secured hosting of credit history data source.

The following figure shows the distributed CCS deployment.

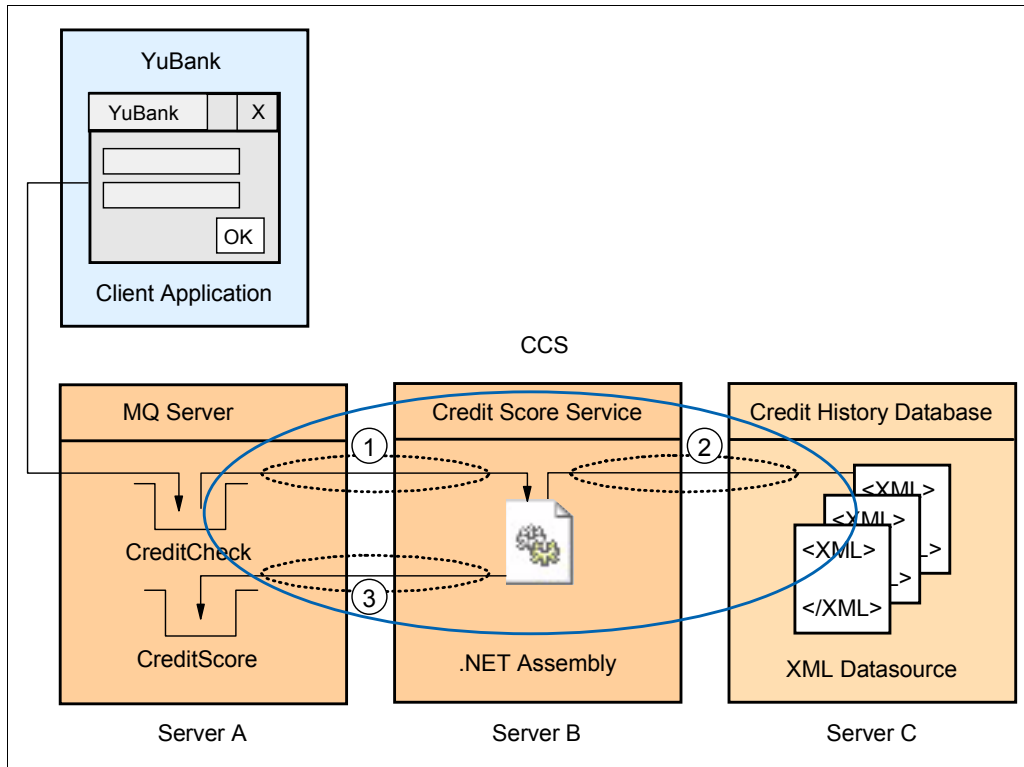


Figure 13-4 Credit check service in distributed environment

The developers of the credit check system are concerned about the transaction support as their original solution is inapplicable in a proposed loosely coupled distributed environment. They decide to incorporate distributed transaction to resolve this issue.

The Credit check service (CCS) exposes its WebSphere MQ server to YuBank as the receiving system for the credit check applications. YuBank submits the application in the designated queue on CCS WebSphere MQ server (CreditCheck on ITSOD). On the application server of the bureau, as shown in Figure 13-4 credit checks that the application is monitoring the CreditCheck queue. The credit check application collects the request and starts processing it.

At this stage, the credit check application invokes a distributed transaction. The scope of this distributed transaction covers the following three operations:

1. Get the credit check request message from the CreditCheck queue located on Server A.
2. Access and update the credit history data source running on Server C.

3. Put the credit score reply back on the CreditScore reply queue on Server A.

There are two possible solutions for this scenario:

- ▶ Linked transactions using manual transaction management
- ▶ Automatic transactions using .NET declarative transaction management

Linked transactions using manual transaction management

Sub transactions involved in this scenario are shown in Figure 13-4 on page 295. These transactions are logically linked and the outcome of entire process depends on the outcome of individual transactions. These transactions are:

1. Get message from WebSphere MQ queue.
2. Update table in SQL server database.
3. Submit message in the WebSphere MQqueue.

Using manual transaction management methodology it is possible to correlate outcomes of the three transactions and complete them as one unit of work. Designing such linked transactions is a difficult task. It is very important to devise a comprehensive test plan before implementing nested transactions. The following code fragment shows the transaction logic where WebSphere MQ operations and SQL server updates are rolled back if the *submit reply to queue* operation failed.

Example 13-6 Managing linked transactions

```
...
mqPutMsgOpts.Options=MQC.MQPMO_SYNCPOINT;

try
{
    mqQueue.Put( mqMsg, mqPutMsgOpts );
    mqMgr.Commit();
    Console.WriteLine("MQ Manager Committed");

    sqltrx.Commit();
    Console.WriteLine("Sales Entry committed");
}
catch (MQException mqe)
{
    // report the error
    System.Console.WriteLine( "MQQueue::Put ended with " + mqe.Message );
    MessageSent = false;
    mqMgr.Backout();
    Console.WriteLine("MQ Manager rolled back");

    sqltrx.Rollback();
}
```

```

        myCommand.Connection.Close();
        Console.WriteLine("Sales Entry rolled back");
    }
    finally
    {
        mqQueue.Close();
        myConnection.Close();
    }
    ...

```

Remember that as shown in Figure 13-4 on page 295 mqQueue.Put() operation is executed on server A where SQL transaction assqltrx takes place on Server C.

Automatic transactions using .NET declarative transaction management

In this example we redesigned the application as a Windows Serviced component. Serviced component architecture extends COM component model in .NET environment. Service component utilize COM+ services such as transaction management, object pooling, just-in-time (JIT) activation and so on. To learn more about how to create serviced components using Microsoft Visual Studio .NET, refer to *COM and .NET Component Services*, by Juval Löwy.

To take advantage of COM+ transaction services the class is required to be marked with Transaction attribute. Transaction begins when any method call is made on the object. The following code example shows the declaration of a service component and the use of transaction attribute

Example 13-7 Serviced component using transaction attribute

```

[Transaction(TransactionOption.Required)]
public class accessMysql : ServicedComponent
{
    ...
}

```

[AutoComplete] attribute

To take advantage of auto completion feature of COM+ methods of the serviced components need to use AutoComplete attribute. AutoComplete attribute utilizes auto-deactivation facility of COM+. This attributes instructs COM+ to deactivate the object when a method carrying AutoComplete attribute returns. Methods that use the AutoComplete attribute do not need to vote explicitly on their transaction outcome. The operations of the method are committed on successful completion.

If the method throws an exception all the operation within the scope of the method are rolled back.

For more information about serviced components and using transaction related attributes, refer to Microsoft Developer Network (MSDN) library at:

<http://msdn.microsoft.com/library/default.asp>

Unlike previous implementation, in declarative model of transaction management we do not use any explicit transaction instructions in the our code. The following code example shows series of operation executed in a transactional context.

Example 13-8 Automatic transaction management in serviced component

```
...
[AutoComplete]
public bool Process()
{
    if(Connect())
    {
        GetMQMsg();

        Put();
        Clear();
        return true;
    }
    return false;
}

private bool GetMQMsg()
{
    ...
    mqQueue.Get( mqMsg, mqGetMsgOpts );
    string xmlMessage = mqMsg.ReadString(mqMsg.MessageLength);
    MessageIDIncoming = mqMsg.MessageId;
    QNameIncoming = mqMsg.ReplyToQueueName;
    QMgrIncoming = mqMsg.ReplyToQueueManagerName;
    InsertDBEntry();
}

private bool InsertDBEntry()
{
    //connect to db
    SqlConnection myConnection = new SqlConnection(@"workstation id=ITS0D;packet
size=4096;integrated security=false;data source='ITS0E';User
ID='xx';Pwd='xxx';persist security info=False;initial catalog=CCS");
    myConnection.Open();
}
```

```

string myInsertQuery = "INSERT INTO account (BankID, Date, CustomerID)
Values('YuBank', '11/11/03', 'Sachin')";
SqlCommand myCommand = new SqlCommand(myInsertQuery);
myCommand.Connection = myConnection;
myCommand.ExecuteNonQuery();

...
}

```

As illustrated in Figure 13-4 on page 295 the resources involved in this scenario are distributed over three servers. In this example code, the Put operation only takes place if the GetMsg operation on Server A AND InsertDBEntry operation on Server C succeeded. However the changes made on CreditCheck queue on Server A and Credit History database on Server C are not committed unless the Put operation on the CreditScore queue is successfully completed. This coordination and transaction outcome handling is achieved by automatic transaction facility of serviced component and COM+.

Using Component Services Manager

During runtime, distributed transactions can be controlled using Component Services Manager, a Microsoft Management Console (MMC) on Windows 2000. For example because of some unhandled error, if Server C running the customer history database does not respond it is possible to resolve the transaction using component services manager. The following figure shows the transactions in MSQLTrx process.

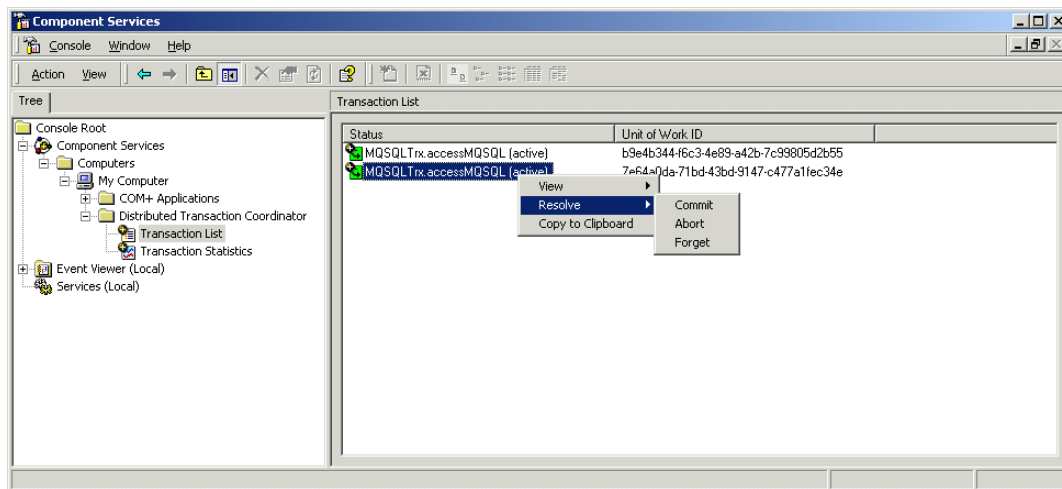


Figure 13-5 Resolving distributed transactions using component services MMC

Note: We recommend that you to examine the known issues with DTC before developing transactional components involving resource managers from different vendors. Knowledge base articles about MS DTC on Windows 2000 can be found at the following location:

<http://support.microsoft.com/search/default.aspx?Query=DTC+windows+2000>

13.2.3 Microsoft Transaction Server: MTS and WebSphere MQ

Microsoft Transaction Server (MTS) is designed to support transactions and distributed processing in heterogeneous environments. MTS works in conjunction with Microsoft DTC under Windows NT®, Windows 95 and Windows 98 environments. This technology has been superseded by COM+ services in the next generation Microsoft systems such as Windows 2000, Windows XP, and Windows Server 2003.

WebSphere MQ provides support for MTS. When you use WebSphere MQ with MTS, WebSphere MQ resources (queue managers) are coordinated by DTC along with the resources of other resource managers involved in a transaction (such as SQL Server and DB2 UDB).

A detailed discussion about WebSphere MQ support for MTS is outside the scope of this redbook. For further information about this topic, refer to the topic “Programming\MTS” in the WebSphere MQ Help Center (installed with the product), the *WebSphere MQ System Administration Guide*, SC34-6068, and the *WebSphere MQ Application Programming Guide*, SC34-6064.

WebSphere MQ V5.3 for Windows also supports other transaction managers, like TXSeries and WebSphere Application Server. A full list of the supported transaction managers can be found under:

http://www-3.ibm.com/software/integration/mqfamily/platforms/supported/wsmq_for_winnt2000_5_3.html

13.3 Web Service transactions

In the previous section we learnt about transactions in local environment and how to handle distributed transactions. Web Services bring a new dimension to distributed application development patterns. Web Services are more loosely coupled than traditional distributed applications. Support systems that were designed to manage traditional object-oriented architectures now need rethinking in service-oriented world of Web Services. Transaction support is one such service that requires reinvestigation. This section describes the current

approaches and emerging specification for supporting Web Services transactions.

13.3.1 .NET Web Services and transactions

In Microsoft .NET environment, XML Web Services defined using ASP.NET support automatic transactions on Microsoft Windows 2000 or later. Microsoft uses declarative style for marking transactional Web objects. Transaction directives can be inserted into any .NET Web application, Web Service or a class. Similar to the serviced components described previously in this chapter, in ASP.NET Web Service, transaction directives instruct the Web Service to participate in an existing transaction, begin a new transaction, or never participate in a transaction.

Transaction metadata described in form of directives represent the logic of a unit of work. A physical transaction occurs when a transactional object accesses a data resource, such as a database or message queue. Transaction metadata is then translated into lower level driver instructions.

It is important to note that the scope of automatic transaction support (in .NET as well as in other frameworks) is limited to the operational boundaries of the current Web Service. The Web Service can directly utilize distributed data resources via resource managers with automatic transaction support. However, the transaction context is lost if the Web Service uses another Web Service to utilize distributed resources. At the time of writing this redbook there is no support (implementation) available for sharing transaction context between Web Services. Refer to 13.3.3, “WS Transaction” on page 303 for more details.

The next section describes an example of .NET transactional Web Service.

13.3.2 Programming Web Services transaction in .NET environment

The following code snippet shows how to declare an automatic transaction by using the “TransactionOption property” of the “WebMethodAttribute” attribute class. Setting the “TransactionOption property” to “TransactionOption.RequiresNew” begins a new transaction each time an XML Web Service client invokes the XML Web Service method.

Example 13-9 TransactionOption property

```
[WebMethod (TransactionOption=TransactionOption.RequiresNew)]
public string GetAdvice(string accountId, string investAmt)
{
    ...
}
```

It is possible to control an object's transactional behavior by setting a transaction attribute value on a page, in an XML Web Service method, or in a class as follows.

Example 13-10 Page transaction attribute

```
<%@ Page Transaction="Required" %>
```

Note: The syntax to declare the transaction attribute varies slightly in a .NET Framework class, an ASP.NET page, and an XML Web Service method.

As described in “Automatic transactions using .NET declarative transaction management” on page 297 .NET Framework provides AutoComplete attributes for handling automatic transactions using COM+ services. Same AutoComplete attributes can be applied to methods in a Web Service class.

Class marked with transactional attributes can invoke operations on various data sources, message queues running in distributed environment. The process involved in programming such distributed transactions in a Web Service is similar to transactions in serviced component. Refer to 13.2.2, “Programming distributed transactions: Credit Check Service” on page 294 for more details.

Windows 2000 uses COM+ services to manage transaction management in ASP.NET Web Service. The following figure shows Component Services console for controlling running ASP.NET Web Service transactions.

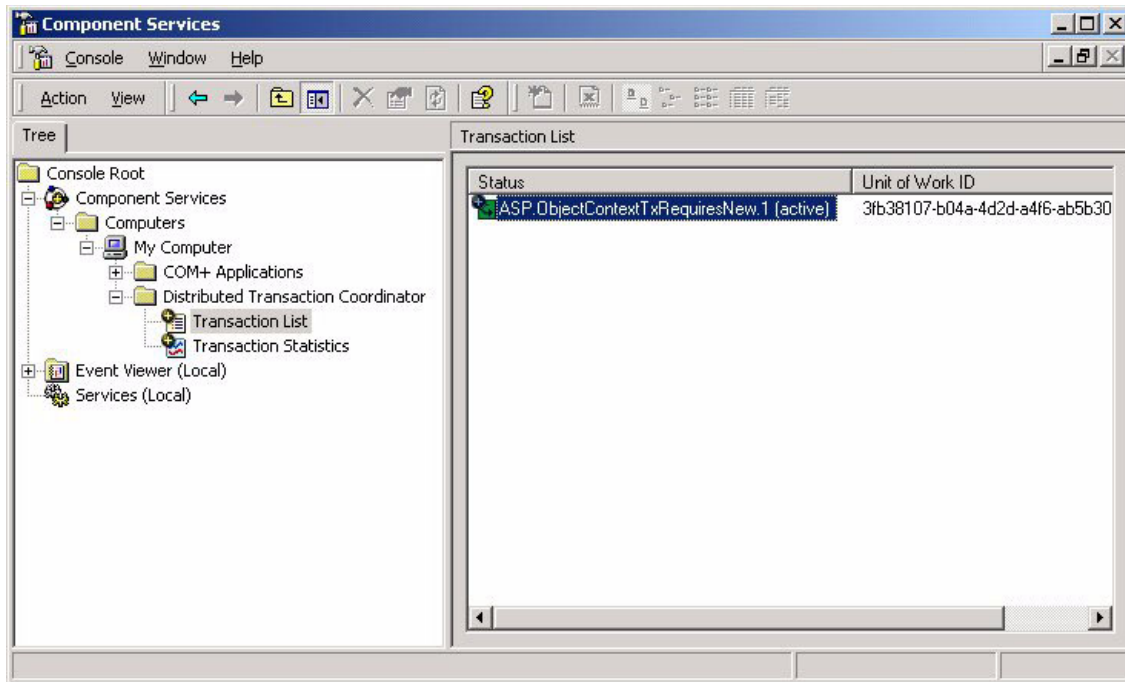


Figure 13-6 Microsoft .NET Web Service participating in a distributed transaction

13.3.3 WS Transaction

As per the method described in the previous section, a Web Service transaction guarantees atomicity of operation within the scope of the Web Services, pretty much same as local transactions. However, the Web Service being called requires invoking other Web Services as part of its operation therefore, the existing distributed transaction coordinator (for example DTC) technology does not provide an adequate solution. For example in .NET, a Web Service transaction begins only when the invoking Web Service method, the method called from the client, has transaction metadata. If the caller Web Service method does not carry the appropriate transaction directives, subsequent Web Services (methods) can neither participate in an existing transaction nor begin a new transaction.

Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP) defined protocols enable basic Web Service inter operability. This presents an opportunity for developers to build federated service-oriented solutions by combining a large number of participating Web Services. A typical example of such federated service is a travel portal combining services from airline companies, travel agents, accommodation portal, car rental companies,

calendar solution, weather service, travel goods and book shop service and so on. As you can imagine, the resulting activities can be complex in structure, transactional in nature and multifaceted in terms of relationships between their participants.

At the time of writing this redbook, SOAP or WSDL specification lacks constructs for defining the coordination between the Web Services, transaction context, service level agreement between services, and so on.

The WS Transaction (see link below) specification proposed by IBM, Microsoft and BEA defines an extensible framework for coordinating different roles that Web Services play in the federated activity. WS Transaction defines two models for transactions over Web Services: Atomic Transactions (AT) where duration is short and level of protection required is high; and Business Activity (BA) transactions for long lived and scalable operation. WS Transaction also defines how Web Services register their intent to use two phase commit and how they communicate transaction status votes (for example readiness, vetoing) with other Web Services.

It is envisaged that existing transaction processing systems (for example COM+, DTC) extends their proprietary protocols to incorporate WS Transaction and inter operate across different vendors and their Web Services infrastructure.

To establish the necessary relationships between participants, the messages exchanged between Web Services carry a "CoordinationContext". It also has a reference to a coordination service. Participants then register for supported coordination protocols and subsequently ensure transactional processing of the message as required.

WS Transaction and WS Coordination specifications are supplementary protocols for the proposed Web Services workflow standard called Business Process Execution Language (WS-BPEL). WS-BPEL is currently going through the standardization process within Organization for the Advancement of Structured Information Standards (OASIS) and will soon be incorporated in related product suit from Microsoft (BizTalk Server Technologies) and IBM (WebSphere Application Server).

You can find more information about the current status of these specifications from this Web site:

<http://www.oasis-open.org/committees/wsbpel>



Best practices

This chapter highlights techniques that help implement effective programs, eliminate common errors, ease long term maintenance and increase the chances for solution extensibility. It also contains solutions for integration issues, common errors, hints and tips as well as the new coding standards for the .NET technology.

14.1 Coding standards

It is advisable to adhere to the coding standards when writing code in different languages. It is worth noting that with the introduction of .NET and the language C# in particular a new set of coding standards have been defined. For more information see:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconclassmemberusageguidelines.asp>

14.2 Hints and tips

The following section provides coding guidance.

14.2.1 XML style comments

For ease of documentation, you can place three slashes above a C# function. This results in an XML comment fragment generation which can later be extracted to document the code.

Example 14-1 XML comment auto generation

```
/// <summary>
///
/// </summary>
/// <param name="camelArgument"></param>
/// <returns></returns>
public bool SomeFunction(string camelArgument)
{
    return true;
}
```

These are typically filled out as follows:

Example 14-2 How to enter XML comments

```
/// <summary>
/// This routine is a sample routine to demonstrate the
/// use of XML style of comments
/// </summary>
/// <param name="camelArgument">This argument is an incoming parameter</param>
/// <returns>The function always returns true</returns>
public bool SomeFunction(string camelArgument)
{
    return true;
}
```

14.2.2 XML processing in Java

There are a variety of ways to construct and parse XML data in a Java program. Coding in both the Simple API for XML (SAX) and the Document Object Model (DOM) can be tedious. JDOM provides a robust, light-weight means of reading and writing XML data without the complex and memory consumptive options that current API offerings provide. It is completely natural API for Java developers, and it provides a low-cost entry point for manipulating XML.

14.2.3 SOAP processing in Java

There are several toolkits available for SOAP processing in a Java program. Apache Axis is an open-source SOAP engine, a framework for constructing SOAP processors. It passed Sun's JAX-RPC and SAAJ compliance tests. Axis can be integrated into existing applications for SOAP processing, and expose the existing methods into Web Services. The SupportPac, WebSphere MQ Transport for SOAP (MA0R), uses Axis for SOAP processing.

14.2.4 XML element versus attribute

In defining an XML schema, there is the decision to use an elements or attributes to represent the data under the root node. There are pros and cons in either way. As a general rule of thumb, attributes are good to describe the integral characteristics of an element, and typically the contents (values) are short. But it is hard to expand an attribute to include sub-contents, if the data structure has to be expanded in the future. Such changes require more code modifications than an element-based structure.

14.3 Common errors

While using WebSphere MQ Transport for SOAP and WebSphere MQ classes for Microsoft .NET in coding .NET applications which send messages via WebSphere MQ, encounter some of the following problems:

- ▶ Runtime errors such as: c:\inetpub\wwwroot\BSS\CreditCheckRequest.cs(2): The type or namespace name 'IBM' could not be found (are you missing a using directive or an assembly reference?) as shown below:

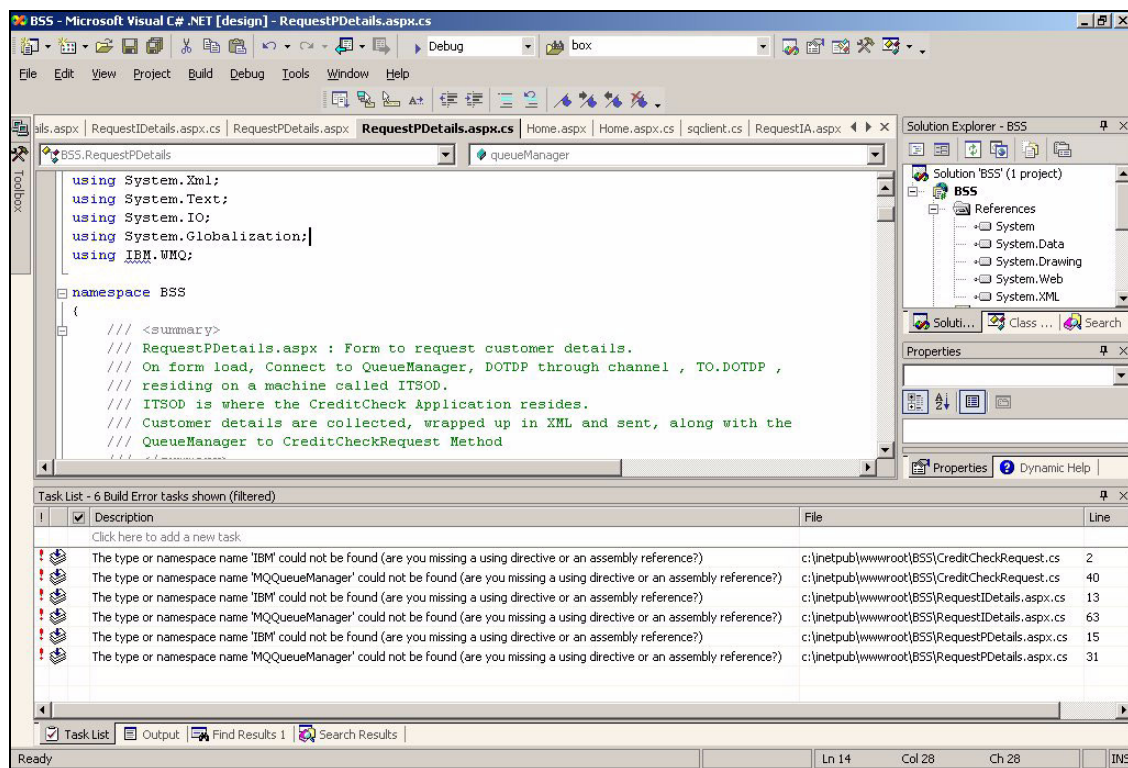


Figure 14-1 Missing assembly reference

In this case, the reference to the WebSphere MQ classes for Microsoft .NET needs to be added to your project. This is achieved by selecting **Project -> Add Reference**.

On the **COM** tab, browse to the location of `amqmdnet.dll` or WebSphere MQ Transport for SOAP — typically located in `C:\Program Files\IBM\WebSphere MQ\bin` and `C:\<ma0r installation directory path>\ma0r\bin`.

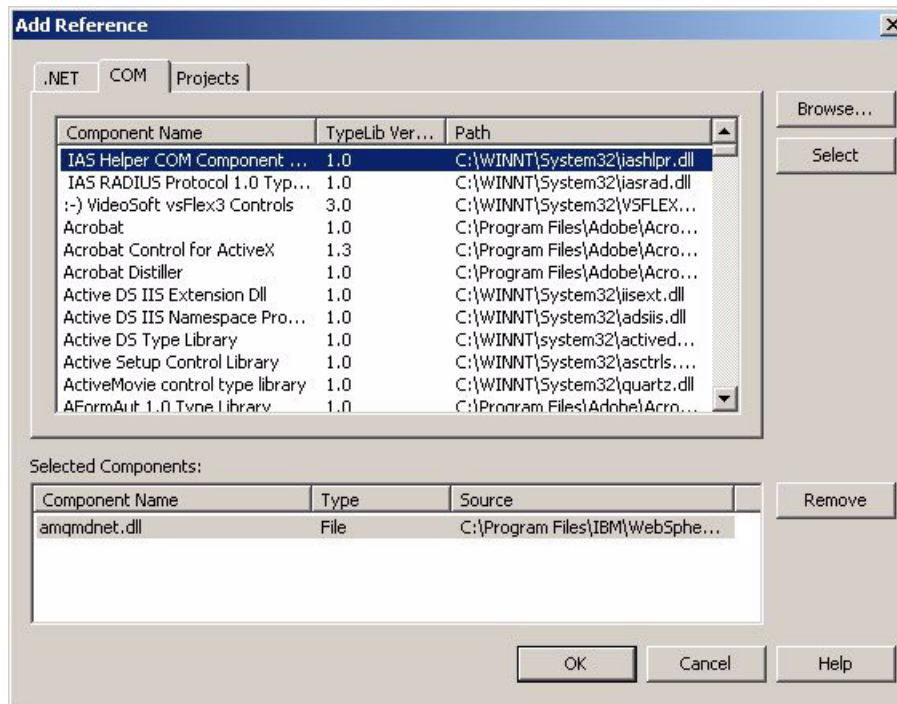


Figure 14-2 Adding missing assembly reference

Select **OK** to add amqmdnet.dll or MQSOAP.dll if you have added MQSOAP.dll, as reference to your project.

Rebuild the project.

► MQExceptions, such as:

- MQException: compCode: 2 Reason: 2059

This happens when the queue manager being accessed is unavailable. For a description of the error, type **MQRC <Reason code>** into a command prompt as shown below:

```

C:\>MQRC 2059
      2059  0x0000000b  MQRC_Q_MGR_NOT_AVAILABLE
C:\>

```

Figure 14-3 WebSphere MQ reason code description

To rectify this problem:

If the queue manager to be accessed is on another machine, ensure the other machine is accessible by testing a network connection to the machine. This is achieved using **ping <machine name>** in a command prompt as shown below:

```

C:\>ping ITS01
Pinging ITS01 [9.20.13.56] with 32 bytes of data:
Reply from 9.20.13.56: bytes=32 time<10ms TTL=128
Reply from 9.20.13.56: bytes=32 time<10ms TTL=128
Reply from 9.20.13.56: bytes=32 time<10ms TTL=128
Reply from 9.20.13.56: bytes=32 time<10ms TTL=128

Ping statistics for 9.20.13.56:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
C:\>

```

Figure 14-4 Testing network connection

If there is a reply then the connection exists otherwise check the network connections.

- Ensure the queue manager name referred to in the code is spelled correctly and that if the queue manager is on another machine, ensure the channel name and host name referred to is specified as shown below:

```

MQQueueManager QM = new MQQueueManager(<QueueManager name>,<channel
name>,<hostname>);
queueManager = new MQQueueManager("DOTDP","TO.DOTDP","ITSOD");

```

- Ensure the queue manager is started.

- MQException: compCode: 2 Reason: 2035. This happens when you are not authorized to connect to the queue manager being accessed. To obtain a description of the error, type **MQRC <Reason code>** into a command prompt as shown in Figure 14-3 on page 310

To rectify this problem:

- Add ASPNET user to the mqm user group by right-clicking on My Computer.

Click **Manage** -> **Local Users and Groups**.

Right-click **ASPNET**, select **Properties** -> the **Member Of** tab as shown below:

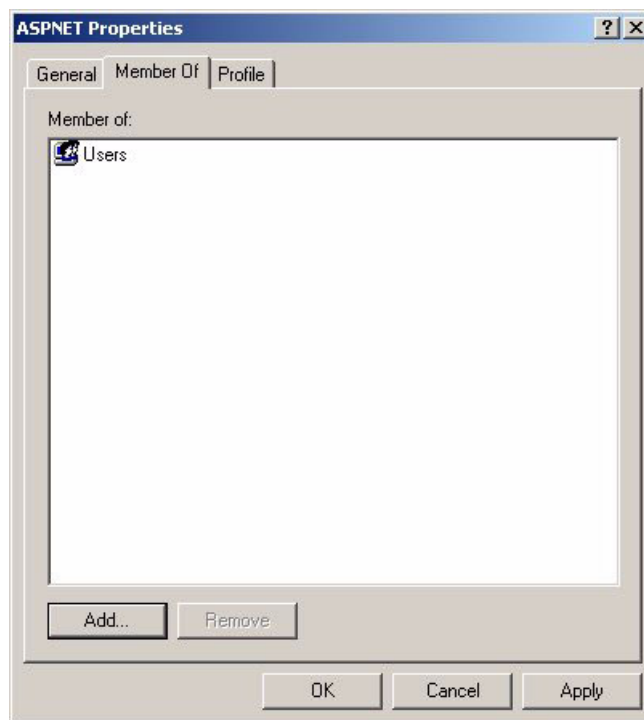


Figure 14-5 Adding ASPNET user to mqm group

- Click **Add** and in the Select Groups window, and select mqm from the list as shown below:

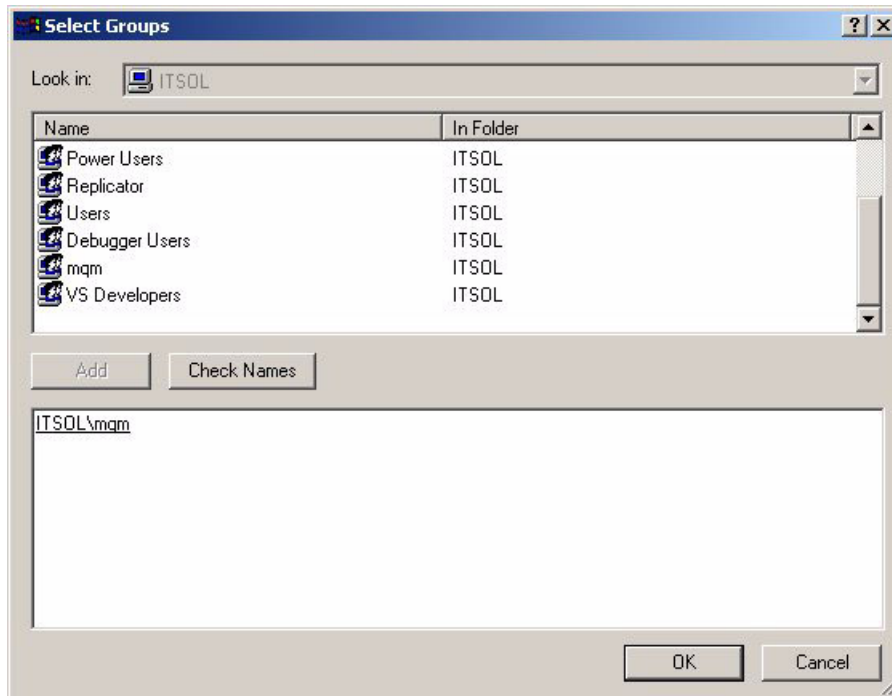


Figure 14-6 Adding ASPNET user to mqm group

- Click **OK**, then click **Apply**.

Note that close may take time for the changes to take effect, so it is advisable to select Apply before Close.

- Java run time errors such as:

Exception in thread "main" java.lang.NoClassDefFoundError: CustomerProfile (wrong name: cps/CustomerProfile) or

Or errors such as:

Exception in thread "main" java.lang.NoClassDefFoundError:
javax/jms/MessageConsumer

This indicates there are jar files and locations missing from the class path. Re-deploy the solution as instructed in the deployment section of the usage scenario solutions in Chapters 7, 8, 9, and 10.

14.4 Testing

As a developer, testing is essential and should be done formally during the software development process. Formal testing, both unit and system, needs to be an integral part of the development process.

Unit tests not only provide a set of coded use cases for documentation of classes, they also represent the most practical design possible. There are various unit testing techniques described briefly below:

- ▶ JUnit
- ▶ NUnit
- ▶ csUnit

14.4.1 Unit Testing with JUnit

JUnit is a regression testing framework written by Erich Gamma and Kent Beck. It is used by developers who implement unit tests in Java. JUnit is Open Source Software, released under the IBM Common Public License Version 1.0 and hosted on SourceForge.

You can find JUnit at:

<http://sourceforge.net/projects/junit/>

It is specifically targeted at Java developers.

14.4.2 Unit Testing with NUnit

NUnit is a .NET testing framework written by Jim Newkirk, Michael Two, Alexei Vorontsov, and Charlie Poole, based on the original NUnit by Philip Craig. NUnit is much the same as all the Extreme Programming test frameworks (xUnits), with two important differences:

NUnit uses the "attribute" feature of .NET to identify tests. Test fixtures are identified by the attribute [TestFixture] and individual tests by [Test]. This means that you can write tests that don't inherit from a fixed TestCase superclass, and NUnit can still find them.

NUnit allows you to write tests in any .NET language. So even though NUnit itself is written in C#, you can write your tests in Visual Basic or C++, or even in ML or Eiffel. Language inter operation is a key characteristic of .NET, and one that the team consider to be quite important.

When writing tests using NUnit, you can run the NUnit application, and it runs and report on your tests. There are two versions of the application, one that runs

at the command prompt, and a GUI version. When referring to NUnit, it is usually the GUI version that is implied.

You can find it at NUnit at:

<http://sourceforge.net/projects/nunit>

14.4.3 Unit Testing with csUnit

Lately there is another contender to NUnit V2.0 (at the time of writing V2.1 is still in beta form) known as csUnit (1.8.8) available from:

<http://www.csunit.org>

Both are available from SourceForge.net if desired.

Comparison with NUnit can be read here:

<http://www.csunit.org/index.php?page=http://www.csunit.org/documentation/index.html>

Essentially csUnit fits into the .Net/VS.NET much better. Their assemblies have now been defined inside the .Net System pane when you go to add reference. It is much easier to use and avoids hunting around for your installed location. It also adds menus into VS2003's menu system. The uninstall of csUnit is problematic but can be improved in later releases.

14.5 Version management

In a project such as the business case scenario discussed in this book, where multiple developers work on one application and series of tests are done. We recommend that version control be used to avoid conflicting code and ensure regular backup. The following are some version control tools:

14.5.1 ClearCase

ClearCase® is a Software Configuration Management System from Rational® Software, Inc. It keeps track of file versions used to build releases of a software product and to assist with the organization and address the increasingly complex problem of development and build management in a team environment.

More information about clearcase can be found at:

<http://www.rational.com/products/clearcase/index.jsp?SMSESSION=NO>

14.5.2 Concurrent Versions System

Client/server Concurrent Versions System (CVS) enables developers in different places to function as a single team. The version history is stored on a single central server and the client machines have a copy of all the files that the developers are working on. CVS maintains a history of all changes made to each directory tree it manages thus giving project managers fine control over the development process.

More information about CVS can be found at:

<http://ccvs.cvshome.org>

CVS can be used within WebSphere Studio Application Developer for source version control.

14.5.3 Visual SourceSafe

When beginning a .NET team development project, it is essential to understand how to establish development processes that work in a team environment. How to set up and work with the team development features supported by the Microsoft® Visual Studio® .NET integrated development environment (IDE), and to be aware of the development techniques (such as, how to set assembly references in the correct way) that must be followed by the development team members to ensure a successful working team.

The following Web site is a good starting point:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/tdlg_rm.asp



Scripts, source code and test data for YuBank

This appendix provides the technical solution to the YuBank business case scenario. It is divided into three sections.

The first section, WebSphere MQ Setup, contains the WebSphere MQ scripts required to setup the environment required to implement the business case scenario solution

The second section, use case 1, contains the source code solution to the .NET application to .NET application as well as the .NET application to J2EE application example.

The third section, use case 2, contains the source code solution to the .NET application to .NET Web Service and .NET application to J2EE Web Service.

All source code associated with YuBank can be downloaded from “Using the Web material” on page 323, **ALL.zip**.

Alternatively the individual programs of interest can be downloaded individually as indicated below:

WebSphere MQ Setup

There are 7 scripts required to setup WebSphere MQ as required for the business case scenario solution.

DOTDP.MQSC is a command file to setup WebSphere MQ objects for queue manager DOTDP. It creates a local queue called CreditCheck for CreditCheck C# application, a model queue called CreditScore for permanent dynamic queues for the replies and a deadletter queue called SYSTEM.DEAD.LETTER.QUEUE. It also creates a server connection channel called TO.DOTDP using an MCAUSER called kulkarni which would have to be changed before running the scripts.

DOTDP.MQSC can be downloaded from “Using the Web material” on page 323, **WMQ Setup.zip**.

DOTIP.MQSC is a command file to setup WebSphere MQ objects for queue manager DOTIP which:

- ▶ Created two local queues called SOAP.RESPONSE.YuBankIASWS and SOAP.RESPONSE.ShareQuote which act as the response queue for .NET Web Service (Investment Advices Service, IAS) and response queue for J2EE Web Service (StockQuote) respectively.
- ▶ Creates a model queue called CustomerProfile.
- ▶ Creates a remote queue called CustomerDetails
- ▶ Creates two transmission queues called DOTOP and DOTEPI
- ▶ Creates two sender channels. One called DOTIP.TO.DOTOP to queue manager DOTOP and another called DOTIP.TO.DOTEPI to queue manager DOTEPI. The CONNNAME parameters have to be changed before running the scripts
- ▶ Created two receiver channels. One called DOTOP.TO.DOTIP from queue manager DOTOP and another called DOTEPI.TO.DOTIP from queue manager DOTEPI
- ▶ Creates a server connection channel called TO.DOTIP using an MCAUSER called stevens which would have to be changed before running the scripts
- ▶ Registers the queue SYSTEM.DEAD.LETTER.QUEUE as the queue managers dead letter queue

DOTIP.MQSC can be downloaded from “Using the Web material” on page 323, **WMQ Setup.zip**.

DOTEPI.MQSC is a command file to setup WebSphere MQ objects for queue manager DOTEPI. It creates a transmission queue called DOTIP. It also creates a sender channel called DOTEPI.TO.DOTIP to queue manager DOTIP, a receiver channel called DOTIP.TO.DOTEPI from queue manager DOTIP and registers SYSTEM.DEAD.LETTER.QUEUE as the queue managers dead letter queue.

DOTOP.MQSC can be downloaded from “Using the Web material” on page 323, **WMQ Setup.zip**.

DOTOP.MQSC is a command file to setup WebSphere MQ objects for queue manager DOTOP. It creates a local queue called CustomerDetails for the CustomerProfile J2EE application. It also creates a transmission queue called DOTIP, sender channel called DOTOP.TO.DOTIP to queue manager DOTIP, a receiver channel called DOTIP.TO.DOTOP from queue manager DOTIP and registers SYSTEM.DEAD.LETTER.QUEUE as the queue managers dead letter queue.

DOTOP.MQSC can be downloaded from “Using the Web material” on page 323, **WMQ Setup.zip**.

WSCliant.MQSC is a command file to setup WebSphere MQ Objects for queue manager WSCliant. It creates two local queues one called SOAP.RESPONSE.RandomNumberNET which acts as response queue for .NET Web Service "RandomNumber" and the other called SOAP.RESPONSE.RandomNumberAXIS as a response queue for J2EE Web Service "RandomNumber" It also creates a transmission queue called WSServer, sender channel called WSCliant.TO.WSServer to queue manager WSServer, a receiver channel called WSServer.TO.WSCliant and registers the queue SYSTEM.DEAD.LETTER.QUEUE as the queue managers dead letter queue.

WSCliant.MQSC can be downloaded from “Using the Web material” on page 323, **WMQ Setup.zip**.

Note: This script only used for the WebSphere MQ Transport for SOAP examples. It is not needed to run the business case application.

WSServer.MQSC is a command file to setup WebSphere MQ Objects for queue manager WSServer. It creates two local queues one called SOAP.RESPONSE.RandomNumberNET which acts as response queue for .NET Web Service "RandomNumber" and the other called SOAP.RESPONSE.RandomNumberAXIS as a response queue for J2EE Web Service "RandomNumber" It also creates a transmission queue called WSCliant, sender channel called WSServer.TO.WSCliant to queue manager WSCliant, a receiver channel called WSCliant.TO.WSServer, a server connection channel called TO.WSServer and registers the queue SYSTEM.DEAD.LETTER.QUEUE as the queue managers dead letter queue.

WSServer.MQSC can be downloaded from “Using the Web material” on page 323, **WMQ Setup.zip**.

Note: This script is only used for the WebSphere MQ Transport for SOAP examples. It is not needed to run the business case application.

initWMQ.bat is a batch file which creates the queue manager, starts it and sets it to automatic re-start, runs the MQSC files for the each queue manager specified as a parameter and starts a listener on port 1414.

Example 14-3 Run initWMQ for queue manager DOTOP

```
initWMQ DOTOP
```

initWMQ.bat can be downloaded from “Using the Web material” on page 323, **WMQ Setup.zip**.

initWMQ_Simple.bat is another batch file which creates the queue manager, starts it and sets it to automatic re-start, runs the MQSC files for the each queue manager specified as a parameter and starts a listener, but this time on port 1415.

initWMQ_Simple.bat can be downloaded from “Using the Web material” on page 323, **WMQ Setup.zip**.

Note: This batch file is only used for the WebSphere MQ Transport for SOAP examples. It is not needed to run the business case application.

Use case 1

This is where the new investment account is opened. The Bank Service System (BSS) is an ASP.NET application that presents several forms on the screen to collect information from the customer. The source can be downloaded from “Using the Web material” on page 323, **BSS.zip**.

Once this information is entered, the ASP.NET application passes the customer’s detail using WebSphere MQ to the Credit Check System (CCS), which is a console mode .NET application. Two solutions exist for CCS as it is coded in both C# and VB.NET in order to highlight the WebSphere MQ coding techniques in the different languages.

The C# version can be downloaded from “Using the Web material” on page 323, **CCS.zip**.

The VB.Net version can be downloaded from “Using the Web material” on page 323, **CCS.zip**.

Once the CCS application receives information via WebSphere MQ it prepares a reply message highlighting the customers credit score. The BSS application determines if the customer is a suitable candidate for investment advice and, if so, forwards the customer details to the Credit Profile System (CPS) which is a J2EE application. The CPS application writes this information to a local database.

The CPS application can be downloaded from “Using the Web material” on page 323, **CPS.zip**.

Use case 2

This is where the BSS application requests investment advice for the customer. It invokes the Investment Advisory System (IAS), which is a .NET Web Service. It is the job of IAS to recommend an investment by sending a SOAP response to BSS using WebSphere MQ.

IAS can be downloaded from “Using the Web material” on page 323, **IAS.zip**.

To determine the investment advice the IAS, acting now as a .NET application, first requests information from the CPS application in order to acquire additional customer information before invoking the Share Quote System (SQS) which is a J2EE Web Service. SQS then looks up share information and provides the results over WebSphere MQ.

The SQS application can be downloaded from “Using the Web material” on page 323, **SQS.zip**.



B

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/SG247012>

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, SG247012.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
ALL.zip	This gives you everything

BSS.zip	This is only the Banking System Server code
CCS.zip	This is only the Credit Check System code
CPS.zip	This is only the Credit Profile Service code
Chapter 3 Demos.zip	This is the sample codes pertaining to chapter 3
IAS.zip	This is the Internet Advisory Service code
SQS.zip	This is the Share Quote System code
WMQ Setup.zip	This is the scripts to setup WebSphere MQ

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	ALL.ZIP is 2.31MBs and expands to 3.79MB (unbuilt)
Operating System:	Windows 2000 or Windows XP (Professional or Server)
Processor:	Pentium® III 500MHz or better
Memory:	256MB or higher is recommended

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.

Example 14-4 Content of All.ZIP

```

bss\BankingServiceSystem
bss\BankingServiceSystem\BankingServiceSystem.sln
bss\IIS\BankingServiceSystem
bss\IIS\BankingServiceSystem\bin
bss\IIS\BankingServiceSystem\img
bss\IIS\BankingServiceSystem\AcctOpenFail.aspx
bss\IIS\BankingServiceSystem\AcctOpenFail.aspx.cs
bss\IIS\BankingServiceSystem\AcctOpenFail.aspx.resx
bss\IIS\BankingServiceSystem\AcctOpenOK.aspx
bss\IIS\BankingServiceSystem\AcctOpenOK.aspx.cs
bss\IIS\BankingServiceSystem\AcctOpenOK.aspx.resx
bss\IIS\BankingServiceSystem\AssemblyInfo.cs
bss\IIS\BankingServiceSystem\BankingServiceSystem.csproj
bss\IIS\BankingServiceSystem\BankingServiceSystem.csproj.webinfo
bss\IIS\BankingServiceSystem\BSS Application Description.txt
bss\IIS\BankingServiceSystem\CreditCheckRequest.cs
bss\IIS\BankingServiceSystem\DisplayAdvice.aspx
bss\IIS\BankingServiceSystem\DisplayAdvice.aspx.cs
bss\IIS\BankingServiceSystem\DisplayAdvice.aspx.resx
bss\IIS\BankingServiceSystem\DisplayError.aspx
bss\IIS\BankingServiceSystem\DisplayError.aspx.cs
bss\IIS\BankingServiceSystem\DisplayError.aspx.resx
bss\IIS\BankingServiceSystem\Global.asax
bss\IIS\BankingServiceSystem\Global.asax.cs

```

bss\IIS\BankingServiceSystem\Global.asax.resx
 bss\IIS\BankingServiceSystem\Home.aspx
 bss\IIS\BankingServiceSystem\Home.aspx.cs
 bss\IIS\BankingServiceSystem\Home.aspx.resx
 bss\IIS\BankingServiceSystem\HomePage.gif
 bss\IIS\BankingServiceSystem\OtherPages.gif
 bss\IIS\BankingServiceSystem\Portfolio.asmx.cs
 bss\IIS\BankingServiceSystem\Portfolio.asmx.resx
 bss\IIS\BankingServiceSystem\Portfolio.cs
 bss\IIS\BankingServiceSystem\RequestAdvice.aspx
 bss\IIS\BankingServiceSystem\RequestAdvice.aspx.cs
 bss\IIS\BankingServiceSystem\RequestAdvice.aspx.resx
 bss\IIS\BankingServiceSystem\RequestIDetails.aspx
 bss\IIS\BankingServiceSystem\RequestIDetails.aspx.cs
 bss\IIS\BankingServiceSystem\RequestIDetails.aspx.resx
 bss\IIS\BankingServiceSystem\RequestPDDetails.aspx
 bss\IIS\BankingServiceSystem\RequestPDDetails.aspx.cs
 bss\IIS\BankingServiceSystem\RequestPDDetails.aspx.resx
 bss\IIS\BankingServiceSystem\Web.config
 bss\IIS\BankingServiceSystem\bin\amqmdnet.dll
 bss\IIS\BankingServiceSystem\bin\BankingServiceSystem.dll
 bss\IIS\BankingServiceSystem\bin\BankingServiceSystem.pdb
 bss\IIS\BankingServiceSystem\img\HomePage.gif
 bss\IIS\BankingServiceSystem\img\OtherPages.gif
 ccs\CreditDatabase.xml
 ccs\C# Version\CCS
 ccs\C# Version\ccs.sln
 ccs\C# Version\CCS\App.ico
 ccs\C# Version\CCS\AssemblyInfo.cs
 ccs\C# Version\CCS\CCS.csproj
 ccs\C# Version\CCS\CCS.csproj.user
 ccs\C# Version\CCS\Database.cs
 ccs\VB.NET Version\CCSVB
 ccs\VB.NET Version\CCSVB.sln
 ccs\VB.NET Version\CCSVB\AssemblyInfo.vb
 ccs\VB.NET Version\CCSVB\CCSVB.vbproj
 ccs\VB.NET Version\CCSVB\Database.vb
 chapter 3 demos\RandomServiceAxis
 chapter 3 demos\RandomServiceNET
 chapter 3 demos\RandomServiceAxis\RandomNumberAXIS.java
 chapter 3 demos\RandomServiceAxis\RandomNumberAxisClient.cs
 chapter 3 demos\RandomServiceNET\RandomNumberNET.asmx
 chapter 3 demos\RandomServiceNET\RandomNumberNET.cs
 chapter 3 demos\RandomServiceNET\RandomNumberNetClient.cs
 cps\cps.bat
 cps\CPSAdminObjects.txt
 cps\cps\CustomerProfile.class
 cps\cps\CustomerProfile.java
 cps\cps\MessageDetail.class

```

cps\cps\MessageDetail.java
cps\cps\MsgHandler.class
cps\cps\MsgHandler.java
cps\cps\OpenAccount.class
cps\cps\OpenAccount.java
cps\cps\QueryCustomerProfile.class
cps\cps\QueryCustomerProfile.java
cps\JAR\jaxen-core.jar
cps\JAR\jaxen-jdom.jar
cps\JAR\jdom.jar
cps\JAR\saxpath.jar
cps\JAR\xerces.jar
cps\JAR\xml-apis.jar
cps\XML\CustomerProfile.xml
ias\YuBankIASWS
ias\IASWebServiceSetup.msi
ias\readme.txt
ias\Setup.Exe
ias\Setup.Ini
ias\YuBankIASWS\bin
ias\YuBankIASWS\AssemblyInfo.cs
ias\YuBankIASWS\Global.asax
ias\YuBankIASWS\Global.asax.cs
ias\YuBankIASWS\Global.asax.resx
ias\YuBankIASWS\iasclient.cs
ias\YuBankIASWS\iasclient.exe
ias\YuBankIASWS\MQAccessCPS.cs
ias\YuBankIASWS\MQSOAP.dll
ias\YuBankIASWS\MessageQueryResult.xml
ias\YuBankIASWS\Portfolio.asmx
ias\YuBankIASWS\Portfolio.asmx.cs
ias\YuBankIASWS\Portfolio.asmx.resx
ias\YuBankIASWS\requestSQS.xml
ias\YuBankIASWS\ShareQuoteService.cs
ias\YuBankIASWS\sqs.cs
ias\YuBankIASWS\sqsoutput.xml
ias\YuBankIASWS\sqsoutput2.xml
ias\YuBankIASWS\Web.config
ias\YuBankIASWS\YuBankIASWS.csproj
ias\YuBankIASWS\YuBankIASWS.csproj.webinfo
ias\YuBankIASWS\YuBankIASWS.sln
ias\YuBankIASWS\bin\amqmdnet.dll
ias\YuBankIASWS\bin\MQSOAP.dll
ias\YuBankIASWS\bin\Portfolio.dll
ias\YuBankIASWS\bin\Portfolio.pdb
ias\YuBankIASWS\bin\YuBankIASWS.dll
ias\YuBankIASWS\bin\YuBankIASWS.pdb
sqs\ShareDataSource.xml
sqs\ShareQuote.java

```

```
sqs\lib\jaxen-core.jar  
sqs\lib\jaxen-jdom.jar  
sqs\lib\jdom.jar  
sqs\lib\saxpath.jar  
wmq setup\DOTDP.mqsc  
wmq setup\DOTEP.mqsc  
wmq setup\DOTIP.mqsc  
wmq setup\DOTOP.mqsc  
wmq setup\initWMQ.bat  
wmq setup\initWMQ_Simple.bat  
wmq setup\WSCClient.mqsc  
wmq setup\WSServer.mqsc
```

Glossary

Atomicity. Atomicity property of a transaction guaranties that the transaction executes exactly once and all the work required by its participants is done or none of it is.

Consistency. The semantics, integrity and consistent state of data must be persevered during transformation from pre-transaction to post-transaction phase. (For example travel estimate transaction should not arbitrarily move decimal point of prices quoted by other services).

Distributed transaction. A distributed transaction updates resources of several resource managers. The resources managers run on different machines.

Durability. A durable transaction assures that on successful completion, all the updates to the resources made by its operations persists, even if the computer crashes immediately after the commit.

Global unit of work. A global unit of work updates resources of the WebSphere MQ queue manager and of other resource managers, like databases. These global units of work can be coordinated by the queue manager or by an external transaction manager.

Isolation. In case of concurrent transaction processing, Isolation guaranties that the outcome obtained from a set of concurrent transactions are identical to the outcome obtained by running each transaction serially.

Local transaction. A local transaction updates resources of one or more resource manager, like WebSphere MQ or DB2 UDB. All resource managers run on the same machine.

Local unit of work. A local unit of work only updates resources of the WebSphere MQ queue manager.

Abbreviations and acronyms

2PC	Two-phase commit	IBM	International Business Machines Corporation
ACID	atomicity, consistency, isolation, and durability	ISO	International Standards Organization
ADO	ActiveX Data Objects	ITSO	International Technical Support Organization
API	Application Programming Interface	IVT	Independent Verification Test
AT	Atomic transactions	J2EE	Java 2 Platform, Enterprise Edition
BA	Business activity	JMS	Java Message Service
BSS	Bank Service System	JNDI	Java Naming and Directory Interface
CA	Certification Authority	JTA	Java Transaction API
CCS	Credit Check System	JVM	Java Virtual Machine
CN	Common Name	JWS	Java Web Service
CPS	Customer Profile System	MAC	Message Authentication Code
CRL	Certificate Revocation List	MCA	Message Channel Agent
CSD	Customer Service Diskette	MCS	Microsoft Certificate Store
CVS	Concurrent Versions System	MMC	Microsoft Management Console
DB	Database	MQI	Message Queue Interface
DIME	Direct Internet Message Exchange	MSDN	Microsoft Developer Network
DLL	Dynamic Link Library	MSMQ	Microsoft Message Queuing (MSMQ)
DN	Distinguished Names	MTS	Microsoft Transaction Server
DOB	Date of birth	OASIS	Organization for the Advancement of Structured Information Standards
DOM	Document Object Model	ODBC	Open Database Connectivity
DST	Digital Signature Trust	OLE	Object Linking and Embedding
DTC	Distributed Transaction Coordinator	PKI	Public Key Infrastructure
ECMA	Standardizing Information and Communication Systems	QM	Quality Management
FTP	File Transfer Protocol	QoS	Quality of Service
GAC	Global Assembly Cache		
HTTP	Hypertext Transfer Protocol		
IAS	Investment Advisory System		

RA	Regional Authorities
RM	Resource Manager
RPC	Remote Procedure Call
SAX	Simple API for XML
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQS	Share Quote System
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TID	Transaction ID
TM	Transaction Manager
TP	Transaction Processing
TPM	Transaction Process Managers
VB	Visual Basic
VB .NET	Visual Basic .NET
WS	Web Service
WS-BPEL	Business Process Execution Language
WSDL	Web Services Description Language
WS-I	Web Services Interoperability
WS-S	Web Service Security
WWW	World Wide Web
XML	Extensible Markup Language

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 about ordering these publications, see “How to get IBM Redbooks” on page 335. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *MQSeries Programming Patterns*, SG24-6506
- ▶ *WebSphere MQ Security in an Enterprise Environment*, SG24-6814

Other publications

These publications are also relevant as further information sources:

- ▶ *WebSphere MQ for Windows Quick Beginnings*, GC34-6073
- ▶ *WebSphere MQ Script (MQSC) Command Reference*, SC34-6055
- ▶ *WebSphere MQ Using Java*, SC34-6066
- ▶ *WebSphere MQ Extended Transactional Clients*, SC34-6275
- ▶ *WebSphere MQ System Administration Guide*, SC34-6068
- ▶ *WebSphere MQ Application Programming Guide*, SC34-6064
- ▶ *WebSphere MQ Application Programming Reference*, SC34-6062
- ▶ *WebSphere MQ Security*, SC34-6079
- ▶ *Principles of Distributed Database Systems (2nd Edition)*, by M. Tamer Ozsu, Patrick Valduriez. Prentice Hall, January 1999, ISBN 0136597076.
- ▶ *Transaction Processing: Concepts and Techniques*, by Jim Gray and Andreas Reuter. Morgan Kaufmann, 1st edition 1993, ISBN 1558601902
- ▶ *COM and .NET Component Services*, by Juval Löwy. O'Reilly and Associates, September 2001, ISBN 0596001037.
- ▶ *Discover SOAP Encoding's Impact on Web Service Performance*, by Frank Cohen; see article at:

<http://www-106.ibm.com/developerworks/webservices/library/ws-soapenc/>

These three tutorials are also relevant as further information sources. Note that a free registration is currently necessary to receive these tutorials:

- ▶ “Creating a Web Service from a Java class”
https://www6.software.ibm.com/reg/devworks/dw-ws-cwsjc-i?S_TACT=103AMW18&S_CMP=DEVXWS
- ▶ “Introduction to Web Services and the WSDK”
https://www6.software.ibm.com/reg/devworks/dw-ws-intwsdk-i?S_TACT=103AMW18&S_CMP=DEVXWS
- ▶ “Web services - The Web’s next revolution”, by Doug Tidwell
<http://www-106.ibm.com/developerworks/edu/ws-dw-wsbasics-i.html>

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ C# standards
<http://www.ecma-international.org/>
- ▶ Location for SupportPac MA0R
<http://www-3.ibm.com/software/integration/support/supportpacs/individual/ma0r.html>
- ▶ Microsoft Developer Network (MSDN) library
<http://msdn.microsoft.com/library/default.asp>
- ▶ developerWorks: IBM resource for developers
<http://www.ibm.com/developerWorks>
- ▶ Discover SOAP encoding's impact on Web Service performance
<http://www-106.ibm.com/developerworks/webservices/library/ws-soapenc/>
- ▶ Specification: Web Services Transaction
<http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/>
- ▶ Transactional client
<http://www-3.ibm.com/software/integration/wmq/transclient.html>
- ▶ Specification: Web Services Security
<http://www-106.ibm.com/developerworks/webservices/library/ws-secure/>
- ▶ SSL 3.0 specification
<http://wp.netscape.com/eng/ss13/>
- ▶ Internet X.509 Public Key Infrastructure

<http://www.ietf.org/internet-drafts/draft-ietf-pkix-logotypes-10.txt>

- ▶ WebSphere MQ supported software

http://www-3.ibm.com/software/integration/mqfamily/platforms/supported/wsmq_for_winnt2000_5_3.html

- ▶ OASIS

<http://www.oasis-open.org/committees/wsbpe1>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols

.NET application to .NET application 8
.NET application to a J2EE Web Service 10
.NET Application to J2EE Application 8
.NET Web Services and transactions 301

A

Access the DatabaseDOM in C# 125
Access the DatabaseDOM in VB .NET 130
Account opening 140
ACID properties 284, 291
Adding a reference to WebSphere MQ Transport for SOAP 189
Adding external classes to the CLASSPATH 219
Adding IAS Web Service proxy 191
Adding the WebSphere MQ reference to the project 119
Additional WebSphere MQ configuration 56
Alternative solutions 161
Apache Axis 7
Application layer security services 252
Application Programming Interface (API) 4
Asymmetric and symmetric cryptography techniques 263
Asymmetric key algorithm 255
Atomic Transactions (AT) 304
Atomicity 285
Attribute types commonly found in DN 259
Authentication 251
Authentication code 256
Authentication service 252
Authority 250–251
Authority database 251
Authority service 252
Authors xiv
AutoComplete attribute 297
Automatic Transaction 286

B

Bank service application 143, 147
 Bank service application (C#) 121
Bridge between WebSphere MQ and Microsoft Mes-

sage Queuing (MSMQ) 162
BSS user interface testing 203
BSS Web Application deployment 197
BSS Web Application Solution 192
Builds of software components and packages 229
Business Activity (BA) 304
Business logic implementation 213
Business Process Execution Language (WS-BPEL) 304

C

C, C++ or Java API. 3
CA certificate 258
CA generates a digital certificate that contains 258
Calling deployWMQService 31
Calling the service from the IAS client 222
Certificate chain 261
Certificate Revocation List (CRL) 259, 262
Certificates
 When they are no longer valid? 261
Certification Authority (CA) 258–259
Change request queue to use new initiation queue 63
Ciphertext 254
Client environment 24
Code example 287
Coding Standards 306
Common errors 305, 307
Common Name (CN) 259
Component Services 291
Component Services Administration 292
Concepts of SSL 262
Concurrent Versions System (CVS) 315
Confidentiality 251
Confidentiality service 253
Configuring BSS to use SOAP 135
Configuring BSS to use SSL 135
Configuring IIS for BSS Web Application 134
Consistency 285
Core systems overview 110
CPS and SQS communication logic testing 201
Create the DOM in C# 125
Create the DOM in VB .NET 129

- Create the new initiation queue 62–63
- Credit check application 124
- Credit check application C# snippet 125
- Credit check application VB .NET snippet 129
- Credit check database 124
- Cryptographic concepts 254
- Cryptographic techniques 255
- Cryptography 251, 254
- csUnit 313
- Current status and future plans 69
- Customer profile application 144, 150
- Customer Service Diskette (CSD)
 - latest 4
- Customizing authentication using SOAPHeaders 198

D

- Data integrity 251
- Data integrity service 253
- Database 145
- Decipherment 255
- Decryption 255
- Define the WebSphere MQ response queue 43
- Define WebSphere MQ channel definitions 46
- Demo (test) certificates 260
- Demo facility 260
- Deploy the service 54
- Deploying BSS 134, 157
- Deploying CCS 136
- Deploying CPS 157
- Deploying the Microsoft .NET service 39
- Deployment 133, 157, 194, 219
- Deployment of J2EE Web Services 48
- Development 119, 147, 213
- Digital certificate 258
 - What is it? 258
- Digital signature 251, 257
- Digital signature process 257
- Direct Internet Message Exchange (DIME) 21
- Distinguished Name (DN) 259
- Distributed test in WebSphere MQ client mode 45, 59
- Distributed test in WebSphere MQ server bindings mode 57
- Distributed transaction 283, 290–291, 294
- Distributed Transaction Coordinator (DTC) 291
- Document Object Model (DOM) 125, 213, 307
- Document Style encoding (also known as message-

- ing style) 21
- Downloading WebSphere MQ transport for SOAP 13
- DTC terms 291
- Durability 285

E

- Eavesdropping 250
- Effective programs 305
- Encipherment 254
- Encryption 254
- Encryption and decryption algorithms 256
- Environment Setup 110
- Error handling in the Web Service 202
- Examples
 - of common security mechanisms 251
 - WebSphere MQ transport for SOAP
 - used with a Microsoft .NET Web Service 36
 - used with J2EE Web Service 52
- Executing MQSoapHost 35
- Executing SimpleJMSListener 52
- eXtensible Markup Language (XML) 6

F

- File Transfer Protocol (FTP) 2
- Functional testing 230

G

- gacutil 220
- gacutil utility 15
- General data flow 231, 241
- Generating WebSphere MQ reply 128, 132
- Global Assembly Cache (GAC) 19
- Global unit of work 291

H

- Hash function 256–257
- Hash method 280
- HashTable 281
- Hints and tips 305–306
- How to
 - Access the DatabaseDOM in C# 125
 - Access the DatabaseDOM in VB .NET 130
 - Create the DOM in C# 125
 - Create the DOM in VB .NET 129
 - Read a WebSphere MQ message in C# 126
 - Read a WebSphere MQ message in VB.NET

- 130
- Send a WebSphere MQ reply 129
- Specify RPC or Document style encoding 22
- Start BSS 136
- Start CSS 137
- Hypertext Transfer Protocol (HTTP) 2, 4, 7

I

- IAS Web Service deployment 194
- IAS Web Service testing
 - using Microsoft Visual Studio .NET 199
- IBM (WebSphere Application Server) 304
- Identification 251
- Identification and authentication 250
- Identification and authentication services 252–253
- Identification service 252
- identification service 252
- iKeyman tool 260
- Impersonating 250
- Importing WebSphere MQ Transport for SOAP 190
- Installation 17
- Installing Internet Information Services (IIS) 90
- Installing Microsoft Visual Studio .NET 93
- Installing WebSphere MQ 90
- Installing WebSphere MQ classes for Microsoft .NET 90
- Installing WebSphere MQ Transport for SOAP 90
- Integration issues
 - solutions 305
- Integration runtime environment 229
- Interface definitions 118, 208
- Internet Information Services (IIS) 4
- Internet X.509 Public Key Infrastructure 254
- Interoperability of the Web Services 2
- Interoperability within heterogeneous environments 2
- Inter-system communications 229
- Introduction 283
- Investment advisory 141
- Investment advisory application 144, 148
- Isolation 285
- IVT 49
- IVT (Independent Verification Test) 17

J

- J2EE technology 6
- Java 2 Platform, Enterprise Edition (J2EE) 1
- Java Message Service (JMS) 146

- Java Message Service (JMS) API 139
- Java Naming and Directory Interface (JNDI) 146
- Java Transaction API (JTA) 285, 331
- Java Virtual Machine (JVM) 219
- JMS administered objects 146
- JMS listener 218
- JMSListener 33
- JUnit 313

L

- Linked transactions using manual transaction management 296
- Local transaction 283, 291, 294
- Local unit of work 291
- Long term maintenance 305

M

- maOr_netdir environment variable 15
- Management Console (MMC) 299
- Manual Transaction 286
- Message Authentication Code (MAC) 257
- Message digest 256–257
- Message flow 80
- Messaging service 1
- Microsoft (BizTalk Server Technologies) 304
- Microsoft .NET 1
- Microsoft .NET Framework 7
- Microsoft .NET Framework and SDK 15
- Microsoft DTC transaction manager 292
- Microsoft Internet Information Services (IIS) 14
- Microsoft Management Console (MMC) 292
- Microsoft Message Queuing (MSMQ) 1, 4
- Microsoft Transaction Server
 - MTS and WebSphere MQ 300
- Microsoft Transaction Server (MTS) 300

N

- NET application to a .NET Web Service 9
- New coding standards for the .NET technology 305
- Non-repudiation 251, 253
- NUnit 313

O

- Obtaining personal certificates 260
- Organization for the Advancement of Structured Information Standards (OASIS) 304
- Overview of security services 250

Overview of WebSphere MQ transport for SOAP 11

P

Persistent storage 215
Personal certificate 258
Plaintext 254
Planning the security services in use cases 252
Pre-Installation 15
Prerequisite software 14
Prerequisite software installation order 15
Process overview 116, 140, 208
Programming distributed transactions 294
Programming local transactions 285
Programming Web Services transaction in .NET environment 301
Public Key Infrastructure (PKI) 262

Q

Quality Management (QM) 228
Quality of Service (QoS) 228

R

Read a WebSphere MQ message in C# 126
Read a WebSphere MQ message in VB.NET 130
Redbooks Web site 335
 Contact us xvii
Registering WebSphere MQ Transport for SOAP 190
Registration Authorities (RAs) 262
Remote Procedure Call (RPC) encoding 21
Remote Procedure Call (RPC) Literal encoding 21
Resource Manager (RM) 291
Roles of a CA 259
Root CA certificate 261
Run the trigger monitor with the new initiation queue 63
Runtime environment 219
Runtime errors 307

S

Scenario overview 117
Secure Sockets Layer (SSL) 4, 253
Secure Sockets Layer (SSL) introduction 262
Secure Sockets Layer (SSL) provides 262
Secure the transportation of messages between applications and Web Services 250
Securing the IAS Web Service 197

Security concepts 250
Security mechanisms 251
Security problems 250
Security services 250
 application layer 252
 overview 250
 transmission layer 252
Security services that are identified in security architecture 250
Security that HTTP provides 2
Send a WebSphere MQ reply 129
Server configuration 82
Service application binary (Java classes and libraries) 219
Service code use of external classes 59
Service operation definition 209
Setting environment variables 15
Setting up the JMS administered objects 158
ShareQuote service deployment 220
Signer certificate 258
Simple API for XML (SAX) 213, 307
Simple demonstration with SSL 65
Simple Mail Transfer Protocol (SMTP) 2
Simple Object Access Protocol (SOAP) 2, 4, 6, 20, 163
SOAP
 What is it? 6
SOAP formatting 20
SOAP processing in Java 307
SOAP style and encoding variants 21
SOAP with attachments 21
Software prerequisites 90
Solution discussion 225
Solutions for integration issues 305
Specify RPC or Document style encoding 22
SSL authentication 263
SSL connection 263
SSL handshake 263
Standard Security 197
Start BSS 136
Start CSS 137
Start the prepared JMS listener 56
Start the prepared Microsoft .NET listener 43
Starting listeners with WebSphere MQ triggering 60
Submit 244
Symmetric cryptography techniques 263
Symmetric key algorithm 255
System Context 208
System context 118, 142

System integration 228
System integration and functional test
 summary 248

T

Tampering 250
Techniques and implementation for securing the
transportation of messages 249
Test 1 Pass known data 137
Test 2 Pass unknown user 138
Test Case 1
 successful account opening
 full qualification 231
 successful investment advisory
 portfolio recommendations 242
Test Case 2
 successful investment advisory
 no investment 245
 unsuccessful account opening
 unqualified credit score 235
Test Case 3
 negative tests
 exception recovery and system unavailability
 236
 unsuccessful investment advisory
 invalid customer data 246
Test Case 4
 negative tests
 exception recovery and system unavailability
 247
Test data 229
Test result 224
Test the service 44, 57
Testing 136, 159, 222, 313
Transaction ID (TID) 294
Transaction Manager (TM) 292
Transaction Models in .NET Framework 286
Transaction Process Managers (TPM) 294
Transaction support under Windows 2000 291
Transactions 283–284
Transactions in a COM+ 285
Transmission layer security services 252–253
Two phase commit (2PC) protocol 293
Two-phase commit
 What is it? 293

U

Unauthorized access 250

Unit Testing with csUnit 314
Unit Testing with JUnit 313
Unit Testing with NUnit 313
Use case 1
 account opening 75, 231
 account opening message flow 80
Use case 2
 investment advisory 76, 241
 investment advisory message flow 81
Use of SSLPeerName 66
User certificate 258
User database 251
Using a different initiation queue 62
Using Component Services Manager 299

V

VB.NET 5
Visual Basic (VB) 5
Visual SourceSafe 315

W

Web Service transactions 283, 300
Web Services 162
Web Services Description Language (WSDL) 25
Web Services Security 334
Web Services Security (WS-Security) 253
WebSphere Application Server 2
WebSphere MQ xiii, 1
WebSphere MQ as a transport mechanism 4
WebSphere MQ classes for Microsoft .NET xiii, xvi
WebSphere MQ classes for Microsoft .NET and
WebSphere MQ classes for Java 161
WebSphere MQ client connection options 30
WebSphere MQ configuration 83
WebSphere MQ definition 218
WebSphere MQ queue setup and WebSphere MQ
transport for SOAP deployment 196
WebSphere MQ setup 112
WebSphere MQ SSL support 263
WebSphere MQ terms 291
WebSphere MQ Transport for SOAP
 SimpleJMSListener 49
WebSphere MQ transport for SOAP xiii, xvi
 .NET deployment 25
 and SSL 64
 application development 23
 checking the release level 20
 deployment for IAS 185

- installation 13
- listener for .NET 33
- overview 11
- What is it? 12
- with J2EE deployment 48
- WebSphere MQ transport for SOAP and SSL 64
- WebSphere MQ URI Syntax 29
- WebSphere MQ, IBM messaging service 1
- What is a digital certificate? 258
- What is SOAP? 6
- What is two-phase commit? 293
- What is WebSphere MQ transport for SOAP? 12
- When certificates are no longer valid? 261
- World Wide Web (WWW) 9
- Write the .NET ASMX service directive file 38
- Write the client application 41, 55
- Write the Web Service 36, 53
- WS Coordination 304
- WS Transaction 303–304

X

- XML data format 209
- XML element versus attribute 307
- XML processing in Java 307
- XML schema 307
- XML style comments 306

Y

- YuBank architecture 82



WebSphere MQ Solutions in a Microsoft .NET Environment

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Redbooks

WebSphere MQ Solutions in a Microsoft .NET Environment

Invoking WebSphere MQ from a .NET application

WebSphere MQ as a SOAP transport mechanism

.NET and J2EE integration using WebSphere MQ

The importance of the .NET platform is growing. This IBM Redbook illustrates how to integrate WebSphere MQ technology in a .NET environment, specifically with Microsoft Windows, by providing samples and guidance about how this can be achieved. It demonstrates the use of WebSphere MQ technology in a Microsoft Windows platform and proves WebSphere MQ is well integrated with the .NET environment.

This redbook demonstrates the use of WebSphere MQ in a .NET Web Service in these ways:

WebSphere MQ is used as a transport mechanism for the invocation of the Web Service by modifying the SOAP protocol wrapper to utilize WebSphere MQ rather than HTTP.

WebSphere MQ is used as a middleware product in the implementation of a Web Service. For example, the Web Service sends an MQ message as a request to another application and when this application responds, the Web Service provides a response back to the .NET Web Service client. Some sample applications in, C#, J2EE and VB.NET also demonstrate this.

Finally, a discussion about transactions is included, which highlights how WebSphere MQ participates in a transaction managed by DTC, and also how WebSphere MQ implements and uses Windows security interfaces, such as Secure Sockets Layer (SSL), is covered.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks