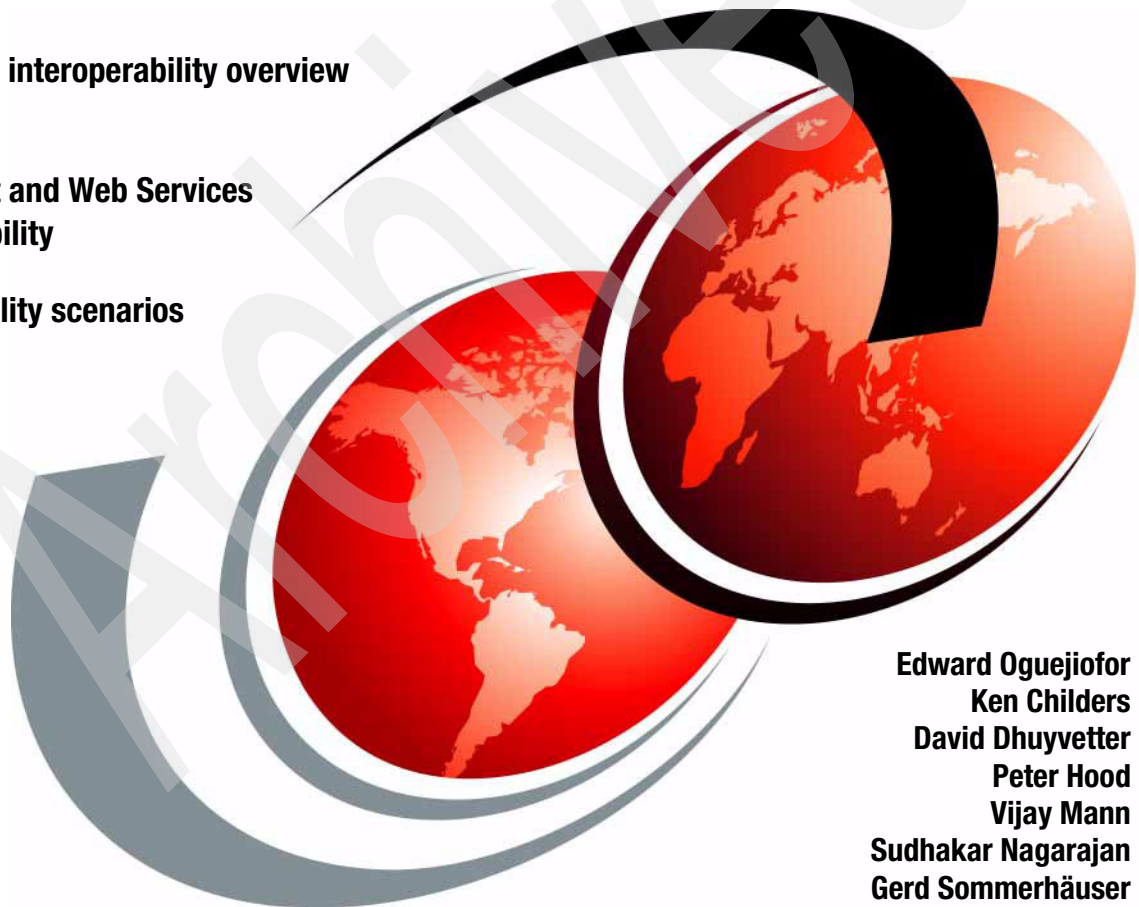


IBM WebSphere and Microsoft .NET Interoperability

Application interoperability overview

Component and Web Services
interoperability

Interoperability scenarios



Edward Oguejiofor
Ken Childers
David Dhuyvetter
Peter Hood
Vijay Mann
Sudhakar Nagarajan
Gerd Sommerhäuser



International Technical Support Organization

IBM WebSphere and Microsoft .NET Interoperability

July 2006

Archived

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

First Edition (July 2006)

This edition applies to IBM WebSphere Application Server V6.0 (product number D51HNLL), IBM Rational Application Developer for WebSphere Software V6.0 (product number D54SDLL), Java 2 Platform, Enterprise Edition (J2EE) 1.4, Microsoft Visual Studio .NET 2003, and the Microsoft .NET Framework Version 1.1.

© Copyright International Business Machines Corporation 2006. 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	ix
Trademarks	x
Preface	xi
What this redbook is about	xi
What this redbook is not about	xii
The target audience	xii
Structure of this book	xiii
The team that wrote this redbook	xiv
Become a published author	xvi
Comments welcome	xvii
Part 1. Introduction	1
Chapter 1. Application interoperability overview	3
1.1 Drivers for application interoperability	4
1.1.1 Technology transition	4
1.1.2 Best of breed integration	5
1.1.3 The extended enterprise	5
1.2 Application interoperability models	6
1.2.1 Application architecture model	6
1.2.2 Interoperability scenarios	9
1.3 Approaches for interoperability	25
1.3.1 Service-oriented approach	25
1.3.2 Class level approach	25
1.4 Elements of interoperability	27
1.4.1 Application interoperability stack	27
1.4.2 Activities	29
1.4.3 Constraints	29
Chapter 2. Introduction to J2EE and WebSphere platform	31
2.1 Introduction to J2EE	32
2.1.1 Roles in J2EE environment	33
2.1.2 J2EE n-tier architecture	35
2.1.3 J2EE component model	38
2.1.4 J2EE application environment	42
2.1.5 Security in J2EE	48
2.1.6 Technologies supporting J2EE architecture	52
2.2 The WebSphere platform	54

2.2.1	The WebSphere family	56
2.2.2	The WebSphere Application Server family	61
2.2.3	Stand-alone server configuration	62
2.2.4	Distributed server configuration	63
2.3	IBM Rational Software Development Platform	66
2.3.1	IBM Rational Software Development Platform technologies	68
2.3.2	The IBM Rational Software Development Platform products	72
Chapter 3.	Introduction to .NET platform	91
3.1	The .NET initiative	92
3.1.1	Windows DNA	92
3.1.2	Evolution of .NET	96
3.2	The .NET suite	100
3.2.1	The .NET Framework	101
3.2.2	ASP.NET	115
3.2.3	.NET enterprise servers	119
3.3	Developing .NET applications	121
3.3.1	Writing a C# application using text editor	121
3.3.2	Developing applications using Microsoft Visual Studio .NET	124
3.3.3	Testing	127
3.4	Deploying and managing .NET applications	133
3.4.1	Deployment	134
3.4.2	Runtime	136
3.4.3	Administration	136
3.4.4	Windows Services	138
3.4.5	Object pooling	139
3.4.6	Remote invocation	139
3.4.7	Web Services	140
3.4.8	Transaction management	141
3.4.9	Security	142
3.4.10	Load balancing and failover	144
3.4.11	Application logging	144
Part 2.	Component interoperability	147
Chapter 4.	Introduction to component interoperability	149
4.1	Components overview	150
4.1.1	Client side components	150
4.1.2	Server side components	153
4.2	Introduction to component interoperability	159
4.3	Why choose component interoperability?	162
4.3.1	Interface	162
4.3.2	Transaction	162
4.3.3	Interaction	162

4.3.4 Events	163
Chapter 5. Designing component interoperability	165
5.1 Application considerations	166
5.1.1 Interaction and state management	166
5.1.2 Message format	175
5.1.3 Life cycle management	176
5.2 Data considerations	176
5.2.1 Type mapping	177
5.2.2 Pass by value or reference	178
5.2.3 Callbacks	179
5.3 Control considerations	179
5.3.1 Factors	180
5.3.2 Approaches	181
5.3.3 Products	182
5.4 Transport considerations	187
5.4.1 Run on the same machine	187
5.4.2 Different machines	190
Chapter 6. Component interoperability scenario	191
6.1 Scenario description	192
6.1.1 Basic interaction outline	192
6.1.2 Message format	193
6.1.3 The calculator service	194
6.1.4 Messaging using WebSphere MQ	196
6.1.5 System prerequisites	201
6.2 Solution overview	201
6.3 Queues installation and configuration	203
6.3.1 Install and configure the WebSphere MQ Server on the server ...	203
6.3.2 Install and configure WebSphere MQ Client on the client.	210
6.4 Create messaging resources, schemas, and classes	211
6.4.1 Create JMS resources in WebSphere Application Server	211
6.4.2 Create the XML schema for messages	219
6.4.3 Generate the .NET classes corresponding to the XML schema ...	221
6.4.4 Generate the Java classes corresponding to the XML schema ...	221
6.5 Developing the .NET Windows Form client application	222
6.5.1 Developing the interoperability adapter	222
6.5.2 Developing the Windows Form .NET client	230
6.6 Developing the WebSphere calculator component	234
6.7 Testing the sample application	244
6.7.1 Troubleshooting	246
Part 3. Web Services interoperability	247

Chapter 7. Introduction to Web Services Interoperability	249
7.1 Introduction to Web Services	250
7.1.1 Web Services background	250
7.1.2 Web Services model	251
7.1.3 Web Services specifications	253
7.1.4 Web Services architecture model	261
7.2 Overview of Web Services Interoperability	262
7.2.1 Profiles	263
7.2.2 Sample applications	266
7.2.3 Testing tools	267
Chapter 8. Designing Web Services interoperability	269
8.1 Elements of Web Services interoperability	270
8.1.1 Web Services activities	270
8.1.2 Web Services constraints	272
8.2 Web Services description	273
8.3 Web Services invocation	278
8.4 Web Services constraints	281
8.4.1 WS-I Basic Profile V1.1	281
8.4.2 WS-I Attachments Profile V1.0	282
8.4.3 WS-I Support	283
8.4.4 Web Services description constraints	285
8.5 WS-Security support	291
Chapter 9. Web Services interoperability scenario	293
9.1 Introduction to the claims processing scenario	294
9.2 Building the WebSphere Claims Web Service	295
9.2.1 Configure the Development Environment	295
9.2.2 Create Web Service from Session EJB	297
9.2.3 Testing with the Web Services Explorer	306
9.2.4 Deploying the Web Service	309
9.3 Building the .NET Claims Web Service	313
9.3.1 Create Web service project	313
9.3.2 Import existing classes	315
9.3.3 Update the Web Service code	317
9.3.4 Building and deploying the Web Service on IIS6	318
9.3.5 Test the Microsoft .NET Web Service	320
9.4 The client application	324
9.4.1 Import the existing client	325
9.4.2 Update the client to use the .NET Web Service	326
9.4.3 Test the updated client	336
9.4.4 Update the .NET Service to call WebSphere findCustomer	338
9.4.5 Test the complete solution	341

9.5 Web Services security	342
9.5.1 Encrypting messages to .NET Web Service	342
9.5.2 Signing requests from .NET to WebSphere Web Services	354
9.6 Difference between the two Web Services	368
9.6.1 Exception handling	368
9.6.2 Object array management.	371
9.6.3 Parameter multiplicity specification	373
Part 4. Appendices	375
Appendix A. Additional material	377
Locating the Web material	377
Using the Web material	377
System requirements for downloading the Web material	378
How to use the Web material	378
Abbreviations and acronyms	381
Related publications	385
IBM Redbooks	385
IBM Redpapers	385
Other publications	386
Online resources	386
How to get IBM Redbooks	390
Help from IBM	390
Index	391

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 about 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:

ClearCase MultiSite®
ClearCase®
ClearQuest®
developerWorks®
DB2®
Everyplace®
IBM®
Lotus®
MQSeries®

ProjectConsole™
PurifyPlus™
Rational Developer Network®
Rational Rose®
Rational Suite®
Rational Unified Process®
Rational®
Redbooks™
Redbooks (logo) ™

RequisitePro®
RUP®
ScriptAssure™
SoDA®
Team Unifying Platform™
Tivoli®
WebSphere®
Workplace™
XDE™

The following terms are trademarks of other companies:

Enterprise JavaBeans, EJB, Java, Java Naming and Directory Interface, JavaBeans, JavaMail, JavaScript, JavaServer, JavaServer Pages, JDBC, JMX, JRE, JSP, JVM, J2EE, J2SE, Sun, Sun Microsystems, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Active Directory, ActiveX, BizTalk, IntelliMirror, IntelliSense, Internet Explorer, JScript, Microsoft, MSDN, Visual Basic, Visual C++, Visual C#, Visual Studio, Windows NT, Windows Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

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

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

Preface

In today's enterprise environment, the existence of heterogeneous platforms is a reality. The need for interoperability between the different technology platforms is not just a question of desirability but rather one of necessity. Interoperability between the two of the most widely deployed platforms, the Java™ 2 platform Enterprise Edition (J2EE™) and .NET, is paramount for most cases.

In this IBM® Redbook, we will explore the different interoperability models for the IBM WebSphere® and the Microsoft® .NET platforms. This book is a good source of information for solution designers, application integrators, and developers who wish to develop solutions that incorporate interoperability of applications running on WebSphere and .NET platforms.

This book builds on two prior IBM Redbooks™, *WebSphere and .Net Interoperability Using Web Services*, SG24-6395, which is available at:

<http://www.redbooks.ibm.com/abstracts/sg246395.html?Open>

And *WebSphere and .NET Coexistence*, SG24-7027, which is available at:

<http://www.redbooks.ibm.com/abstracts/sg247027.html?Open>

What this redbook is about

This redbook is about:

- ▶ Application interoperability

It is about business and technology drivers for application interoperability and the mapping of multi-tiered architecture models to interoperability models.

This book introduces the concept of Application Interoperability Stack, which defines a structured approach for interoperability considerations and design. Considerations for interoperability include types of application interaction (synchronous or asynchronous), activities, and constraints that are applied to the application interoperability stack to realize interoperability in different scenarios.

- ▶ Component interoperability

It addresses interoperability at the component technology level between J2EE components (Java Bean and Enterprise Java Bean) and .NET components which are classes that implement the System.ComponentModel.IComponent interface.

- ▶ Web Services interoperability

This redbook is also about realizing one of the main promises of Web Services, interoperability across platforms, applications, and programming languages. It focuses on the use of Web Services technologies to achieve interoperability between applications running on WebSphere and .NET platforms.

What this redbook is not about

This redbook is not about:

- ▶ Competitive comparison of IBM WebSphere and Microsoft .NET

This redbook is not about comparing the two technology platforms for technical merits and advantages. And when we do make comparisons, it is for the purpose of identifying the issues that you need to consider as you make interoperability design decisions.

- ▶ Java or C# programming

While this redbook is about interoperability between WebSphere/J2EE and .NET, it is not a tutorial on how to program in Java or Java 2 platform Enterprise Edition technologies. Nor does it address programming in C# or .NET. There are lots of texts that address programming in IBM WebSphere and Microsoft's .NET.

- ▶ Web Services programming

Though we discuss interoperability using Web Services, it is not the goal of this redbook to teach you Web Services programming either on IBM WebSphere or Microsoft's .NET.

The target audience

This redbook is aimed at the diverse set of professionals who have the responsibility for designing and implementing interoperability solutions between applications running on IBM WebSphere and Microsoft's .NET platforms. These include IT architects who have to architect the solution, and IT specialists, application integrators and developers who have to design, code, and test the interoperability solutions.

Structure of this book

Part 1, “Introduction” on page 1 consists of three chapters that provide an overview of application interoperability. Starting with the drivers for interoperability, we identify business and technical reasons for implementing interoperability between IBM WebSphere and the Microsoft .NET platforms. This part introduces you to application interoperability models and the concept of *Application Interoperability Stack*, which provides the structure for analyzing and designing interoperability solutions.

To round-up the Introduction part, we introduce the IBM WebSphere and Microsoft .NET platforms. It is not unusual to discover that while architects, developers and IT specialists working on one of these platforms have garnered much experience on that platform, their exposure to the other platform is either non-existent or just minimal. The introduction provides an overview of both platforms.

Part 2, “Component interoperability” on page 147 addresses interoperability at the component level between Java Bean and Enterprise Java Bean components, and .NET components. This part provides an overview of J2EE and .NET component models and technologies. We follow this with discussions on designing component level interoperability solutions and conclude with a sample component interoperability implementation between WebSphere and .NET.

Part 3, “Web Services interoperability” on page 247 provides an overview of interoperability using Web Services technologies. This part also includes discussions on designing Web Services interoperability and a sample Web Services interoperability implementation between WebSphere and .NET.

Part 4, “Appendices” on page 375 provides additional information and pointers to related publications.

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, Raleigh Center.



The IBM redbook team (left to right): Sudharkar Nagarajan, Vijay Mann, Edward Oguejiofor, Peter Hood.

Edward Oguejiofor is a project leader at the International Technical Support Organization, Raleigh Center. He has over 20 years experience in distributed and enterprise systems architecture and design. His areas of expertise include Web Services and Service-Oriented Architecture. He also provides technical and thought leadership in emerging technologies and collaborative services. He holds a degree in Computer Science from the Imperial College of Science and Technology.

Ken Childers is a software engineer at IBM Austin working in the WebSphere System test group since 2001 developing test cases for WebSphere releases. He has worked on Versions 3.5, 4, 5, 5.1, 6, and 6.x of WebSphere. For the last two years, he has focused mainly on Web Services and interoperability between WebSphere and .NET. Prior to 2001, he worked in software development for several years, developing network stack projects on UNIX® and as a software consultant working on special projects on customer engagements.



David Dhuyvetter is a Senior Software Engineer with the WebSphere Enablement Team. He has been with IBM since 2000, and has over 10 years experience in distributed transactional systems. David has focused on application development and migration, and is a co-author of the migration Redbooks for WebSphere Versions 5 and 6. David received his Master's of Science in Computer Science from California State Polytechnic University, Pomona in 1993.

Peter Hood is a IT Specialist in Australia. He has seven years of experience in IBM A/NZ Global Services working as an IT Specialist. He has experience in a number of technologies, ranging from IBM WebSphere and Microsoft .NET to general Internet based technology. He has utilized a number of different architectures ranging from high-performance n-tier Web applications to more traditional client-server models. As a result of his experience, Peter has been involved in consulting and assisting a number of troubled projects in both short and long term secondments. He works in teams ranging from several people of similar skills to large multi-disciplinary teams with a range of backgrounds and skill-sets. Peter has been involved in the design of many Internet-based applications and been the lead developer in many of these. He holds a Bachelor's degree in Computer Science from Melbourne University and also a Master's degree in Internet and Web computing from the Royal Institute of Technology. Peter has also co-authored *Patterns: Implementing Self-Service in a SOA Environment*, SG24-6680.

Vijay Mann is a Technical Staff Member at the IBM India Research Laboratory in New Delhi, India. He is currently a part of the WebSphere Application Server eXtended Deployment (XD) development team and has been working on the various health monitoring features in XD. He has been with IBM since 2003 and has worked extensively with various WebSphere Application Server offerings in the areas of BPEL4WS based business processes, Web Services, performance tuning, and problem determination. His overall experience includes six years of systems programming in C/C++ and Java using various EAI technologies, such as CORBA, RMI, and SOAP. He holds a Bachelor's degree in Electronics and Telecommunications from Malaviya National Institute of Technology, Jaipur, India and a Master's degree in Computer Engineering from Rutgers, The State University of New Jersey, USA.

Sudhakar Nagarajan is an Advisory Software Engineer, engaged in the WebSphere quality initiative and focusing on IBM middleware product integration in the Software group at Research Triangle Park (RTP), North Carolina. Prior to joining the Software group, he was an IT specialist at Global Services. His background includes over ten years of application design, development, and project management on both mainframe-based and distributed systems across a wide variety of industries and platforms including Government industry. He has

worked extensively with WebSphere, J2EE and Java programming. He holds a Master's degree in Manufacturing Engineering".



Gerd Sommerhäuser is an IT Specialist for Microsoft technologies at the IBM Global Services AMS Organization in Germany. He has been responsible for architecting several global e-business solutions in different Microsoft environments over the last years. Being a core member of IBM Germany's .NET Web Services community, he focuses on his expertise in .NET technologies, e-business and Internet technologies, and is working as a technical and architectural consultant in customer projects.

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

From the International Technical Support Organization, Raleigh Center:

- ▶ Saida Davis
- ▶ Carla Sadtler
- ▶ Margaret Ticknor
- ▶ Jeanne S Tucker

From International Technical Support Organization, Poughkeepsie Center:

Cheryl Pecchia

From IBM WebSphere SVT Group, Austin, Texas:

- ▶ Huiran Wang
- ▶ Tam Dinh

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 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 e-mail 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



Part 1

Introduction

Application interoperability overview

The heterogeneous computing environment that exists either within organizations or across the extended enterprise make support for application interoperability essential for driving end-to-end business processes, providing seamless experience for customers, and for collaborating with business partners and suppliers.

In this chapter, we provide an overview of application interoperability starting from the drivers for interoperability, and the review of different interoperability models. Finally, we introduce interoperability taxonomy and structures for analyzing and designing interoperability solutions.

This chapter contains the following sections:

- ▶ Drivers for application interoperability
- ▶ Application interoperability models
- ▶ Approaches for interoperability
- ▶ Elements of interoperability

1.1 Drivers for application interoperability

If your organization is anything like most, your information technology (IT) department has defined your enterprise architecture, which specifies your IT principles, guidelines, and standards. And your IT standards include the technology platform that is approved and mandated for use throughout your organization. In most cases, the technology platform is either J2EE or .NET, the two most widely deployed platforms today.

Note: The focus of this redbook is on interoperability between WebSphere, the IBM implementation of the J2EE specification, and Microsoft's .NET. You will find that these drivers hold true regardless of the technology platforms.

Having standardized on a technology platform, it takes compelling reasons for an organization to deviate from its established standard. IT organizations usually find themselves constrained by budget cuts while facing increasing pressures to deliver solutions to the business. One of the most common challenges that you face under these conditions is that of developing solutions in a heterogeneous environment. It quickly becomes obvious that ripping and replacing pieces of your systems in order to conform to the prescribed standard technology platform is not the safe and practical choice.

The reasons for interoperability vary from organization to organization and from one given situation to another; however, they tend to fall into broad categories, some of which include the following.

1.1.1 Technology transition

Companies constantly undergo change. Whether business or technology driven, some of the changes result in technology transition. The changes may impact the technology platform and when they do, it is not always feasible to ensure the continued use of the mandated technology platform.

Mergers or acquisitions

IT departments are constantly faced with managing and executing the technology transition following a merger or an acquisition. In addition to integrating disparate and distributed systems infrastructure, IT departments are expected to show rapid return on investment by seamlessly integrating the enterprise applications and business processes.

Interoperability is perhaps the best means for meeting the goal of providing a single system view to customers, employees, and business partners of the merged company with minimal disturbance.

Platform migration

When making technology platforms shift, for example, changing the back end of an e-commerce Web site from Microsoft .NET to WebSphere in order to take advantage of WebSphere's scalability, your transition strategy may require that you run the two systems in parallel. If that is the case, interoperability becomes a necessity.

Emerging technologies

The emergence of certain technologies are such that to remain competitive you must adopt the new technology immediately. The World Wide Web is an example of such technology. Web Services is another example and in Chapter 7, "Introduction to Web Services Interoperability" on page 249, we address interoperability using Web Services.

1.1.2 Best of breed integration

One approach many companies adopted for their e-business enablement to reduce costs, increase productivity, and ensure return on investment was the deployment of *Best of Breed* applications.

Best of Breed (BoB) applications provide higher quality and deeper functionality for each business function. However, to take advantage of the BoB applications, you have to enable information sharing between the different applications. Application interoperability offer the means for integrating the various BoB applications and your existing applications.

1.1.3 The extended enterprise

To gain collaborative advantage, enterprises are extending their business processes and IT systems beyond the organizational boundary to include business partners, suppliers, and even customers.

The extent to which information flows seamlessly through the extended enterprise depends on the interoperability between disparate systems and applications.

1.2 Application interoperability models

A quick review of the drivers for interoperability reveal how prevalent the need for interoperability is in today's heterogeneous and networked systems environment. It is inevitable, therefore, that sooner or later you will be faced with having to implement interoperability between systems. So it is necessary to consider interoperability as an integral part of your IT systems design and not as an after thought.

Quoting from *ISO/IEC 2382-01, Information Technology Vocabulary, Fundamental Terms*¹, interoperability is defined as follows:

"The capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units".

It is not practical for us to address every possible interoperability scenario in this redbook, so we introduce interoperability models as a means for categorizing interoperability scenarios. We do so through interoperability points that map to the n-tier application architecture model.

1.2.1 Application architecture model

The preferred architecture model for today's enterprise scale system is the client server model, specifically, the n-tier client server architecture model. In an n-tier architected system, well defined layers form tiers, each performing a distinct and separate function. In addition to making it possible to reuse functionality and distribute the application in a network environment, multiple tiers make it easier to implement interoperability. The different functions in each tier can be designed, developed, and deployed on different technology platforms.

Most client server applications consists of three tiers:

- ▶ Presentation
- ▶ Business logic
- ▶ Data tiers

However, n-tier applications can have more tiers. Both WebSphere (J2EE) and .NET support multi-tiered architecture models that can include one or more sub-tiers in the presentation, business logic, or data tiers.

¹ <http://jtc1sc36.org/doc/36N0646.pdf>

Figure 1-1 shows the n-tier architecture model.

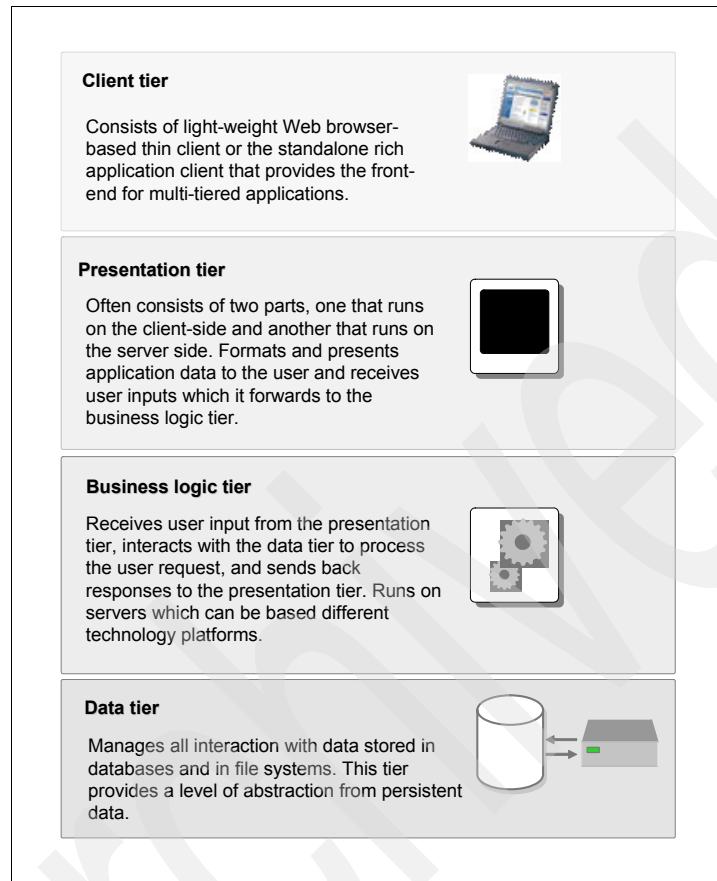


Figure 1-1 N-tier architecture model

Client tier

Client tiers support three basic types of client applications. These are the light-weight Web browser based thin clients, the stand-alone rich application clients, and smart clients.

► Thin application clients

Usually browser based. The browser renders the user interface and typically hosts components and a scripting engine that provide complex presentation logic.

► Rich application clients

Are stand-alone applications that run on client machines and provide user interface functionality, such as toolbars, menubars, support drag and drop, context sensitive help, and undo/redo.

► Smart Clients

You can also have smart clients, which are simply an evolution of rich clients combined with thin client features. Smart clients use compiled code to provide application user interface and client logic without relying on the Web browser.

Figure 1-2 shows clients in a multi-tier architecture model.

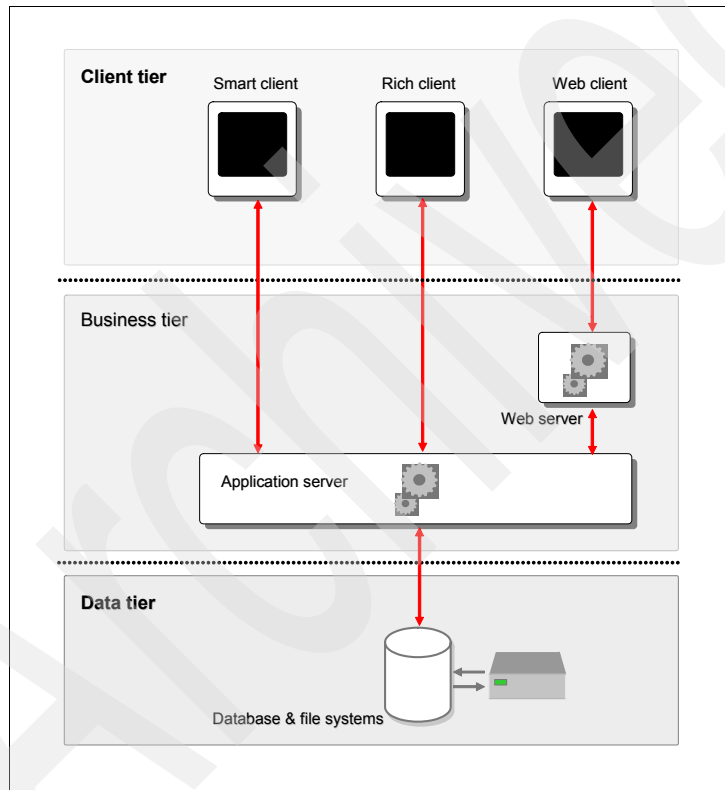


Figure 1-2 Clients in a multi-tier architecture model

Presentation tier

The presentation tier drives the application user interface and captures all user interactions. It handles the visual presentations, accessibility, translation of user inputs to application events, invocation of services in the business logic tier, and management of interaction state.

The presentation tier is distributed between the client and the server. The client-side of presentation tier leverages the client tier to provide the application user experience. The server-side presentation tier interfaces with the business logic tier and manages the interaction between the user and the application logic.

Business logic tier

In the classic 3-tier architecture model, there is a single business logic tier. However, it is typical to have sub-tiers, such as the Web tier in this middle layer. The business logic tier implements the core application functionality, which usually makes use of the component model for the technology platform. For example, COM+ for .NET and Enterprise JavaBeans™ (EJB™) for WebSphere and J2EE. These business tier components run on Application Servers, which provides the platform for the components.

Data tier

The data tier provides a level of abstraction from all persistent data and manages all interaction with data stored in databases and in file systems. Its primary function is retrieving, storing, and updating of data. This tier provides data services that can be accessed mainly from the business logic and in some cases from the presentation tier as well.

1.2.2 Interoperability scenarios

In the previous section, we presented the n-tier architecture model. Each tier in the n-tier model is a possible interoperability point between WebSphere and .NET applications. An enumeration of the platform and tier interactions show that not all possible scenarios are feasible. In this section, we present and discuss the different interactions and identify the scenarios that are feasible and those that are highly improbable.

Note: We explicitly discuss *one-to-one* interoperability between WebSphere applications and .NET applications. Depending on the scale of your system, it is likely that you will need to consider potentially added complexities (such as concurrency and quality of service) of *many-to-many*, *one-to-many*, or *many-to-one* runtime interoperability dynamics.

Client tier interoperability

Interoperability between the client tier and other tiers in an n-tier architected application scenario depend on the type of client. For example, while client tier to client tier interoperability between two thin client applications is impractical, interoperability between the client tier and the other tiers (business logic and data tiers) is feasible.

Client to client tier interoperability

Direct interoperability between a WebSphere and J2EE thin client application with a .NET thin client application is highly improbable. Smart and rich clients interoperability, however, are more likely. A Java rich client application, such as a Swing or an AWT application (possibly deployed in the WebSphere client container), can interoperate in a peer-to-peer setting with a .NET rich client application, where each client application exposes specialized functions.

A potential hybrid scenario is a peer-to-peer setting with a thin client application interoperating with a rich client application (via http). With this scenario, in addition to exposing specialized functions, the rich client application also functions as a pseudo Presentation tier to the thin client and incorporates a Web server (see Figure 1-3).

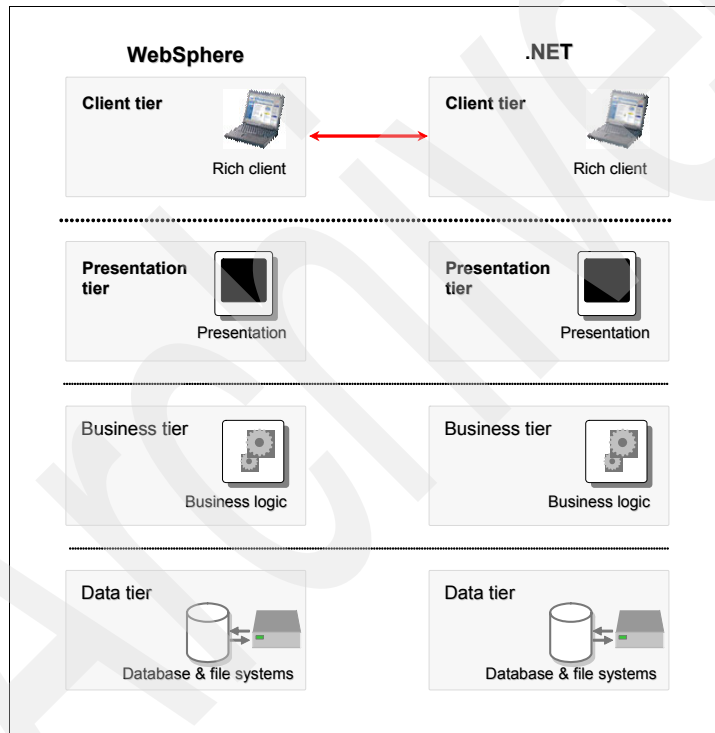


Figure 1-3 Client tier to client tier interoperability

Although interoperability between client tiers is technically possible, it is highly unlikely that you will design or implement client tier to client tier enterprise scale solutions between WebSphere and .NET.

Client tier to presentation tier interoperability

Interaction between the client tier and the presentation tier is a normal occurrence in an n-tier architected application. In a WebSphere and .NET interoperability setting, you will find that the need for a WebSphere client to interoperate with a .NET presentation tier or vice versa will occur quite frequently. However, the scenario where the client and the presentation tiers are implemented on the same platform and interoperating with a business logic tier that is implemented using the other technology platform is the most commonly occurring interoperability scenario.

► WebSphere client tier to .NET presentation tier

In this scenario, you can have a Java thin client application (possibly a Java Applet) or a rich (or smart) client, such as a Swing (or AWT) client application deployed in the WebSphere client container, interoperating with the presentation tier of a .NET application (possibly an ASP.NET artifact deployed in IIS) (see Figure 1-4).

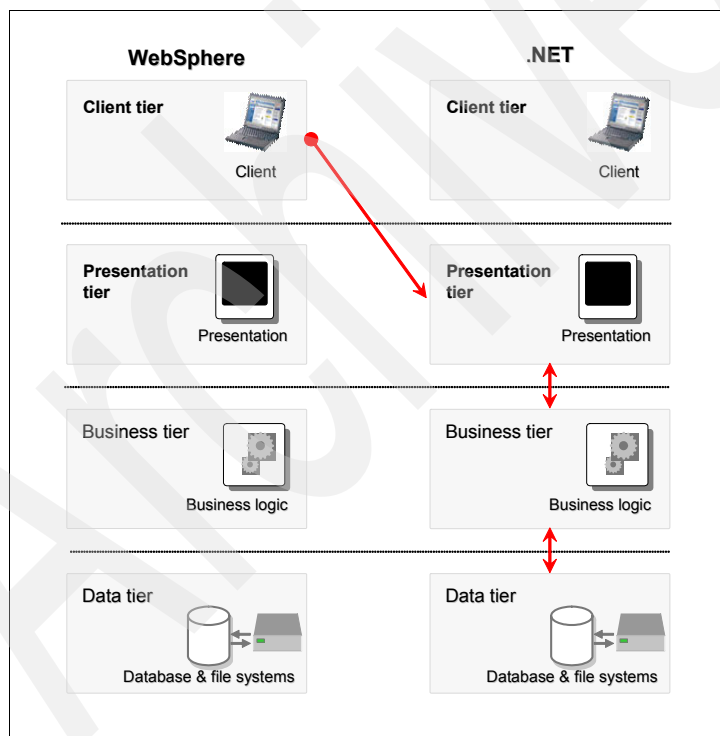


Figure 1-4 WebSphere client to .NET presentation tier interoperability

► .NET client tier to WebSphere presentation tier

A .NET client application, for example, a rich or smart client application, such as a Windows® Form application interoperating with a Java application (possibly a JSP™ or Servlet) deployed in the WebSphere Web container.

Figure 1-5 gives an overview of this tier.

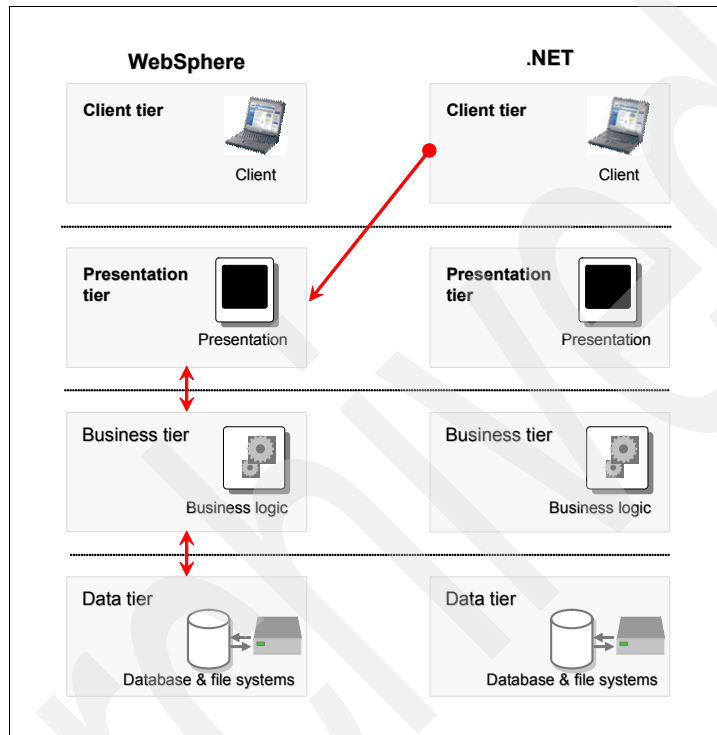


Figure 1-5 .NET client to WebSphere presentation tier interoperability

Client to business tier interoperability

This is a frequently occurring scenario where you have the need to replace the technology platform for your business logic tier while preserving the client tier applications.

A typical example of this type of scenario is a two-tier application, where a rich client application interacts with the business logic that resides on the server, for example, a rich (or smart) client, such as a Swing client application or an AWT client application, deployed in the WebSphere client container, interoperating with .NET business tier components. Or a .NET rich (or smart) client application, such as a Windows Form application, interoperating with Servlets and EJBs deployed in the WebSphere Web and EJB containers.

Figure 1-6 gives an overview of this interoperability.

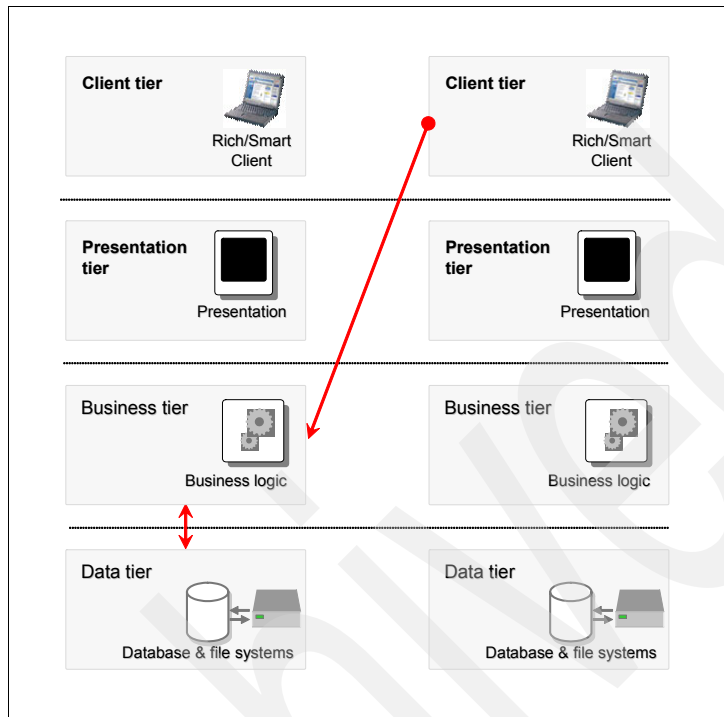


Figure 1-6 Rich and smart client to business tier interoperability

Thin client applications also have the capability to interoperate directly with the business logic tier. With the growing support for XMLHttpRequest object, which is implemented by most browsers using the Asynchronous JavaScript™ and XML (AJAX) design pattern, you can implement interoperability from a thin client application to the business logic tier.

Using JavaScript and XML serializers and deserializers, you can create an AJAX based browser SOAP Web Services client that can be used to invoke Web Services in the business and data tiers (see Figure 1-7).

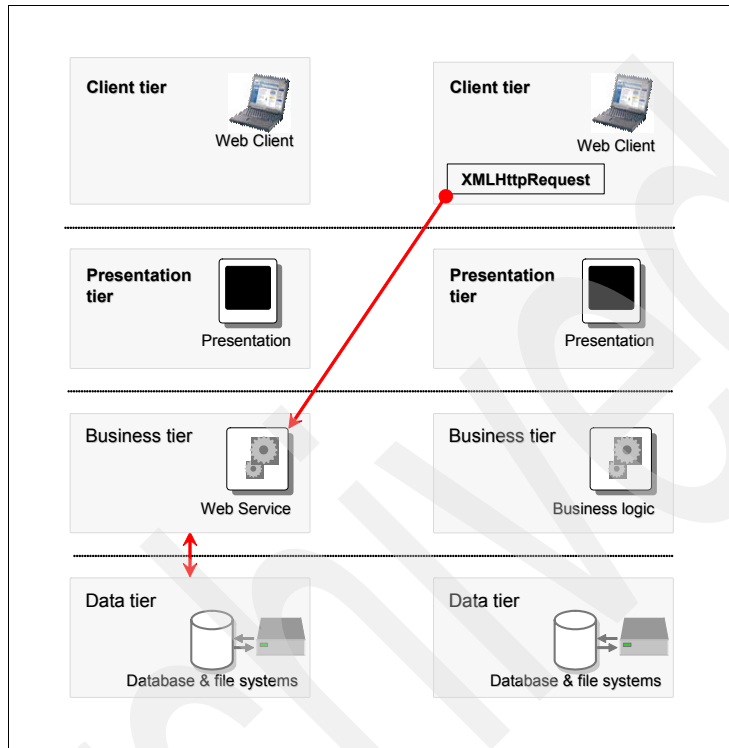


Figure 1-7 Client tier to business logic tier interoperability using AJAX design pattern

Using DHTML behaviors, you can also enable client tier to business logic tier interoperability. The use of Web Services behavior in the client tier application enables invocation of business tier functionality exposed as Web Services using SOAP and Web Services Description Language (WSDL) (see Figure 1-8 on page 15).

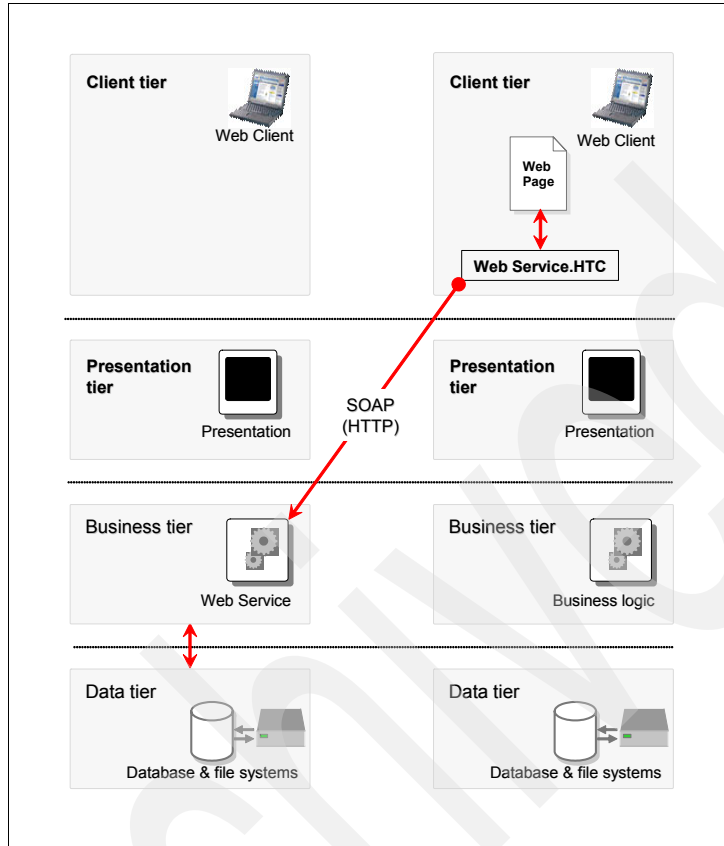


Figure 1-8 Client tier to business logic tier interoperability using DHTML behaviors

Client to data tier interoperability

By their very nature, rich and smart client applications can implement the entire application logic in the client piece and interact with the data tier simply for persistence. This two tier client application scenario is true of both WebSphere and .NET platforms. In this scenario, a client implemented in .NET or WebSphere J2EE platform, interoperates with resources in the data tier that is running on the other platform (see Figure 1-9).

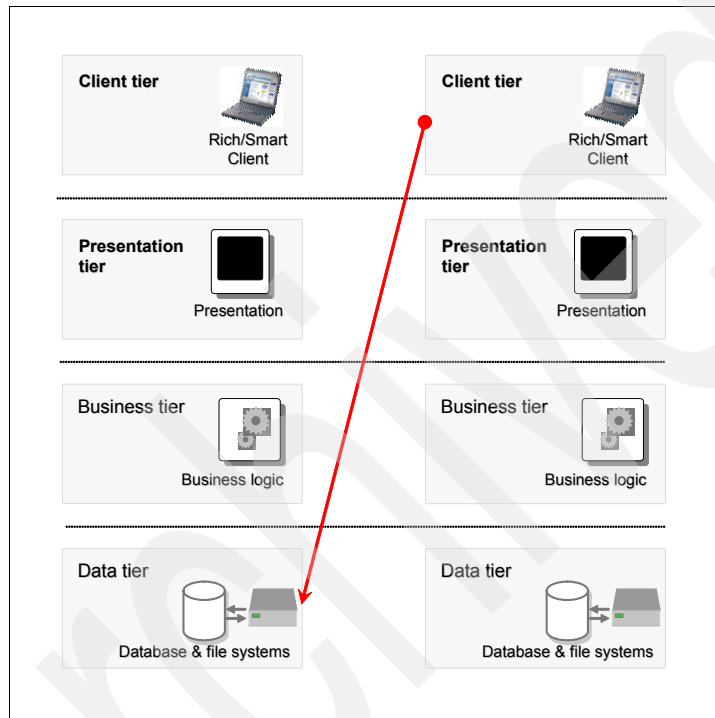


Figure 1-9 Two tier rich client application: client to data tier interoperability

The client tier to data tier interaction using browser thin clients can be implemented using the AJAX design pattern or DHTML.

Note: Although the technical capability is there to implement client to data tier interoperability, unless you are developing a simple data driven application, you should not embark on this track for an enterprise scale distributed system.

Presentation tier interoperability

The presentation tier provides the management, coordination, and orchestration of the user interface and user interactions in n-tier architected applications. It renders the application's visual appearance and layout, maintains state through user sessions and business logic process flow, and invokes services in the business logic tier.

In 2-tier and 3-tier applications, the presentation tier usually includes the client tier as well. However, in n-tier applications, where the two tiers exist separately, you can have client tier to presentation tier interoperability (see “Client tier to presentation tier interoperability” on page 11). The following are additional cross tier interactions from the presentation tier.

Presentation tier to presentation tier

Presentation tier interaction results in scenarios where WebSphere presentation components, such as JSP, servlet, or JavaBeans deployed in the Web container, interoperate with ASP.NET or underpinning assemblies deployed in IIS.

Applications in this scenarios tend to be thin client applications (rich or smart client presentation tier interoperability are similar to client tier interoperability in “Client to client tier interoperability” on page 10). This scenario presents a number of issues, such as session and state management, making it impractical for creating robust and scalable solutions between WebSphere and .NET (see Figure 1-10).

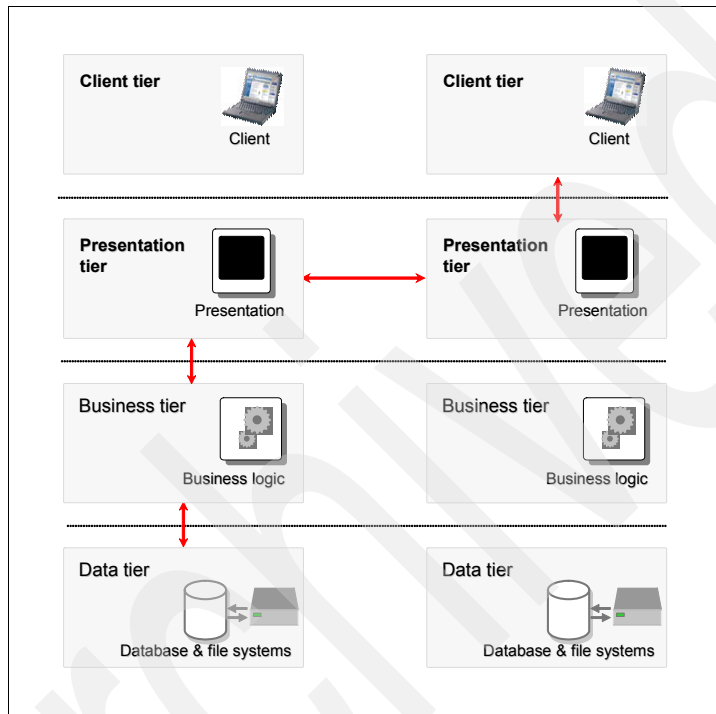


Figure 1-10 Presentation to presentation tier interoperability

One practical use of presentation tier interoperability is redirection of URLs. URL redirection is used for a number of reasons, such as forcing HTTP requests to HTTPS or redirecting requests sent to a URL and meant for the associated Web server so the request is handled by another Web server. This can be realized through client side redirection where the Web server responds with a Location: header directing the client to the second URL, or through the server side redirecting or forwarding. For example, a Web server can transparently rewrite a requested URL for a WebSphere Web site to a .NET Web site.

Presentation tier to business tier

This is the most common occurring interoperability scenario between WebSphere and .NET. Consider, for example, the scenario where you replace your .NET business tier with WebSphere J2EE implementation just so you can take advantage of WebSphere security, reliability, and scalability. Your ASP.NET presentation tier application will have to interoperate with Servlets and Enterprise JavaBeans (EJB) components, which implements the business functions. You can equally have the scenario where JSP presentation tier application interoperates with COM+ components.

In Chapter 7, “Introduction to Web Services Interoperability” on page 249 and Chapter 9, “Web Services interoperability scenario” on page 293, we present example scenarios that show implementation detail for presentation tier to business tier interoperability going from WebSphere to .NET and vice versa (see Figure 1-11).

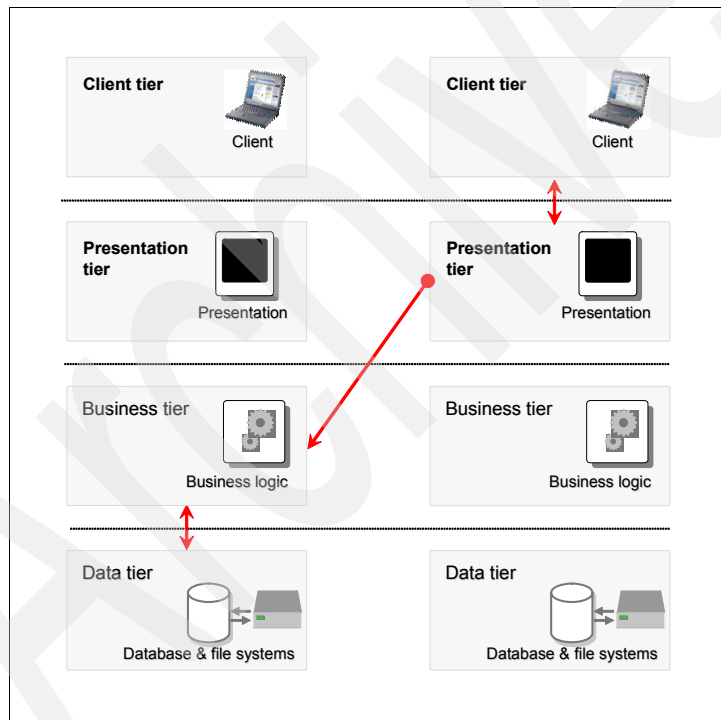


Figure 1-11 Presentation tier to business tier interoperability

Presentation tier to data tier

Data driven Web sites that display static and dynamic information are examples of this type of scenario. While data driven applications such as online shopping, online auctions, and business to business applications do include significant application logic that reside in the business tier, a great deal of useful data driven applications have little or no business logic.

Imagine that you have to build a dynamic Web application that generates pages on the fly from data residing in a database and accepts user data for insertion to the database or for updating of records already in the database. With such an application, where you have no business logic, you bypass the business tier and go directly to the data tier from the presentation tier.

Consider that you have a WebSphere data driven application (using Java Server Faces (JSF) or JSP) that accesses data from .NET using ADO.NET. Or a .NET application (using WebForm, WinForm or ASP.NET) which interoperates with WebSphere using JDBC™ or SDO for data access (see Figure 1-12).

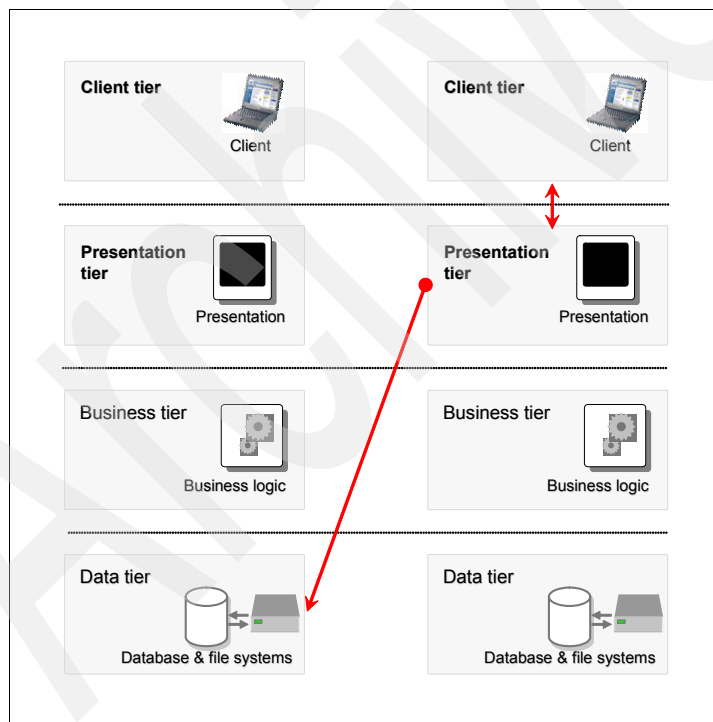


Figure 1-12 Presentation tier to data tier interoperability

Business tier interoperability

Interoperability between business tiers is driven primarily by the need to reuse business functions that already exist. The single most important benefit of reusing business function is the cost avoidance in duplicating the functionality. You also avoid the high degree of risk typically involved in developing new software. In some cases, such as credit checking and shipping tracking in e-commerce applications, you may not have the option to reimplement the business function.

The business tier include business functions and processes that create, process, and persist data. These processes and functions rely on the data tier for access to and persistence of data. Hence, interoperability between the business logic and data tiers is a necessity.

Business tier to business tier

Imagine that you have an e-commerce application where you have a .NET front end that includes your order processing and to take advantage of WebSphere's robust scalability, your fulfilment, CRM, and supply chain applications are implemented using WebSphere. Consider that your end-to-end e-commerce solution will have .NET COM+ components interoperating with Servlets and EJBs running in WebSphere EJB containers, as well as third-party services, such as credit checking, to seamlessly meet your customer needs (see Figure 1-13).

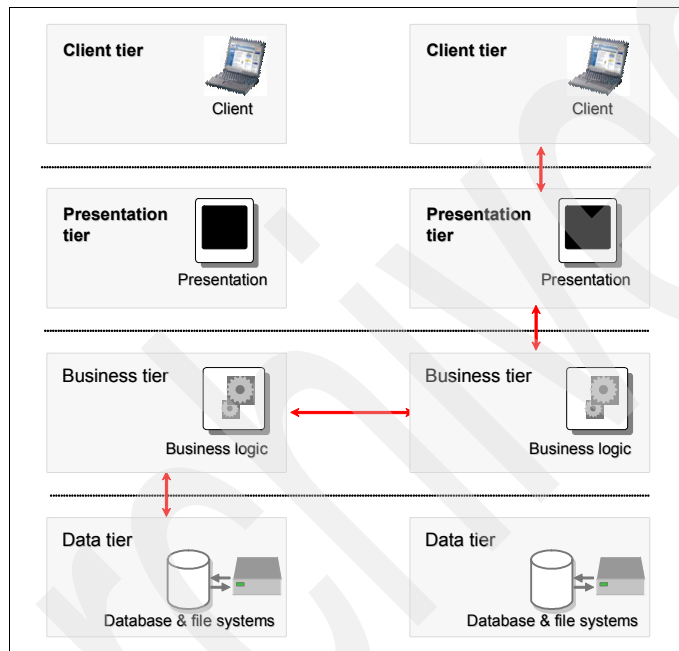


Figure 1-13 *Business tier to business tier interoperability*

Business tier to data tier

The data resource tier in the .NET architecture is primarily about accessing data and interacting with resources using ADO.NET. However, it also includes Services that are used to connect to systems external to .NET through *Service Agents*.

The data tier in J2EE n-tier architecture is the Enterprise Information Systems (EIS) tier. EIS provide connectivity to enterprise systems (for example, ERP systems) and supports business-to-business (B2B) transactions using the J2EE Connector Architecture (JCA). EIS is similar to "Services" in the .NET data tier. Persistence and access to data in J2EE is realized through the use of Entity Enterprise JavaBeans, which runs in the business tier.

Note: Persistence in J2EE is handled by use of Entity Enterprise JavaBeans. This is either container managed, where the container implements and handles the persistence, or component managed, where the developer of the component provides the code to handle persistence. Entity EJBs run in the EJB container in the business logic tier. This is in contrast to .NET, where all persistence is handled in the data tier.

Interoperability between the business and data tiers is not necessary for access to and persistence of data only. Consider a Servlet or EJB running in a WebSphere EJB container interoperating with .NET data tier; the Servlet or EJB will be interacting with either ADO.NET to access data that is persisted in .NET, or with a Service Agent to a service that is external to .NET.

On the other hand, a .NET business function component (COM+ component or assembly) running in the business tier and interacting with WebSphere data tier will, in effect, be interoperating with an external system through the J2EE Connector Architecture (JCA).

Note: In the case where the business tier is interacting with either Services in .NET or EIS in J2EE, the interoperability is similar to business tier to business tier interoperability.

Figure 1-14 gives an overview of this interoperability.

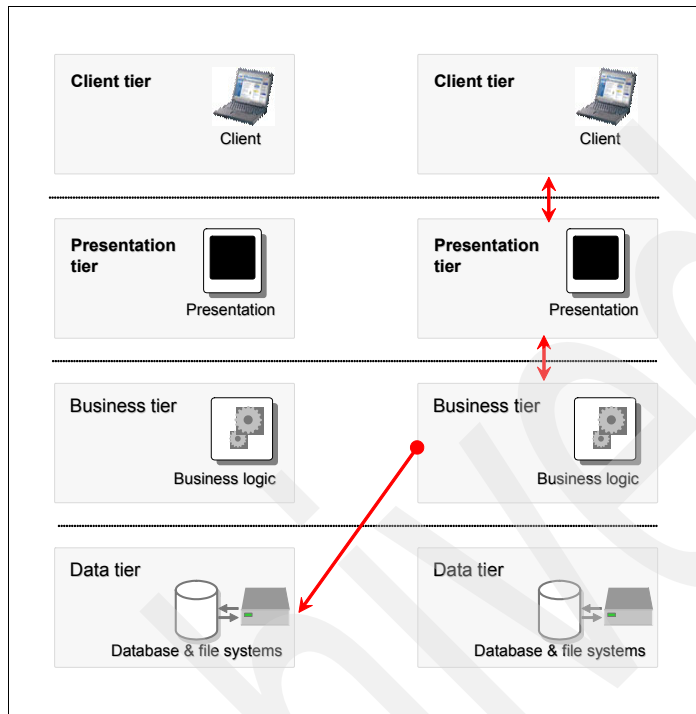


Figure 1-14 Business tier to data tier interoperability

Data tier interoperability

The data tier in .NET performs two distinct functions. The first is to access and persist data through ADO.NET. The second function is to access services external to .NET through *Service adapters*.

The WebSphere equivalent of the data tier is the Enterprise Information Systems (EIS) tier. EIS provides access to external systems just like Services in .NET. Unlike .NET, the WebSphere data access and persistence is provided mainly through Entity Enterprise JavaBeans, which run in the business logic tier.

In addition to typical data access, interoperating at the data tier is usually associated with accessing business functions or processes through *Service adapters* in .NET or using the J2EE Connector Architecture (JCA) to connect to enterprise systems or any other systems for which there is an adapter.

Data tier to data tier

Data tier interoperability between WebSphere and .NET enable interoperability between WebSphere EIS and Services in .NET. By interoperating through WebSphere EIS or .NET Services, it is possible to make calls to and interoperating with yet a third system.

Note: You can develop a J2EE Connector Architecture or .NET “Service” adapter for connecting to any system or service of your choice.

1.3 Approaches for interoperability

Interoperability is the capability to communicate, execute programs, or transfer data, and there are a number of options for accomplishing this. These options fall into two basic types: the first is service-oriented interoperability approach, and the other is the class level approach. The service-oriented approach makes use of Web Services technologies to expose functionality in either WebSphere or .NET to the other platform. The second approach is interoperability at the class level, where Java or .NET classes can make calls to the other seamlessly.

1.3.1 Service-oriented approach

The Web Services model, specifications, and service-oriented distributed architecture are widely adopted industry approaches for building systems that interoperate. This is certainly one approach for implementing WebSphere and .NET interoperability. You can have your ASP.NET presentation tier application or functions running in the business logic tier interoperating with Web Services deployed in the WebSphere business tier. The reverse is also applicable, where you have JSP application or Servlets and EJBs running in the WebSphere Web or EJB containers interoperating with Web Services deployed in .NET. We provide a detailed discussion of the Web Service interoperability approach in Part 3, “Web Services interoperability” on page 247.

1.3.2 Class level approach

The other approach for interoperability between WebSphere and .NET is the class level approach. With this approach, you are able to call .NET classes from Java and vice versa. There are a number of options available for accomplishing class level interoperability, which include the following.

Porting

This is a rather tedious approach where you port the entire .NET framework to Java or vice versa. Both Java and .NET platforms are very large with thousands of classes to port. The hard work does not end with that the initial port and its accurate implementation; you still have the difficult task of repeating the effort with each new release of Java or .NET platform. There are also issues with supporting namespaces that are tied too closely to .NET runtime.

Cross compilation

Java and any language supported in .NET can be compiled into an intermediate bytecode. You can, for example, cross compile a Java class into the Microsoft Intermediate Language (MSIL) that runs in the .NET Common Language Runtime (CLR). With .NET cross language capability, you can then call or inherit the Java class from any other class written in, say, C# or Visual Basic®. You can also convert the MSIL of a .NET C# or Visual Basic class into Java bytecode, which then executes in a Java Virtual Machine.

Remote procedure calls

You can use the remote procedure call mechanism available on either Java or .NET to create interoperability solutions. Java offers the Java Remote Method Invocation API, or Java RMI. Using Java RMI, you can perform remote procedure calls where an object on one Java virtual machine invokes methods on an object in another Java virtual machine. You can introduce a bridge that will enable you to use Java RMI to create binary and XML interoperability with .NET applications.

On the .NET side, you have .NET remoting, which is the .NET framework for allowing objects in different AppDomains, processes, and in different machines to communicate. You can use .NET remoting to implement binary and XML communication between Java/J2EE and .NET systems.

Bridging

The bridging approach for class level interoperability enables Java classes running in a Java Virtual machine (JVM™) and .NET classes running in a CLR to seamlessly access and invoke each other. This is accomplished through bridging software, which exposes classes on one platform to the other through the use of proxy classes that are created on each platform. The proxy classes appear and behave as the native classes and can be inherited from.

Messaging and queuing

Messaging and queuing allow applications running on different platforms and at different times to communicate. The IBM WebSphere MQ and Microsoft's MSMQ are messaging and queuing products that you can use to implement interoperability between Java in WebSphere and .NET, exchanging both binary and XML messages.

Note: You can also implement Web Services interoperability between WebSphere and .NET using WebSphere MQ or Microsoft MSMQ. WebSphere MQ can be 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.

For more details, see the IBM Redbook *WebSphere MQ Solutions in a Microsoft .NET Environment*, SG24-7012.

Part 2, “Component interoperability” on page 147 provides a detailed discussion of the class level interoperability approach.

1.4 Elements of interoperability

Whether you implement Web Services or class level interoperability, there are a number of steps that are common to both approaches. Regardless of the approach, you have to pass data from either the Java application to the .NET application or vice versa. You need a mechanism for the control of function calls and the transport of data between the two platforms.

We identified three core interoperability elements. These are the application interoperability stack, which categorizes the different functions for interoperability, the set of activities performed in each of the functional category in the stack, and the constraints that are applied to the activities to ensure best interoperability.

1.4.1 Application interoperability stack

The application interoperability stack consist of four layers, which represent key functions for communicating and exchanging data between systems for all interoperability scenarios. The layers include the application layer, which provides the interface with your application semantics, the data layer, which ensures that the syntactic and semantic aspects are maintained for data exchanged between the systems, the control layer, which provides the mechanism for making remote functions available for local invocation, and finally

the transport layer, which handles the different means for transporting data and function calls between interoperating platforms.

Figure 1-15 gives an overview of the stack.

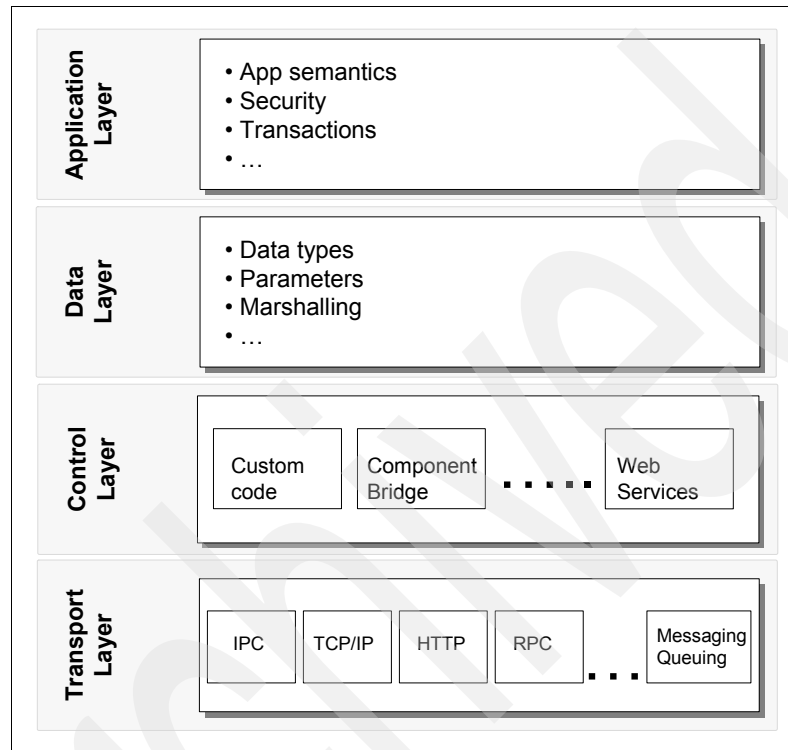


Figure 1-15 Application interoperability stack

Application layer

The application layer is where you extend your applications semantics to the remote system. The application interaction logic, such as synchronous and asynchronous interactions, transaction management, and security constraints, are all handled in this layer. You will also implement all necessary state management for synchronous and asynchronous interactions, and transactions in this layer.

Note: Interoperability between IBM WebSphere and Microsoft .NET platforms include definition of activities that implement the flow and execution of business processes in the application layer. However, we do not discuss business process interoperability in this redbook.

Data layer

An important aspect of application interoperability is the exchange of messages between the interoperating applications. These include the input parameters and return values to function calls. In a distributed and heterogeneous computing environment, you have to ensure that the data passed between systems are matched syntactically and semantically.

The data layer provides the following interoperability services:

- Data serialization and marshalling

Technology platforms have their unique data types and data representation. The data layer provides the serialization and marshalling services to ensure accurate syntactic and semantic exchange of data between interoperating platforms.

- Data access performance

The data layer also implements a service that supports data access strategies with performance considerations, such as data aggregation, chunky, or chatty data access.

Control layer

The control layer is where the implementation of the functional and procedural means for interoperating between disparate systems is realized. Any one of the different class level interoperability mechanisms or Web Service technology can be used.

Transport layer

This layer provides reliable transfer between interoperating applications. It includes the different mechanisms by which the interoperating applications can exchange data, ranging from shared memory for processes running on the same machine to asynchronous messaging using messaging and queuing technology.

1.4.2 Activities

Associated with each layer in the application interoperability stack are a set of activities. Activities are the actual operations that comprise a class level Web Services interoperability. For example, Web Services activities for the control layer include writing and processing SOAP envelope.

1.4.3 Constraints

Where the application interoperability stack and activities define the functions and operations necessary for Web Services interaction, constraints are applied to each activity specific to interoperability.

Constraints vary depending on the type of interoperability (class level or Web Services), the activity, and application requirements. Security constraints are example of application requirements constraints.

Introduction to J2EE and WebSphere platform

The purpose of this chapter is to provide an overview of the Java 2 Enterprise Edition (J2EE) specifications and IBM's implementation of the specifications. This chapter is for readers who have experience developing applications using Microsoft .NET framework and need to become familiar with J2EE and the IBM WebSphere platform.

This chapter contains the following sections:

- ▶ Introduction to J2EE
- ▶ The WebSphere platform
- ▶ IBM Rational Software Development Platform

2.1 Introduction to J2EE

J2EE is a comprehensive set of specifications for designing, developing, and deploying multitier, server-based applications. The J2EE specifications were designed through an open process that involved a large number of contributors. The J2EE specification defines the following key elements:

- ▶ Standard application model for developing multitier applications
- ▶ Standard platform for hosting applications
- ▶ Compatibility test suite for verifying that J2EE products comply with the J2EE specification
- ▶ Reference implementation that provides an operational definition of J2EE

The J2EE multi-tiered architecture defines a client tier, a middle tier (consisting of one or more sub tiers), and an enterprise information system back-end tier. The client tier supports two basic types of client applications:

- ▶ Web applications
These use Web browsers as the client software.
- ▶ Stand-alone client applications
These are full fledged client side applications that are more often than not written in Java. The stand-alone client application runs on client machines and provides user interface functionalities, such as toolbars, menu bars, support for drag and drop, context sensitive help, and undo/redo.

The middle tier includes two sub-tiers:

- ▶ The Web tier
- ▶ The Enterprise JavaBeans (EJB) tier

This tier provides business logic, client, component, and data services through Web and EJB containers. The enterprise information system (EIS) tier supports access to information sources and external systems.

Figure 2-1 on page 33 shows the J2EE architecture.

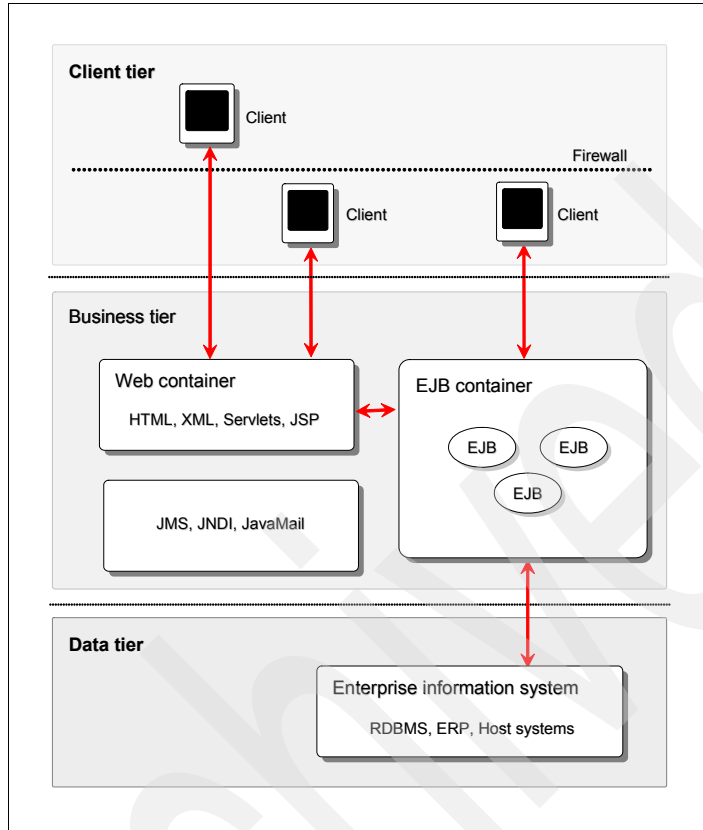


Figure 2-1 J2EE architecture overview

2.1.1 Roles in J2EE environment

The J2EE specification defines a number of distinct roles in the application development and deployment workflow. The roles serve to identify the tasks to be performed and the responsibilities of various team members involved in the development, deployment, and management of J2EE applications. Understanding this separation of roles is important when developing and deploying J2EE applications.

The following are the defined J2EE platform roles:

► Product provider

The product provider designs and offers the J2EE platform, APIs, and other features defined in the J2EE specification for purchase. Providers are typically system, database, application server, or Web server vendors. They implement products that meet the J2EE specifications, including component containers and J2EE platform APIs.

► Tool provider

The tool provider creates tools used for the development and packaging of application components.

► Application component provider

The application component provider creates Web components, enterprise beans, applets, or application clients to use in J2EE applications. Application Component Providers produce the building blocks of a J2EE application. They typically have expertise in developing reusable components as well as sufficient business domain knowledge.

► Application assembler

The application assembler takes a set of components developed by component providers and assembles them in the form of an enterprise archive (EAR) file.

► Deployer

The developer is responsible for deploying an enterprise application into a specific operational environment.

► System administrator

The system administrator is responsible for the operational environment on which the application runs.

Figure 2-2 on page 35 shows the J2EE application life cycle.

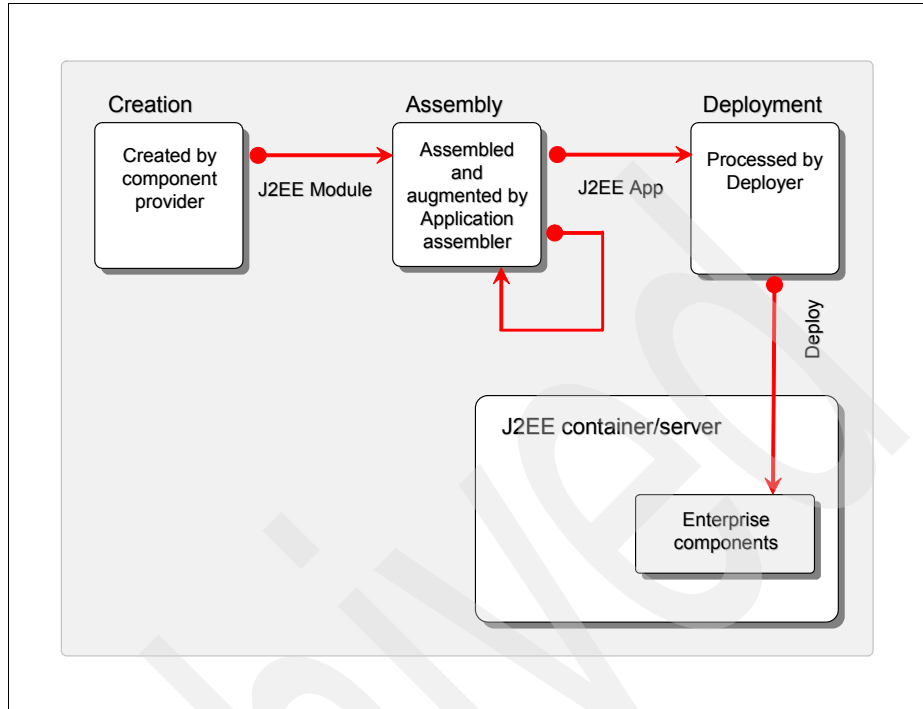


Figure 2-2 J2EE application life cycle

2.1.2 J2EE n-tier architecture

The J2EE platform defines an n-tiered architecture that consists of a presentation tier (the client tier), two business logic tiers (the Web and EJB sub tiers), and the data resource tier (Enterprise Information System). In the J2EE world, applications are often designed using the so called Model View Controller (MVC) paradigm. The MVC architecture pattern decouples data access, business logic, data presentation and the user interaction. The model maintains the business logic and data. Views render the content of models, and maintain visual consistency of changes in the model. Views also capture and forward user inputs to the controller. Controllers define application behavior and interpret user inputs into actions to be performed by the model. In the world of J2EE applications, views are implemented using `JavaServer™ Pages™`, controllers are typically implemented using `Servlets`, and models are realized with `Enterprise JavaBeans`.

Figure 2-3 shows the MVC architecture pattern.

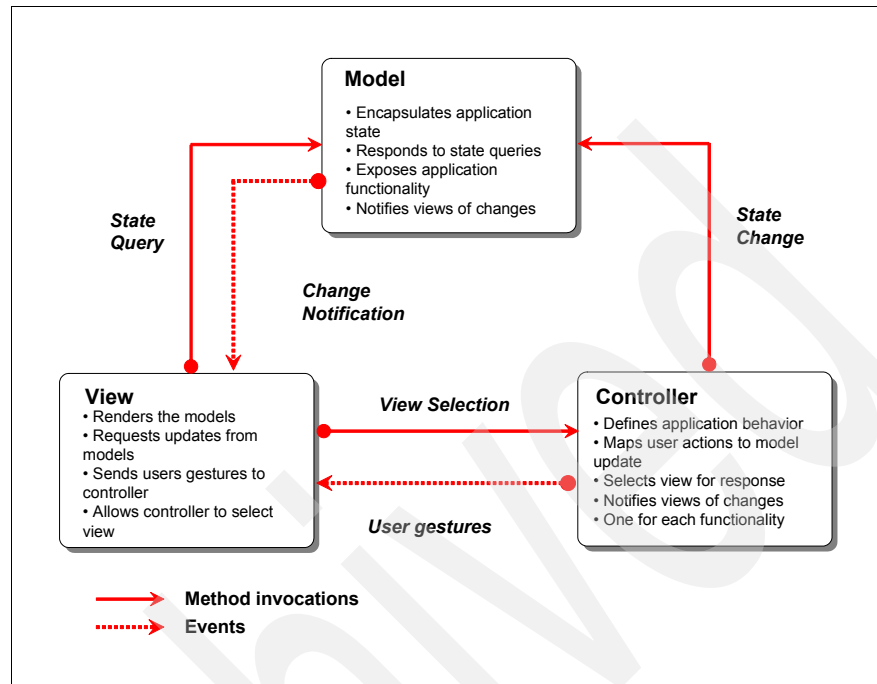


Figure 2-3 MVC architecture pattern

J2EE presentation tier

The J2EE client layer hosts client applications. Client applications perform a number of functions, including the following:

- ▶ Presenting the user interface
- ▶ Validating of user inputs
- ▶ Communicating with the server
- ▶ Managing conversational state

Two types of clients are supported: Web clients and application clients. They are explained as follows:

- ▶ Web clients

Java Web clients run in Web browsers and render the Web tier content in applications, and the browsing experience is enhanced through Applets and JavaScript. Applets are GUI components used to create a highly dynamic user interface for J2EE applications. Although they can execute in any environment that supports the applet programming model, applets typically execute in a Web browsers. Communication between a browser-based applet

and the server side is over HTTP, while the data interchange is facilitated through XML. Non-Java clients written in, say, C++ or Visual Basic can communicate with and make use of the J2EE Web and EJB tiers. They either run in a browser or provide their own HTML rendering.

- Application clients

Application clients are like traditional desktop stand-alone applications. Java application clients are packaged in JAR files and execute in their own Java virtual machines. Application clients may be installed explicitly on client machines or provisioned on demand using Java Web Start. Just like Java components, Java application clients run inside J2EE containers and rely on them for system services. Non-Java application clients make use of the J2EE Web and EJB tiers.

J2EE business logic tier

The J2EE business logic tier consists of two sub-tiers, the Web tier which handles requests from and responses to the presentation tier clients, and the Enterprise JavaBeans (EJB) tier, which hosts the application specific business logic components.

- Web tier

The Web tier hosts two J2EE components (Servlets and JavaServer Pages) for processing requests from the presentation tier clients and generating responses that are sent back to clients. Servlets and JSP are hosted by the Web container. It provides the network services for handling the requests and responses. Web containers provide access to J2EE services and communications APIs.

- EJB tier

The EJB tier hosts the application business objects that are typically implemented as Entity and Session Beans, data access, and value objects. The EJB container provides system level services, such as transaction management, concurrency control, and security.

J2EE data resources tier

The data resource tier in J2EE is the *Enterprise Information Systems (EIS)* tier. This tier handles integration with existing enterprise systems and data sources. EIS tier relies on EJB container services, a number of J2EE APIs, and the J2EE Connector Architecture to provide connectivity to enterprise systems and support business-to-business (B2B) transactions.

Figure 2-4 shows the J2EE multi-tier architecture.

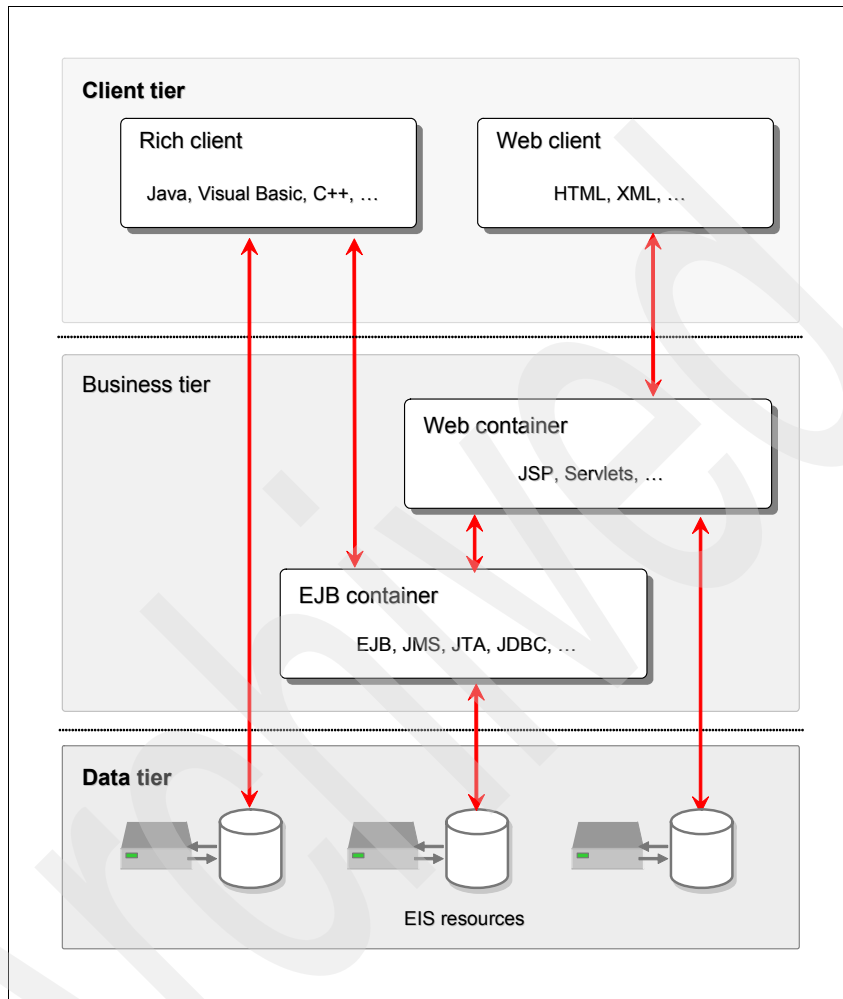


Figure 2-4 J2EE multi tier architecture

2.1.3 J2EE component model

Java components are reusable application level software modules that you can develop and assemble to create sophisticated applications. There are four basic types of components in the J2EE environment.

JavaBeans

JavaBeans are Java classes written to conform to the JavaBeans specification. They are actually part of the Java 2 Standard Edition platform (J2SE™). JavaBeans components can be used by any application or component that understands the JavaBeans format.

Note: J2EE is built on Java 2 Standard Edition platform (J2SE) and it relies on J2SE for a number of its services, of which JavaBeans support is one.

Web components

These components typically execute in business logic tier and may respond to HTTP requests. There are four types of Web components, including Servlets, JSP pages, filters, and Web event listeners:

- ▶ Servlets are Java classes.

These used to extend the capabilities of servers. They are commonly used to extend the applications hosted by Web servers. Servlets support dynamic Web page content, provide database access, and serve multiple clients at any one time.

- ▶ JavaServer Pages (JSP).

JSPs are a server-side scripting technology that enable Java code to be dynamically embedded within Web pages (HTML files) and executed when the page is served, in order to return dynamic content to clients. JSPs are compiled at runtime into servlets that execute to generate the resulting dynamic HTML. Subsequent calls to the same JSP simply execute the compiled servlet.

- ▶ Filters

Filters are objects that can transform a request or modify a response. Unlike Servlets, they do not create responses. They preprocess requests before they reach Servlet, and postprocess responses leaving Servlets.

Enterprise JavaBeans components

An Enterprise JavaBeans is a server-side component that encapsulates the business logic. In the Model-View-Controller (MVC) design pattern EJBs represent the model. There are three types of EJBs:

► Session EJB

Performs tasks on behalf of a client. Within a session EJB, there are two implementation types:

– Stateful

Acts on behalf of a single client and maintains client-specific session information across multiple method calls and transactions. It exists for the duration of a single client/server session. Because the client interacts (talks) with its bean, this state is often called the *conversational state*. The state is retained for the duration of the client-bean session. If the client removes the bean or terminates, the session ends and the state disappears.

– Stateless

Are pooled by their container to handle multiple requests from multiple clients. During a client invocation of a stateless bean's method, the bean's instance variable may contain a state, but only for the duration of the invocation. When the method is finished, the state is no longer retained. All instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client.

► Entity EJB

An entity EJB, once created (either explicitly or implicitly), can be subsequently found with a key for use by a client application. It persists until the remove operation is invoked by a client application or until the data is explicitly removed from the underlying store. There are two types of entity EJB, depending on how persistence is managed:

– Container-managed persistence (CMP)

Container managed persistence handles all database access required by the entity bean. The EJB's code contains no database access (SQL) calls. As a result, the EJB's code is not tied to a specific database. Because of this flexibility, even if you redeploy the same entity EJB on different J2EE servers that use different databases, you do not have to modify or recompile the EJB's code.

– Bean-managed persistence (BMP)

With bean-managed persistence, the developer handles all storage specific access required by the entity bean. This allows the developer to use non-relational storage options.

- ▶ **Message-driven bean (MDB)**

Message-driven EJBs receive and process JMS messages. MDBs have no interfaces. They can be accessed only through messaging and they do not maintain any conversational state. MDBs interact with asynchronous clients through listener queues, and provides separation between message processing and business logic.

Client components

These are code that execute in the client Java virtual machine. There are two types of client components:

- ▶ **Application clients**

Application clients are stand-alone GUI desktop applications written in the Java programming language that offer the native operating system user experience. Application clients have access to all J2EE middle tier facilities.

- ▶ **Applets**

Applets are GUI components that run in a Java enabled Web browser. Applets embedded in Web pages are used to provide a user interface for J2EE applications.

Figure 2-5 shows the n-tier architecture components.

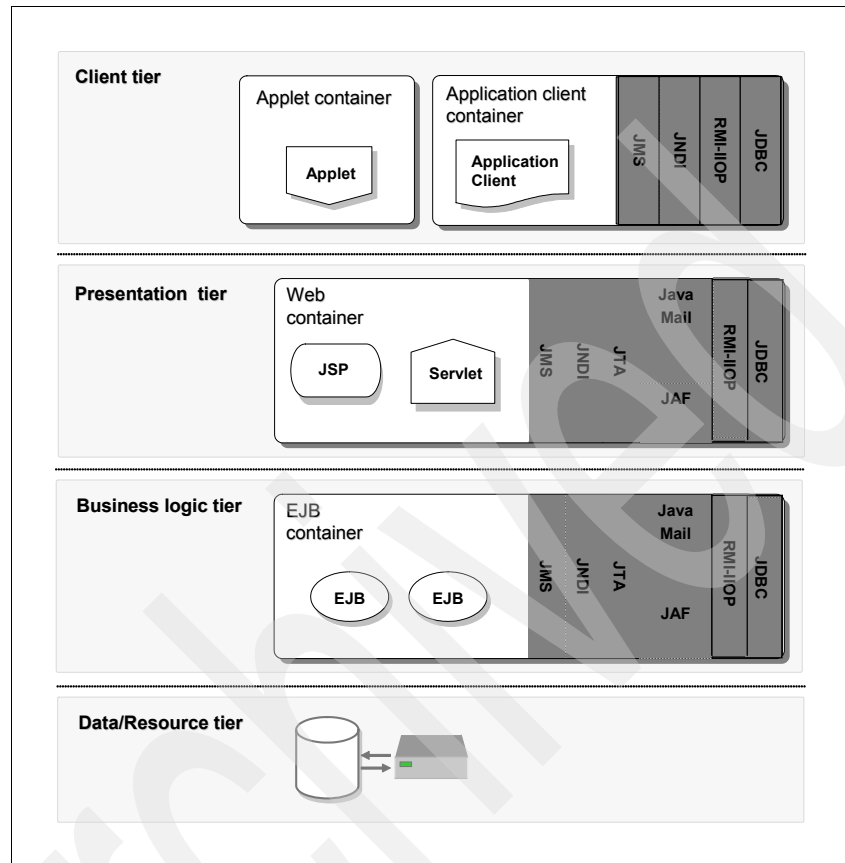


Figure 2-5 Components in the n-tier architecture

2.1.4 J2EE application environment

The Java 2 Enterprise Edition platform is the Java platform for developing and running distributed multi-tier architected Java applications. Java platform is the virtual machine runtime environment known as the Java Virtual Machine (JVM), which acts as an abstraction layer between the host operating system and the Java application, enabling platform independence. Like any physical machine, the virtual machine has an instruction set and provides services to Java applications, such as memory management and input and output operations.

Java programs are compiled into machine independent byte code. The Java Virtual Machine interprets this byte code at runtime. Optimizations are provided through just in time compilation. This works by identifying frequently executed

code segments and then repeatedly optimizing them for execution. Operating system specific optimizations also exist, allowing the Java Virtual Machine to scale.

Java programming language

The JVM bytecode interpreter is language neutral; however, in practice, Java platforms advocate the use of the Java programming language. J2EE applications are almost exclusively developed using the Java programming language. Java is an object-oriented programming language suitable for developing a wide range of applications from programs that run in the browser and applications that run on consumer devices to enterprise scale distributed applications. The advantage of developing your J2EE application entirely in Java is that you can easily move the application to any other operating system environment so long as a JVM exists for the environment.

Note: Java applications can interact with applications written for the native platform (for example, a C-style function within a Dynamic Link Library (DLL) on Windows) through the Java Native Interface (JNI). The Java Native Interface allow you to call unmanaged native code from within the J2EE application and vice-versa.

Application packaging

J2EE applications are packaged as Enterprise Archive (EAR) files. An EAR file is composed of a set of modules. Each module includes a deployment descriptor that specifies how to assemble and deploy the module in a runtime environment. Customized information can be provided at both assembly time and deployment time without having to recompile the application objects.

Figure 2-6 shows the J2EE packaging files.

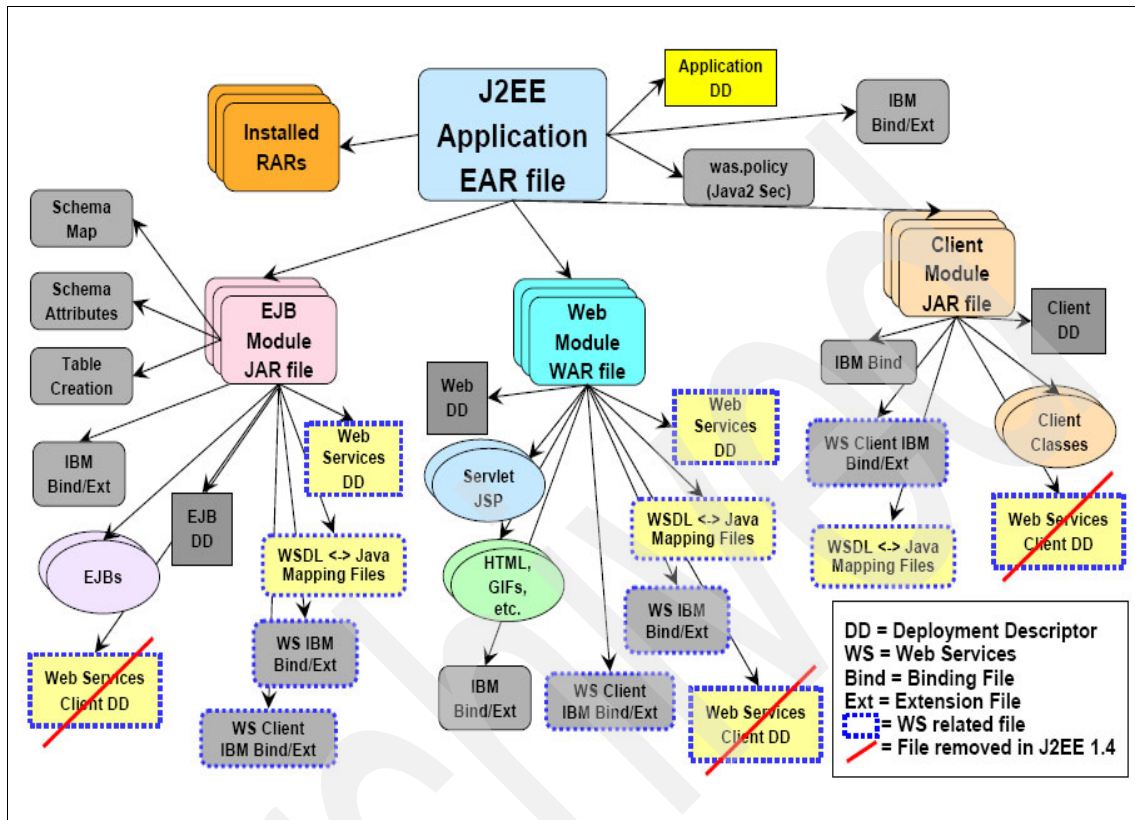


Figure 2-6 WebSphere J2EE packaging files

An EAR file is composed of any number of the following components:

- EJB modules

An EJB module contained in an EJB JAR file contains one or more EJBs.

- Web modules

A Web module is a single Web application in a Web archive (WAR) file. An EAR file can contain multiple Web applications; each Web application in an EAR file must have a unique deployment context.

- Application client modules

An application client module contains a single, stand-alone Java application that is intended to run within an application client container. It includes a specialized deployment descriptor and is composed similarly to an EJB JAR file. The JAR file contains the classes required to run the stand-alone client

and any libraries needed to access JDBC, JMS, JAXP, JAAS, or an EJB client.

- ▶ Resource adapter modules

Resource adapter modules are packaged in Resource Adapter Archive (RAR) files. An RAR file contains Java classes and native libraries required to implement a Java Connector Architecture (JCA) resource adapter to an enterprise information system. Resource adapters do not execute within a container. Rather, they are designed to execute as a bridge between an Application Server and an external Enterprise Information System (EIS).

- ▶ JAR files

Optionally, additional JAR files, so-called utility JARs, containing dependent classes or other components required by multiple modules.

Attention: The IBM WebSphere Application Server V6.0 provides an extension of the J2EE EAR configuration information for resources typically required by J2EE applications. It is not mandatory to supply the additional information at packaging time. The Enhanced EAR Editor can be used to edit several WebSphere Application Server V6 configurations settings and published with the EAR at the time of deployment.

Enhanced EAR resources supported include:

- ▶ JDBC Providers
- ▶ DataSources
- ▶ Substitution variables
- ▶ Class loader policies
- ▶ Shared libraries
- ▶ JAAS authentication aliases
- ▶ Virtual hosts

Containers

The Java 2 Enterprise Edition application server facilitates development of distributed applications by providing a number of containers that provide services to the applications deployed within them. The type of services provided to the application by the container include security, transactions, object pooling, and naming services. The availability of such services enables the application developer to focus on implementing business and presentation logic.

The following containers are defined by the J2EE specification:

► Web container

In addition to managing the Web applications and components that are deployed within it, the Web container is responsible for providing the following services to Web applications:

- On startup of the application server, the container establishes a transport mechanism between the Web server and the Web container. This communication channel is used by the container to receive requests and send responses on behalf of the Web applications running within it.
- The container manages a configurable pool of worker threads for the processing of servlet and Java Server Page requests.
- Session management is also provided by the Web container. It is responsible for creating and invalidating sessions, providing a session tracking mechanism, and writing and reading session data on behalf of the Web application.
- The Web container also provides a virtual hosting mechanism, allowing a single application server to masquerade as multiple Web sites.

► EJB container

The EJB container is responsible for managing the life cycle of the Enterprise Java Beans deployed within it and providing threading and transaction support to them. The container also provides a caching mechanism to optimize access to EJBs.

► Client container

The client container is installed separately on the client and manages the runtime environment for fat client applications. J2EE application client containers provide standard services to the client application, including remote object discovery and remote method invocation, thus allowing the application client to access business logic on server side components, such as EJBs, and also providing access to other resources.

Figure 2-7 on page 47 shows J2EE containers and components.

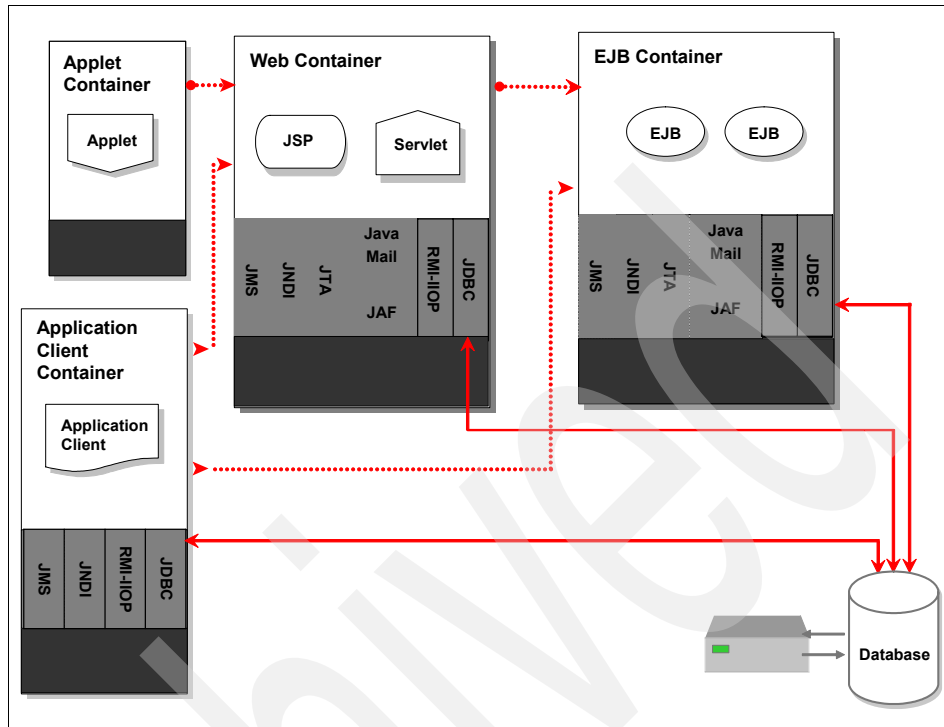


Figure 2-7 J2EE containers and components

Life cycle management

In Java 2 Enterprise Edition, there are two levels of object life cycle management. The Java Virtual Machine provides life cycle management of Java objects, while life cycle management of J2EE application components is a function of the container; see “Containers” on page 45.

In Java, it is the responsibility of the Java Virtual Machine to dynamically load, link, and initialize classes and interfaces. The task of loading a class is delegated either to the bootstrap classloader of the JVM or a user defined classloader. To load a class or interface, the classloader will first attempt to locate a binary representation of the class or interface by traversing the classpath and then create an object instance from the binary representation. Linking is the process of taking the class or interface and combining it into the runtime state of the Java virtual machine so that it can be executed. Upon successful instantiation, the class or interface is finally initialized by executing its `init()` method.

During the life cycle of an object, the JVM maintains a table of references to that object.

Object destruction is also the responsibility of the Java Virtual Machine. This is achieved by means of a garbage collection process that runs periodically to clean up any dereferenced objects. The algorithm used in garbage collection is dependent upon the implementation of the JVM. Generally, though, garbage collection is implemented as a mark and sweep algorithm. In this algorithm, the garbage collection process starts by locating all dereferenced objects in the memory heap and marks them for deletion. In the second phase, the garbage collector removes, or sweeps, the marked objects from the memory heap. JVM technology has improved significantly, resulting in improved performance of the garbage collection process. These enhancements include optimizations such as memory heap defragmentation, short and long lived object heaps, and multi-threaded garbage collection.

The life cycle management of J2EE components such as servlets and EJBs is the responsibility of the container in which they reside.

Object pooling

Object pooling is a mechanism employed in many object-oriented languages that enables the reuse of objects. The advantages of using a pooling mechanism is that it avoids the overhead of continuous object creation and destruction, and improves memory usage. Fewer objects have to be created since they are shared among clients.

Java 2 Enterprise Edition application servers incorporate object pooling mechanisms to improve application performance. In WebSphere Application Server, object pools, such as JDBC connection pools, are created by the application server and made available as resources to J2EE applications. It is the responsibility of the application server to manage the object pool.

2.1.5 Security in J2EE

Almost every enterprise has security requirements and specific mechanisms and infrastructure to meet them. While the implementation and levels of service provided by enterprise security systems may vary, they all address the following considerations to some extent:

- ▶ **Authentication**

This mechanism addresses how entities that are attempting to communicate prove to one another that they are who they say they are.

- ▶ **Access control for resources**

This is the process of ensuring that access to protected application resources is restricted to only those users or groups of users who have authority to access them.

- ▶ Data integrity
Data integrity considerations address how to validate that data passed between two entities has not been modified in some way by a third party while in transit.
- ▶ Confidentiality or data privacy
Mechanisms relating to confidentiality or data privacy deal with how to ensure that data can only be read by those users who are authorized to do so.
- ▶ Non-repudiation
Non-repudiation is a way of providing absolute proof that a particular user performed some action.
- ▶ Auditing
Auditing is the process of creating a tamper proof trail of security-related events in order to evaluate security policies and mechanisms, and when used in conjunction with non-repudiation, to provide evidence of malicious user actions.

The J2EE specification defines a number of goals for security within J2EE applications:

- ▶ Portability
The J2EE security model must support the concept of portability. In other words, it must be implemented in such a manner that J2EE applications are decoupled from the underlying security implementation.
- ▶ Transparency
J2EE application developers wishing to implement security in their components should not need to understand security in order to do so. However, in practice, a developer should at least have an understanding of the security considerations addressed above.
- ▶ Isolation
This is related to the portability requirement. What the specification says here is that authentication and access control should be performed in accordance with instructions provided in the deployment descriptors, and managed by the systems administrator. This ensures that the application is decoupled from the underlying security implementation.
- ▶ Extensibility
The J2EE specification provides security application programmer interfaces. Provided that the application restricts itself to using these APIs for implementing security in its components, it will retain independence from the underlying platform, and thus retain portability.

► Flexibility

The application should not impose a specific security policy; rather, it should facilitate the implementation of security policies.

► Abstraction

The mapping of security roles and access requirements to environment specific security roles, users, and policies should be specified in the applications deployment descriptors. The deployment descriptors should also document which security properties can be modified by the deployer.

► Independence

Required security behaviors and deployment contracts should be implementable using a variety of popular security technologies.

► Compatibility testing

The J2EE security model should be described in such a way that an implementation can readily be certified as compatible or not.

► Secure interoperability

Application components executing in a J2EE product must be able to invoke services provided in a different J2EE product, irrespective of whether the same security policy is used.

Figure 2-8 on page 51 illustrates how WebSphere Application Server leverages the security provided by the operating system and other Java and J2EE components.

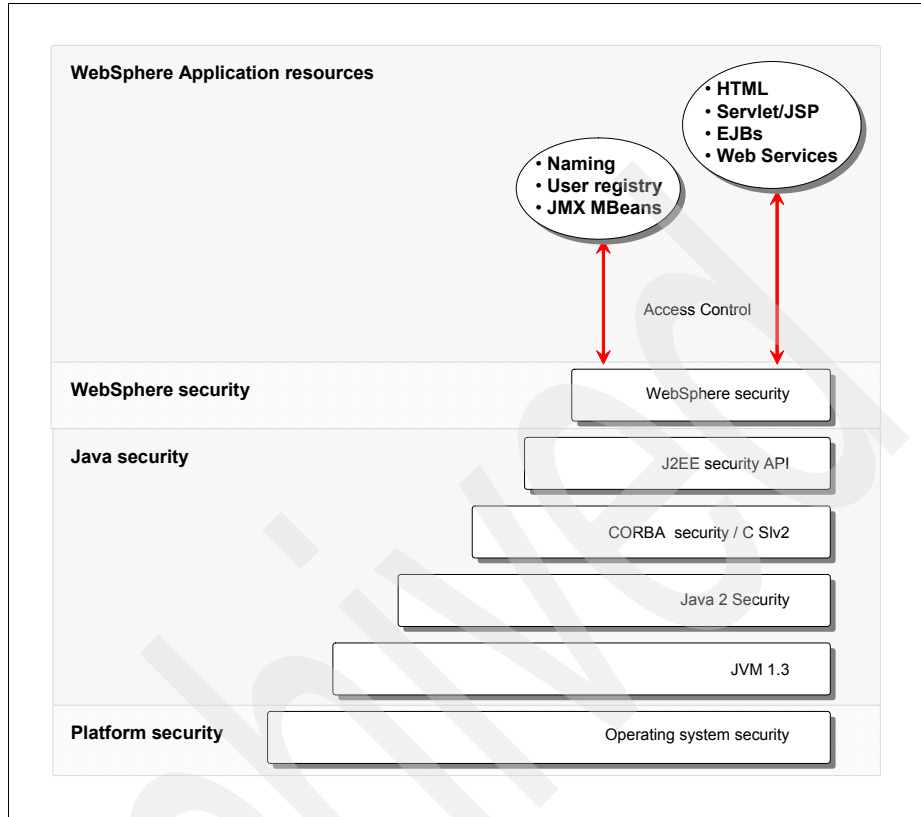


Figure 2-8 WebSphere environment security layers

Due to the layered nature of the WebSphere Application Server security model, the security considerations described earlier in this section are automatically addressed by services provided in the operating system and the J2EE security model.

When developing a J2EE application for WebSphere Application Server, it is important that decisions relating to security be made early in the application development life cycle, preferably during the design phase.

In J2EE applications, security is configured through the deployment descriptors, and it is the responsibility of the container to enforce the security policy.

WebSphere Application Server provides implementations of user authentication and authorization mechanisms, providing support for various user registries:

- ▶ Local operating system User registry
- ▶ LDAP User registry
- ▶ Custom User registry

Authentication mechanisms supported by WebSphere are:

- ▶ Simple WebSphere Authentication Mechanism (SWAM)
- ▶ Lightweight Third Party Authentication (LTPA)

2.1.6 Technologies supporting J2EE architecture

The J2EE specification include technologies that were designed to support system level services and simplify application development.

Component technologies

Components share the need for runtime functions, such as life cycle management, security, and threading. In the J2EE platform, containers provide the support and management of these and other runtime services. The services a component require are declared at installment time when the component is deployed in the container. The component developer need only specify the abstract names of resources and at deployment time, and the deployer customizes the component to the environment, mapping the abstract names to concrete resources. The assembly process requires that you specify container settings for each component you are deploying. The component container settings customize the support to be provided by the J2EE server platform.

Service technologies

These technologies enable uniform access to databases, transactions, naming and directory services, and enterprise information systems.

- ▶ Java Database Connectivity (JDBC)
Provide uniform connectivity and access to a wide variety of tabular data sources.
- ▶ Java Transaction API (JTA)
Allow implementation independent access to transactions. JTA specifies standard Java interfaces for distributed transaction systems between the transactional application, J2EE server, and the transaction manager, which controls access to the shared resources.

► Java Naming and Directory Interface™ (JNDI)

Provide methods for performing standard directory and naming operations. It provides APIs for storing and retrieving any type of named Java object, associating attributes with the objects, and searching using associated attributes.

► J2EE Connector Architecture (JCA)

Provides the Java technology solution for connecting application servers to enterprise information systems (EIS). By providing a standard J2EE Connector architecture resource adapter that plugs into your J2EE application server, an EIS vendor enables connectivity between the application server, the EIS product, and your J2EE application.

Communication technologies

Provides the mechanisms for communication between clients and servers, and between collaborating objects hosted by different servers. The J2EE specification requires support for the following types of communication technologies:

► Internet protocols

Includes Transport Control Protocol over Internet Protocol (TCP/IP), Hypertext Transfer Protocol (HTTP), and Secure Socket Layer (SSL)

► Remote Method Invocation (RMI) protocols

Remote Method Invocation uses a common mechanism for invoking methods on a remote object, stubs, and skeletons. The stub is located on the client or local system and acts as a proxy to the remote object. The client makes a method call on the stub, which then handles the complexities of the remote method call. This mechanism makes the underlying communications transparent to the client. As far as the client is concerned, the method call appears local.

When a method on the stub is invoked, the following actions occur:

- a. The stub establishes a connection to the Java Virtual Machine where the remote object resides.
- b. Method parameters are written and transmitted to the remote Java Virtual Machine. This process is more frequently known as *marshalling*.
- c. The stub then waits for the return value from the method invocation.
- d. When received, the return value is read or unmarshalled.

- ▶ Object Management Group (OMG) protocols

Includes the Common Object Request Broker Architecture (CORBA) technologies, Java Interface Definition Language (IDL), and RMI over Internet Inter-ORB Protocol (IIOP).

IIOP is the underlying communication protocol for Remote Method Invocation. The use of the IIOP protocol in J2EE facilitates existing application and platform integration by allowing objects written in other CORBA-enabled languages to communicate with Java-based applications.
- ▶ Messaging technologies

Includes Java Message Service (JMS), JavaMail™, and JavaBeans Activation Framework (JAF).
- ▶ Data formats

Includes HTML, image files in two formats, GIF and JPEG, JAR files, class files (compiled Java files), and XML.

2.2 The WebSphere platform

IBM software brands provide you with an integrated platform for designing, building, running, and managing innovative e-business solutions. The five IBM software brands are as follows:

- ▶ Rational®

This is the Rational software brand. It includes products for all aspects of software design and construction such as requirements and analysis, software configuration management, and software quality. The Rational brand includes the Rational Application Developer for WebSphere Software—the comprehensive Integrated Development Environment (IDE) for designing, developing, analyzing, testing, profiling, and deploying applications using Java, J2EE, Web, Web Services, Service-Oriented Architecture, and portal technologies.
- ▶ WebSphere

This is a family of products that provides the robust application and transaction infrastructure that is the foundation for the IBM software platform. The real-time, distributed infrastructure enables the development and deployment of integrated solutions that range from simple Web applications to enterprise scale business-to-business, e-commerce, and transactional applications.

► DB2®

This is the family of database servers that supports information management. Integration is at the center of the IBM software solution platform. This is evident in the way IBM uses DB2 for building information integration middleware and information management software that includes warehousing, data mining, content management, and media asset management.

► Tivoli®

The efficient, secure, and healthy functioning of your IBM infrastructure is supported by Tivoli branded products. Your IT infrastructure is equally important, as the applications that support your business processes are critical to meeting the IT services needs that business increasingly demands. Tivoli products provide a range of infrastructure management services that include availability, configuration, service level, application, workload, and security management.

► Lotus®

Lotus offers solutions that focus on collaboration and human productivity. Lotus products are geared towards effective communication, working more productively, teaming, leveraging of collective knowledge, and learning. Lotus Workplace™ and WebSphere portal products give you the flexibility to meet the information needs of your workers, business partners, suppliers, and clients by delivering the appropriate information when and how it is needed.

Figure 2-9 shows the IBM software platform.

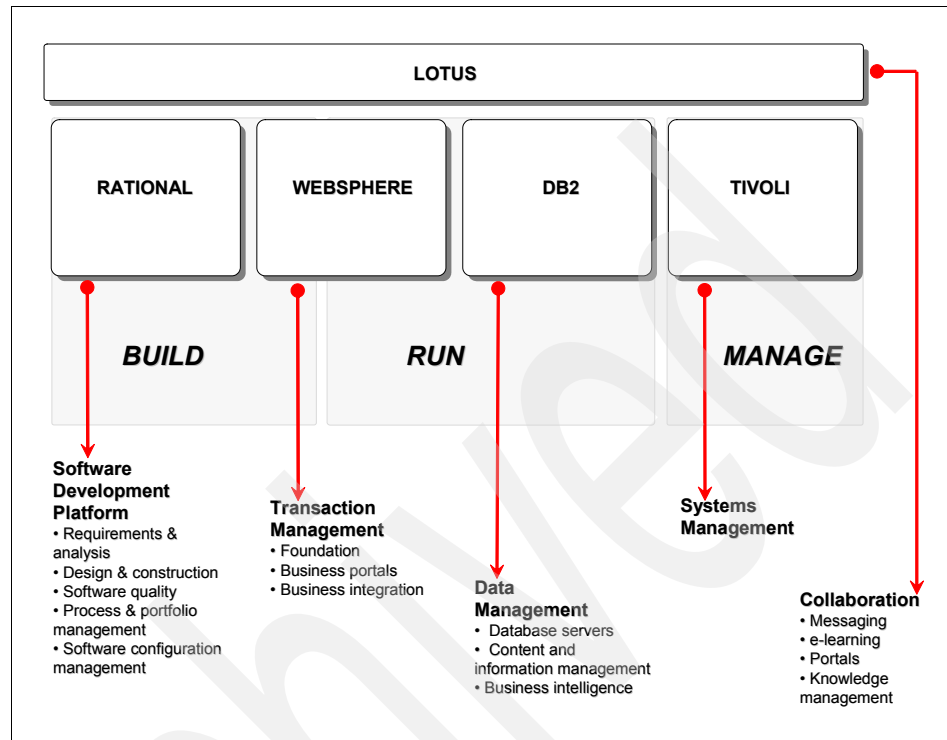


Figure 2-9 IBM software platform

2.2.1 The WebSphere family

IBM WebSphere is a family of products that form the foundation for a comprehensive e-business platform that deliver runtime and integration capabilities for building and deploying applications ranging from simple Web publishing application to enterprise scale transactional application.

The WebSphere family consists of three categories of tools.

Foundation and tools

The foundation of the WebSphere platform is the WebSphere application server and the set of tools for building applications that run in the environment.

► Application servers

The WebSphere Application Servers are Java-based J2EE runtime environments that feature high performance and extremely scalable transactional engines capable of handling high-volume secure transactions and Web Services. WebSphere Application Server include:

– WebSphere Application Server

Provides an integrated foundation platform for messaging flows, Web Services, and J2EE applications.

– WebSphere Application Server Network Deployment

This configuration builds on the base WebSphere application server to provide services that include clustering, high availability, Web, and edge-of-network services.

– WebSphere Extended Deployment

Delivers add-on features that provide a dynamic high-performance environment for WebSphere applications.

► Edge servers are tightly integrated with the WebSphere family

They distribute processing to the edge of the network, providing caching, content distribution, and load balancing functionality.

► WebSphere Host Integration server

Leverages and extends existing assets for new e-business solutions without making changes to the existing applications. Solutions include an emulation client that can be accessed over the Web, and a communications server that offers secure and reliable connections to host applications over a variety of networks and protocols.

Business integration

The WebSphere business integration is a set of products that include tools and process templates that will enable you to connect applications and share information within your company and externally with business partners and suppliers. The WebSphere business integration products include:

► Application connectivity, which includes:

– WebSphere Business Integration Message Broker

Transforms and enriches in-flight data.

– WebSphere Business Integration Adapters

Allows for the creation of processes that exchange data between systems.

– WebSphere Data Interchange

Provides data translation and transaction management.

- WebSphere MQ
Enables connectivity between applications running on different systems.
- WebSphere MQ Everywhere
Extends the reach of applications to the mobile environment.
- ▶ Process integration, which includes:
 - WebSphere Business integration Connect
Enables the connection of a large number of trading partners.
 - WebSphere Business Integration Event Broker
Enables handling of high message volumes and large numbers of endpoints.
 - WebSphere Business Integration Express for Item Synchronization
Provides item synchronization for suppliers through registries.
 - WebSphere Business Integration Monitor
Displays real-time data from events created by IBM WebSphere MQ Workflow.
 - WebSphere Business Integration Server
Enables small and medium businesses to quickly and easily integrate their applications.
 - WebSphere Business Integration Workbench
Provides a process modeling tool enabling testing and validation of business processes.
 - WebSphere Business Integration Workbench Server
Provides repository management and Web publishing capabilities in a centralized data warehouse.
 - WebSphere InterChange Server
Offers mission-critical reliability, availability, and centralized system and configuration management.
 - WebSphere MQ Workflow
Aligns and integrates organizational resources and capabilities with business strategies.
- ▶ Other business integration, which includes:
 - WebSphere Global Data Synchronization for WebSphere Product Center
Provides electronic global data synchronization solution between trading partners.

- WebSphere Business Integration Modeler
Provides tools to design, test, and communicate complex business processes.
- WebSphere Business Integration Server Express Plus
Enables small and medium businesses to quickly and easily integrate their applications.
- WebSphere Business Integration Server Foundation
Delivers an integration platform to build and deploy composite applications.
- WebSphere Product Center
This is a product information management solution that helps assemble an accurate, consistent central repository, linking products, location, trading partners, organizations, and terms of trade information, which is typically scattered throughout the enterprise.
- WebSphere Studio Application Developer Integration Edition
Provides the integrated development environment for building and deploying composite J2EE applications that deploy to the IBM WebSphere Business Integration Server Foundation.

Business portals

These are the Web equivalent of your desktop. They deliver applications specifically tailored to each employee's role on all kinds of devices. Portals also provide multiple access channels to your clients, partners, and suppliers, allowing them to interact with your applications, content, and processes. They personalize and optimize the user experience and make it easier to do business with you. At the center of the portal component architecture are *portlets*. Portlets are complete applications that conform to the portlet API and follow the model-view-controller design pattern. They provide access to all types of resources, including applications, Web-based content, Web pages, Web Services, and syndicated content feeds. You can create your own portlets or select from a catalog of portlets created by IBM and by IBM Business Partners. Personalized Web pages are created by arranging portlets on portal pages tailored to individuals, teams, divisions, and organizations.

WebSphere Portal is available in a number of packages:

- WebSphere Portal for multiplatforms
This provides powerful collaboration capabilities, such as instant messaging, team workplaces, people finder, and e-meetings.

WebSphere Portal for Multiplatforms includes two offerings:

- Portal Enable

This is the base offering and provides personalization, content, and document management, search, and workflow functions along with the scalable portal framework.

- Portal Extend

This adds powerful collaborative, extended search, and Web analysis features to enhance portal effectiveness.

- ▶ WebSphere Portal Express for multiplatform

This combines the features of the WebSphere Portal family with simplified installation and the option for user or processor-based licensing. This combination enables small businesses as well as departments within larger companies to more easily deploy sophisticated employee, business partner, and client portals. WebSphere Portal Express for Multiplatforms includes two offerings:

- WebSphere Portal - Express

This is the base offering and contains the portal framework, a document manager, a selection of portlets, the portal toolkit, and WebSphere Application Server.

- WebSphere Portal - Express Plus

This adds in features for team collaboration, including instant messaging and virtual teamrooms.

- ▶ WebSphere Commerce

This is a set of integrated software components for building, maintaining, and managing stores on the Web. WebSphere Commerce automates and integrates cross-channel marketing and sales processes to enable companies to conduct business with their clients when they want, how they want, and where they want. WebSphere Commerce delivers the following:

- Rich, contextual client and user experience across channels.
- Hundreds of seamless, cross-channel processes. It also accelerates the implementation of new ones.
- Multiple business models in a single solution.

WebSphere Commerce includes three offerings:

- WebSphere Commerce Business Edition

This provides a powerful, flexible infrastructure based on a unified platform for running large, high-volume B2B and advanced B2C e-commerce Web sites for global e-businesses.

- WebSphere Commerce Professional Edition

This increases site functionality for B2B businesses and B2C retailers by enhancing client buying experiences, improving operational efficiencies, and accommodating high transaction volumes.

- WebSphere Commerce - Express

This provides the core capabilities growing firms need to get started or expand their e-commerce site at a minimum investment—all on a rock solid platform that can support more advanced functionality as their needs grow.

- ▶ WebSphere Everyplace® mobile portal

This extends the capabilities of the IBM WebSphere portal to mobile devices. It offers capabilities for mobile portal navigation that delivers compelling and differentiating user experience and flexible navigation across wireless devices.

2.2.2 The WebSphere Application Server family

The IBM WebSphere application server is a family of server products. Each member of the family has essentially the same architectural structure.

- ▶ The WebSphere Application Server and WebSphere Application Server Express: You are limited to stand-alone application servers. Each stand-alone application server provides a fully functional J2EE 1.4 environment.
- ▶ The WebSphere Application Server Network Deployment (WAS ND): Offers advance configurations and central management of multiple application servers to support high availability, scalability, and performance to suit the complex needs of enterprise systems.
- ▶ The WebSphere Extended Deployment: Includes features and functions that extend the capabilities of WebSphere Application Server Network Deployment (WAS ND) to deliver on demand responsiveness, simplified administration, and high performance enhancements, such as resource virtualization and pooling using node groups and dynamic clusters.

Table 2-1 shows the WebSphere Application Server family.

Table 2-1 WebSphere Application Server family feature summary

	Express and Base	Network/Extended Deployment
Application server	Yes	Yes
Application server clustering	No	Yes
External Web server	Yes	Yes
External generic server	No	Yes
WebSphere JMS servers	No	Yes

2.2.3 Stand-alone server configuration

Express, Base, and Network Deployment all support a single stand-alone server environment. With a stand-alone configuration, each application server acts as a unique entity. An application server runs one or more J2EE applications and provides the services required to run those applications. Multiple stand-alone application servers can exist on a machine, either through independent installations of the WebSphere Application Server code or through multiple configuration profiles within one installation. However, WebSphere Application Server does not provide for common management or administration for multiple application servers. Stand-alone application servers do not provide workload management or failover capabilities.

Figure 2-10 on page 63 shows an architectural overview of a stand-alone application server.

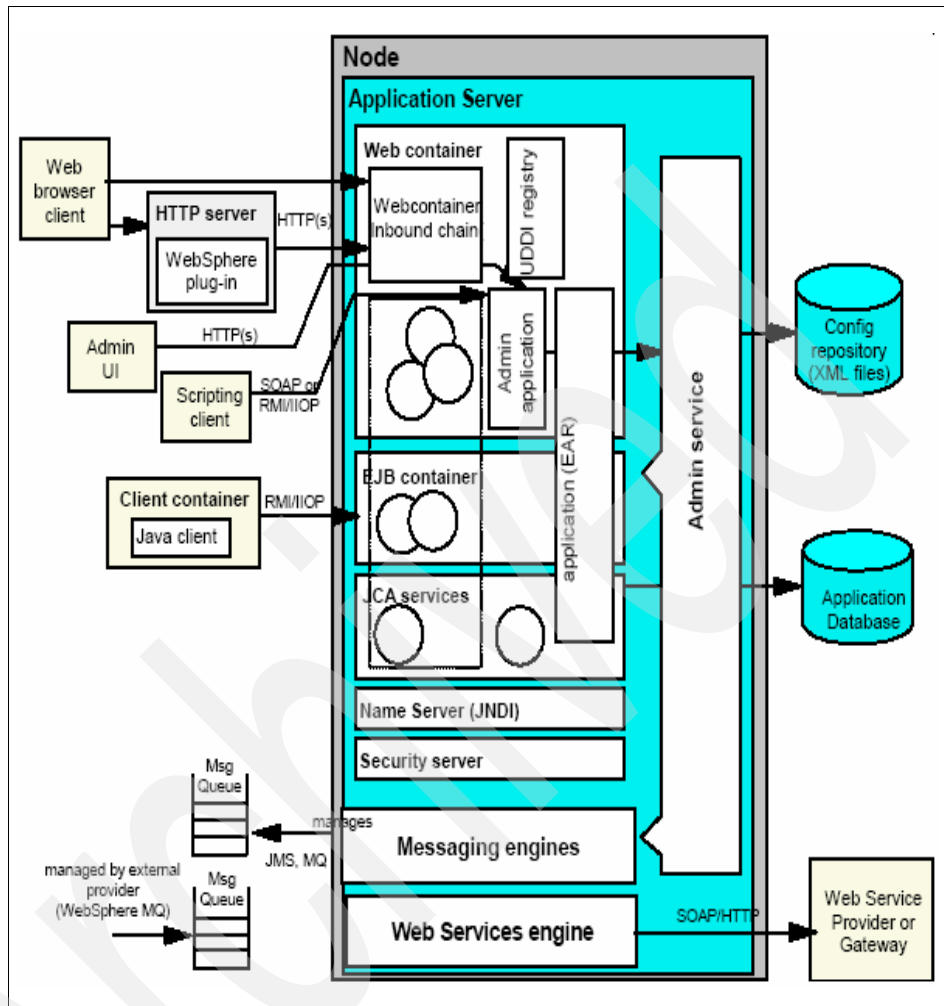


Figure 2-10 Architectural overview of stand-alone WebSphere Application Server

2.2.4 Distributed server configuration

With WebSphere Application Server Network Deployment and Extended Deployment, you can build a distributed server configuration, which enables central administration, workload, and failover management. In this environment, you integrate one or more application servers into a cell that is managed by a deployment manager. The application servers can reside on the same machine as the deployment manager or on multiple separate machines. Administration and management is handled centrally from the administration interfaces via the deployment manager.

With this configuration, you can create multiple application servers to run unique sets of applications and then manage those applications from a central location. However, more importantly, you can cluster application servers to allow for workload management and failover capabilities. Applications that you install in the cluster are replicated across the application servers. When one server fails, another server in the cluster continues processing. Workload is distributed among Web containers and Enterprise JavaBeans containers in a cluster using a weighted round-robin scheme.

Figure 2-11 illustrates the basic components of an application server in a distributed server environment.

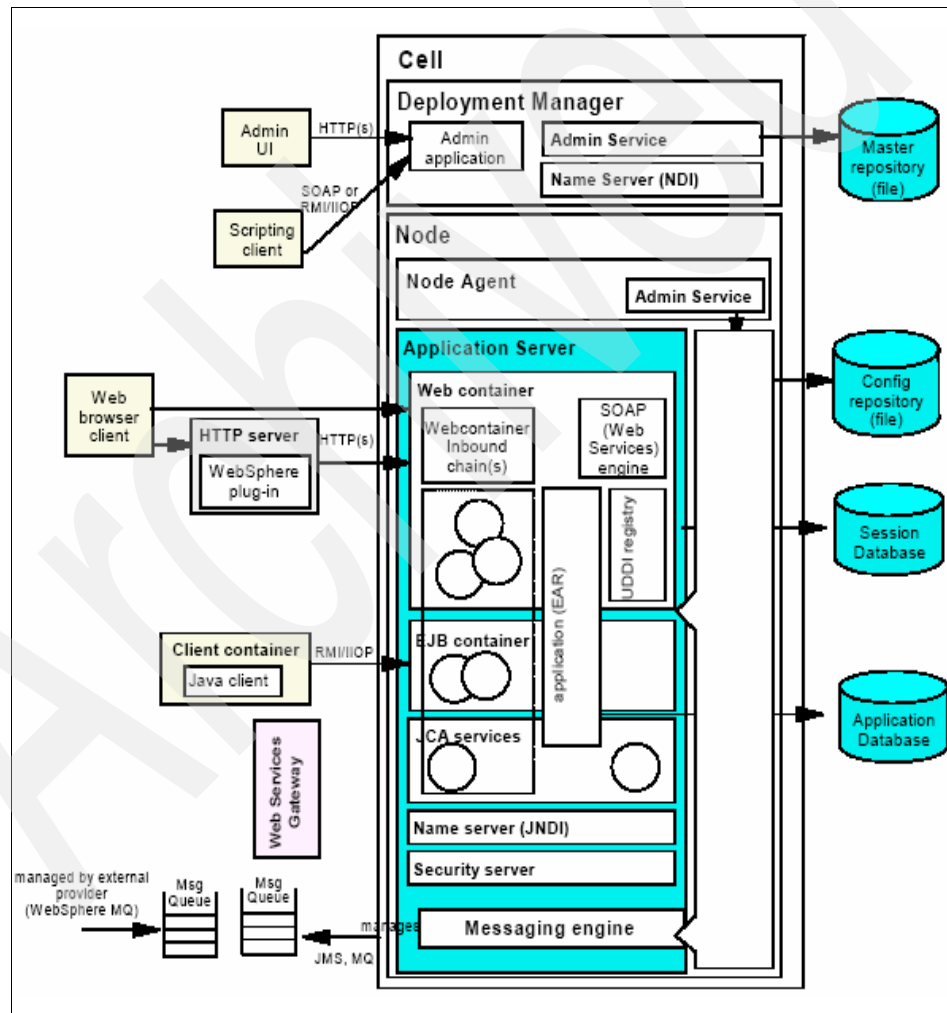


Figure 2-11 WebSphere Application Server distributed server environment

Application servers, nodes, and cells

Regardless of the configuration, the WebSphere Application Server is organized based on the concept of cells, nodes, and servers. While all of these elements are present in each configuration, cells and nodes do not play an important role until you take advantage of the features provided with Network Deployment.

Application servers

The application server is the primary runtime component in all configurations and is where an application actually executes. All WebSphere Application Server configurations can have one or more application servers. In the Express and Base configurations, each application server functions as a separate entity. There is no workload distribution or common administration among application servers. With WebSphere Application Server Network Deployment, you can build a distributed server environment consisting of multiple application servers maintained from a central administration point. In a distributed server environment, you can cluster application servers for workload distribution.

Nodes, node groups, and node agents

A node is a logical grouping of server processes that are managed by WebSphere and that share common configuration and operational control. A node is associated with one physical installation of WebSphere Application Server. In a stand-alone application server configuration, there is only one node.

With WebSphere Application Server Network Deployment, you can configure multiple nodes that you can manage from one common administration server. In these centralized management configurations, each node has a node agent that works with a deployment manager to manage administration processes.

A node group is a new concept introduced with WebSphere Application Server V6. A node group is a grouping of nodes within a cell that have similar capabilities. A node group validates that the node is capable of performing certain functions before allowing those functions. For example, a cluster cannot contain both Windows nodes and nodes that are not Windows based. In this case, you can define multiple node groups, one for the Windows nodes and one for nodes other than Windows. A DefaultNodeGroup is automatically created based on the deployment manager platform. This node group contains the deployment manager and any new nodes with the same platform type.

The Node Agent communicates directly with the Deployment Manager, and is used for configuration synchronization and administrative tasks, such as stopping/starting of individual application servers and performance monitoring on the application server node.

Cells

A cell is a grouping of nodes into a single administrative domain. In the Base and Express configurations, a cell contains one node. That node may have multiple servers, but the configuration files for each server are stored and maintained individually.

In a distributed server configuration, a cell can consist of multiple nodes, which are all administered from a single point. The configuration and application files for all nodes in the cell are centralized into a cell master configuration repository. This centralized repository is managed by the deployment manager process and synchronized with local copies that are held on each of the nodes.

2.3 IBM Rational Software Development Platform

The IBM Rational Software Development Platform is a combination of a comprehensive set of tools, proven best practices, and professional services designed to enable business driven development and support the four imperatives for software development. It provides a development platform with the right tools for rapid, flexible, and incremental application development across disparate systems and heterogeneous environments. It is built on an open and modular framework that enables plug-ins that support a variety of application types and user development paradigms. These paradigms include service-oriented, component-based, and pervasive and embedded architectures. The IBM Rational Software Development Platform is:

- ▶ **Proven**

It consists of tools and best practices that are the choice of thousands of high-performance teams.

- ▶ **Complete**

It drives business value throughout the software life cycle by supporting every member of the development team, as well as business and operations stakeholders.

- ▶ **Open**

You can leverage existing assets and choose from a wide array of development languages, development platforms, and partner technologies, including Microsoft .NET, BEA, and Oracle solutions.

- ▶ **Modular**

You can choose the exact tools and adoption path that best fit your needs. You can use existing tools with components of the IBM Rational Software Development Platform.

The modular design of SDP is based on Eclipse, which provides different perspectives for different kinds of application development and development roles, thereby enabling development teams to:

- ▶ **Build**
Build new applications and software-based products that create value.
- ▶ **Extend**
Extend the value of existing, commercial-off-the-shelf (COTS) and packaged applications by customizing them to suit specific business needs.
- ▶ **Integrate**
Integrate new, existing, and pre-packaged applications through software development.
- ▶ **Modernize**
Modernize existing business applications by transforming valuable existing resources through a modern programming paradigm.
- ▶ **Deploy**
Deploy new applications and upgrades securely and rapidly with the flexibility of controlled distribution, configuration, and management.

Figure 2-12 shows the software development with the IBM Rational Software Development Platform.

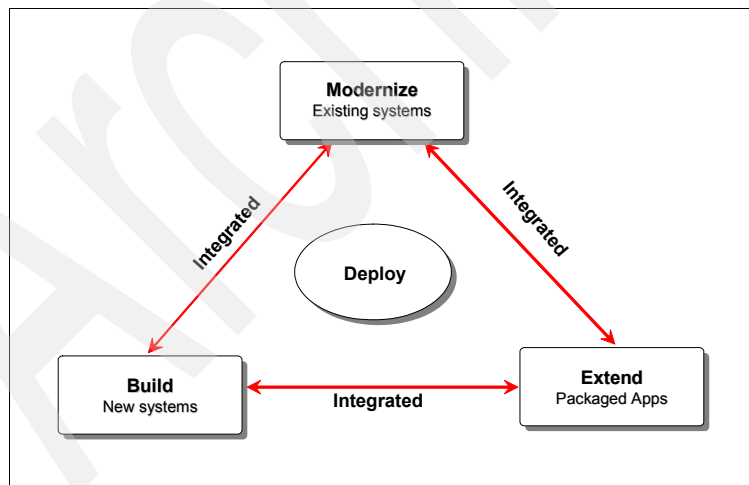


Figure 2-12 Software development with the IBM Rational Software Development Platform

2.3.1 IBM Rational Software Development Platform technologies

The technical underpinnings of the IBM Rational Software Development Platform's open, customizable, extensible, and integrated framework are a set of technologies infrastructure that consists of the following elements.

Eclipse

This is the open source development project that includes a generic framework for tool integration, and a Java development environment built using the framework. Its open and extensible plug-in architecture enables the core framework to be extended by tools that support various kinds of application development approaches, resulting in the Eclipse Integrated Development Environment (IDE), a workbench with broad industry support.

Eclipse technology is at the center of the IBM Rational Software Development Platform strategy. It provides a powerful and yet flexible tool integration infrastructure on which IBM has created its next-generation software tooling platform. The Eclipse platform performs three primary functions in the IBM Rational Software Development Platform:

- ▶ It provides the UI framework for a visually consistent rich client experience as you move between activities within the IBM Rational Software Development Platform.
- ▶ It supports the sharing of information across different activities through use of a common set of models expressed in the Eclipse Modeling Framework (EMF) technology.
- ▶ Its integration infrastructure enabled the creation of teaming capabilities available throughout the IBM Rational Software Development Platform.

The Eclipse Modeling Framework (EMF)

This is the Eclipse project framework that aids in the creation of software models and generation of code based on the class models. It brings together UML, XML, and Java, whereby you can create an EMF model starting from either UML, XML, or using annotated Java. The integrated nature of EMF makes it possible to output UML diagrams, XML schema, or Java interfaces from an EMF model, or to interchange the model between tools and applications. EMF incorporates Java Emitter Templates (JET), the Eclipse tool for code generation. JET can generate Java, SQL, or JSP code. Whenever the model changes, EMF regenerates the code. Similarly, changes to the code update the model.

The open Model-Driven Development platform (MDD)

Model-Driven Development is all about software development that is centered around the meta-data definition of systems, in other words, using models to design, construct, deploy, and modify systems. The foundation for MDD is based on the Object Management Group's concept of Model Driven Architecture (MDA). MDA prescribes separation of the specification of a system from how the system is realized on a given platform. MDA calls for the following:

- ▶ Specifying a system independently of the platform that will support it
- ▶ Specifying platforms
- ▶ Choosing a particular platform and transforming the system specification into one for that particular platform

IBM leveraged the Eclipse framework in creating the SDP model driven development infrastructure (see Figure 2-13 on page 70). Three primary components are at the center of this infrastructure:

- ▶ Open source technologies provided by the Eclipse project, including UML2 metamodel—realization of UML2.0 specifications in EMF, and Haydes testing framework.
- ▶ IBM value-add capabilities built on top of the set of open source technologies.
- ▶ Core IBM technologies for data sharing, artifact management, team interaction, and information aggregation.

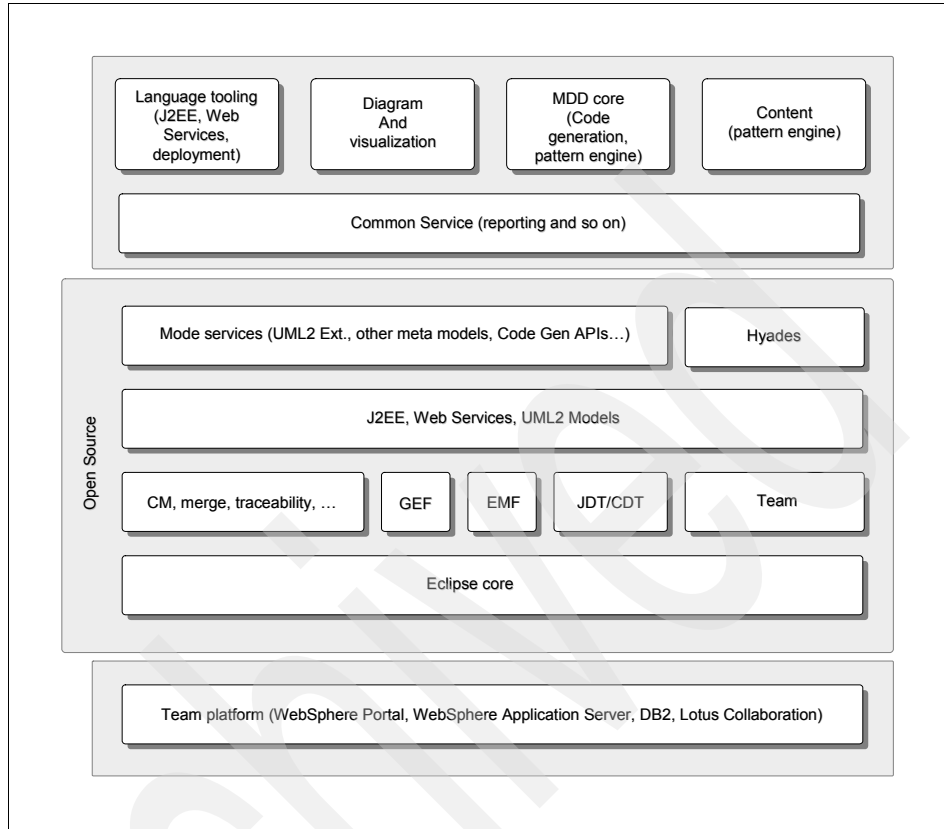


Figure 2-13 IBM MDD platform

The WebSphere programming model

The Service-Oriented Architecture (SOA) strategy and implementation on the IBM WebSphere platform and the entire IBM middleware stack (DB2, Tivoli, and Lotus) strongly influenced the IBM Rational Software Development Platform programming model. Key elements of the programming model include:

- ▶ Service Data Objects (SDOs)

SDOs provide a simplified data access programming model for various resources (data as well as EIS) and complement the core Web Services standards XML, Web Services Definition Language (WSDL), and Simple Object Access Protocol (SOAP).

- ▶ BPEL4WS

The service orchestration and component scripting standard that supports workflow and business process integration.

► **JavaServer Faces**

The Java technology that simplifies building user interfaces for Java server based applications. It enables developers who are not J2EE experts to easily build Web applications by assembling user interface components.

Role-based solutions portfolio

The IBM Rational Software Development Platform leveraged the flexibility of the Eclipse framework to provide user interfaces that enable users to work in an environment that is tailored to their specific roles and the development tasks being performed. The use of a common set of models in the infrastructure makes it easy for the different roles to share artifacts across different activities.

Figure 2-14 shows roles that exist in the IBM Rational Software Development Platform.

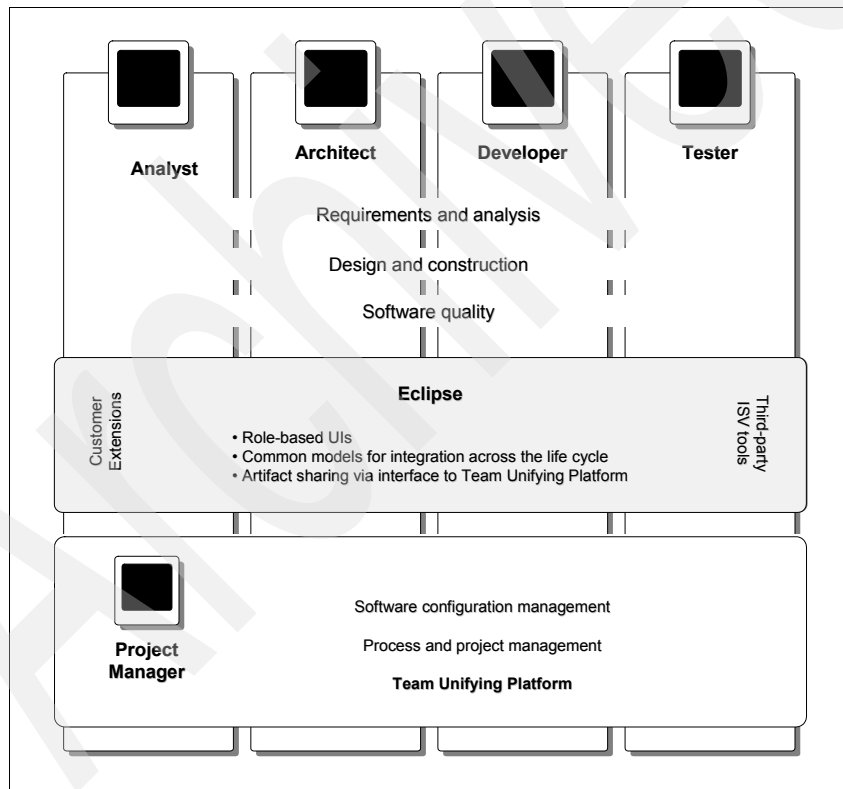


Figure 2-14 Roles in the IBM Rational Software Development Platform

2.3.2 The IBM Rational Software Development Platform products

The IBM Rational Software Development Platform (SDP) includes an expansive portfolio of IBM products covering all aspects of software product development. The tools map to the following major areas of the software development life cycle:

- ▶ **Requirements and analysis**
Integrated tools for requirements management, use case development, business modeling, and data modeling.
- ▶ **Design and construction**
Tools for architecture and design modeling, model-driven development, component testing, and runtime analysis activities.
- ▶ **Software quality**
Tools that address the three dimensions of software quality: functionality, reliability, and performance.
- ▶ **Software configuration management**
Solutions for simplifying and managing change, including version control, software asset management, and defect and change tracking.
- ▶ **Process and portfolio management**
Integrated solutions that help teams manage change and requirements, model and test systems, implement a proven development process, and assess and report progress.

Requirements and analysis

Integrated within the IBM Rational Software Development Platform are requirement and change management features that allow each role in the development team to have a direct window into user needs.

- ▶ **Project managers**
Project managers can view and create project requirements and track requirements back to their original sources as enhancement requests.
- ▶ **Developers**
Developers can review requirements and use cases (directly from within products like IBM Rational Software Architect and Rational Application Developer) while they are developing the software.
- ▶ **Testers**
Testers can get a jump-start on testing activities by viewing project requirements directly from their test management environment.

- ▶ Administrators

Administrators can include requirements when creating project baselines.

Key products

IBM offers a number of products for requirements and analysis. The following are the key IBM products:

- ▶ Rational RequisitePro®

Promotes better communication and enhances teamwork by integrating requirements across IBM Rational Software Development Platform tools, ensuring that all team members are informed of the most current requirements information. It provides detailed traceability views that display parent/child relationships and show requirements that may be affected by upstream or downstream changes. Rational RequisitePro enables detailed attribute customization and filtering to maximize the informative value of each requirement within a project.

- ▶ IBM Rational Rose® Data Modeler

Provides the data analyst with a UML-based visual modeling tool that integrates data modeling with application modeling and system development.

- ▶ IBM Rational Software Modeler

A UML-based visual modeling and design tool that supports model-driven development. It enables architects and designers to produce platform-independent models. Developers use the architecture models and patterns as the basis for implementation, thereby accelerating the development process.

Other IBM requirements and analysis products include the following:

- ▶ WebSphere Business Integration Modeler

This product is used to define, model, analyze, simulate, and report business processes. It also extends IBM WebSphere MQ Workflow with business tools to visualize process impact.

- ▶ WebSphere Business Integration Monitor

Monitors business processes in real time, using visual dashboards for improved business decision-making.

Table 2-2 shows the requirements and analysis key products.

Table 2-2 Requirements and analysis key products

Products	Business Analyst	Systems Analyst	Data Analyst
IBM Rational RequisitePro	x	x	
IBM Rational Rose Data Modeler		x	x
IBM Rational Software Modeler		x	
WebSphere Business Integration Modeler		x	
WebSphere Business Integration Monitor		x	

Design and construction

IBM design and construction tools are an integral part of the IBM Rational Software Development Platform. Built on the Eclipse open framework, these products help developers maximize their productivity. They not only simplify software development, but also transform software construction by introducing model driven development. These products support design and construction activities including architecture, design and modeling, construction, rapid application development (RAD), component testing, and runtime analysis activities.

Figure 2-15 on page 75 shows the IBM Rational developer products.

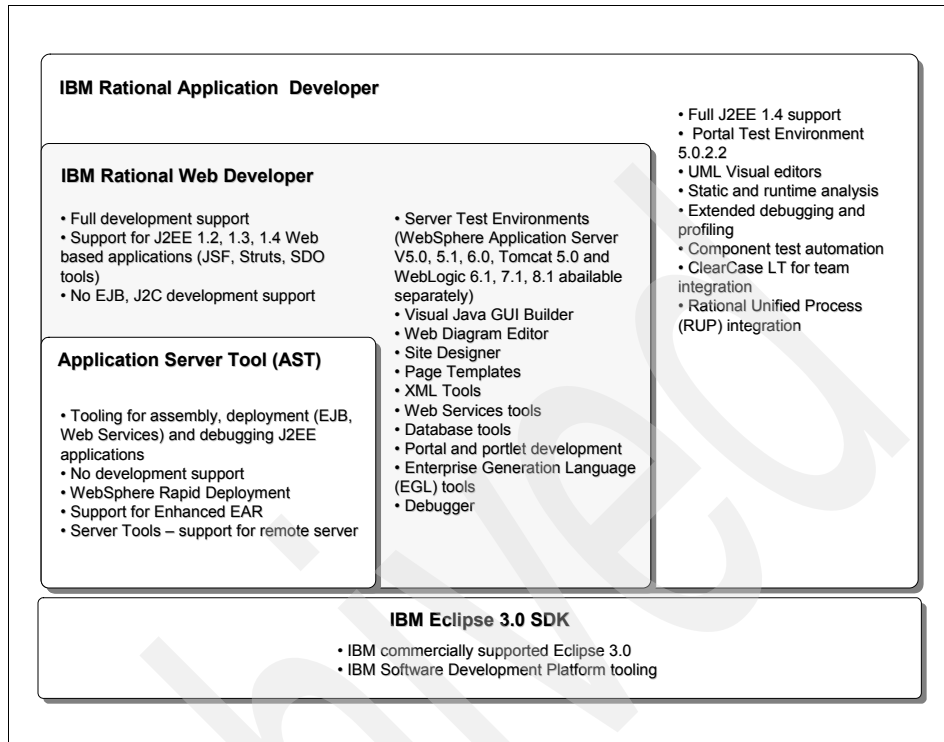


Figure 2-15 IBM Rational developer products

Key products

The products for design and construction range from those used to build enterprise-wide applications to tools for building embedded systems and software products for mobile devices. These products offer capabilities that include choice of development styles, modeling, round-trip engineering, model execution, and existing integration. The following are the key IBM products for software design and construction:

► **IBM Rational Web Developer for WebSphere Software**

This provides the integrated development environment and tools you need to develop Web applications. The range of Web applications can vary from a simple application (consisting of only static Web pages) to an advanced e-commerce application that includes JavaServer Pages (JSP) files and Java servlets.

Web applications in the Rational Web Developer are maintained in projects. Two types of Web projects are supported:

- Dynamic Web projects

These can contain dynamic J2EE resources, such as servlets, JSP files, filters, and associated metadata, in addition to static resources, such as images and HTML files.

- Static Web projects

These only contain static resources.

IBM Rational Web Developer supports all activities involved in planning, creating, and deploying Web sites, including the design, construction, testing, publishing, and maintenance.

- ▶ IBM Rational Application Developer for WebSphere Software

This is a comprehensive environment for constructing a variety of classes applications by individuals or advanced distributed development teams.

IBM Rational Application Developer for WebSphere Software supports all the features of Web application development in IBM Rational Web Developer for WebSphere Software. Additionally, it provides support for developing the following types of applications:

- Portal projects

These projects are used for creating applications that customize both the structural and stylistic framework of portals.

- Portlet projects

These projects help you develop portlet applications that conform to the support for the IBM portlet API and the JSR 168 portlet API for the IBM WebSphere Portal.

- Java projects

These projects contain files necessary for building Java applications. Associated with each project is a Java builder that can incrementally compile Java source files as they are changed. Here are some of the things you can do with Java projects:

- Develop J2EE applications.
- Use the visual editor for Java to create your application's graphical user interface (GUI).
- Visually develop Java applications using UML diagrams. You can use UML to develop artifacts for Java applications or for round-trip engineering.

- You can make system-wide changes by refactoring your code without affecting the behavior of the program.

– Data access

This includes tools that allow you to use wizards and views to define and work with databases. Data access projects allow you to create and work with the following:

- Tables, views, and filters
- SQL statements
- DB2 UDB routines (such as DB2 UDB stored procedures and user-defined functions)
- SQLJ, SQL DDL, DADX, and XML files

You can also visually develop database applications using Unified Modeling Language (UML) class, Information Engineering (IE) notation, or Integration Definition for Information Modeling (IDEF1X) notation. You can use these diagrams to do the following:

- Automatically abstract your application's structural information from the code so you can gain understanding of database applications.
- Develop and redesign database applications for better maintainability.
- Explore and navigate data elements and relationships in your applications.

– EJB project

This is the environment that you use to develop and test enterprise beans and to organize and manage the resources contained in an EJB module. You must first create an EJB project, after which you can create session, entity, or message-driven beans. You can add enterprise beans to the EJB project in one of the following ways:

- Create new enterprise beans.
- Define new beans from existing Java classes.
- Import enterprise beans from EJB JAR files.
- Generate beans by bottom-up mapping from relational database tables.

You can visually develop EJB applications using Unified Modeling Language (UML) diagrams. UML diagrams depict components and elements in your application. Working with UML models, you are able to:

- Analyze and understand your application to identify and understand the relationships to enterprise beans.
- Create representations of your application in another form.

- Automatically abstract your application's structural information from code to a higher level of abstraction.
- Redesign your application for maintainability.
- Produce your application's structural information without access to the design from which it was originally developed.
- Modify the target system or develop and generate new systems.
- Understand and develop behaviors and interactions of enterprise beans.

– Web Services projects

These support the following aspects of Web Services development:

- Discover
Browse the UDDI Business Registries or WSIL documents to locate existing Web Services for integration.
- Create or transform
Create bottom-up Web Services from existing artifacts, such as JavaBeans, enterprise beans, URLs that take and return data, DB2 XML Extender calls, DB2 Stored Procedures, and SQL queries. Create top-down Web Services from WSDL discovered from others or created using the WSDL Editor.
- Build
Build by wrapping existing artifacts as SOAP accessible services and describing them in WSDL. The Web Services wizards assist in generating Java client proxy to Web Services described in WSDL and in generating JavaBean skeletons from WSDL.
- Deploy
Deploy Web Services into the WebSphere Application Server or Tomcat test environments.
- Secure
Secure Web Services running on WebSphere Application Server.
- Test
Test Web Services running locally or remotely in order to get instant feedback.
- Develop
Develop by generating sample applications to assist you in creating your own Web service client application.

- Publish

Publish Web Services to UDDI V2 or V3 Business Registry, advertising your Web Services so that other businesses and clients can access them.

- EGL

Enterprise Generation Language (EGL) is the development environment and programming language that lets you write fully functioning Web applications, where the details of Java and J2EE are hidden. You can generate Java source code from your EGL application. EGL outputs are executable objects that can be deployed on different platforms.

- XML

Provides a comprehensive visual Extensible Markup Language (XML) development environment with the following editors:

- XML editor

This is used for creating and viewing Extensible Markup Language (XML) files. You can use it to create new XML files from scratch, existing DTDs, or existing XML schemas. You can also use it to edit XML files, associate them with DTDs or schemas, and for validation.

- DTD editor

This is used for creating and viewing Document Type Definitions (DTDs). Using the DTD editor, you can create DTDs and generate XML schema files.

- XML schema editor

This is used for creating, viewing, and validating XML schemas. You can use the XML schema editor to perform tasks such as creating XML schema components, importing and viewing XML schemas, generating relational table definitions from XML schemas, and generating JavaBeans for creating XML instances of an XML schema.

- XSL editor

This can be used to create new Extensible Stylesheet Language (XSL) files or to edit existing ones. You can use content assist and various wizards to help you create or edit the XSL file. Once you have finished editing your file, you can also validate it. Also, you can associate an XML instance file with the XSL source file you are editing and use that to provide guided editing when defining constructions such as an XPath expression.

- XPath expression wizard

This can be used to create XML Path Language (XPath) expressions. XPath expressions can be used to search through XML documents, extracting information from the nodes (such as an element or attribute).

- XML to XML mapping editor

This is used to map one or more source XML files to a single target XML file. You can add XPath expressions, groupings, Java methods, or conversion functions to your mapping. Mappings can also be edited, deleted, or persisted for later use. After defining the mappings, you can generate an XSLT script. The generated script can then be used to combine and transform any XML files that conform to the source DTDs.

- RDB to XML mapping editor

This is used for defining the mapping between one or more relational tables and an XML file. After you have created the mapping, you can generate a document access definition (DAD) script, which can be run by the DB2 XML Extender to either compose XML files from existing DB2 data, or decompose XML files into DB2 data.

► IBM Rational Software Architect

This builds on the capabilities of the IBM Rational Application Developer for WebSphere Software to provide an integrated design and development tool that leverages model-driven development. IBM Rational Software Architect enables architects and designers to produce language-independent Unified Modeling Language (UML) models of software architecture and reusable assets. It offers architects and developers a rich set of model-driven development and runtime analysis capabilities that include:

– Design

Design application software by building modeling projects. You model the design of applications by capturing application requirements and modeling application behavior in use case models, model the user workflow using activity diagrams, capture the application domain in an analysis model, and capture the architecture in a design model.

– Transformation

Transformation of the source file to generate a target form of output such as the transition from model to code. You can generate Enterprise JavaBeans (EJB) and Java classes from annotated UML model elements by running the EJB transformations, or generate Java classes from UML models.

- Discovery

Detects patterns in application code automatically. By reviewing the topic diagrams, you are able to gain an understanding of how an application is designed.

- Authoring

You are able to integrate software design solutions into Unified Modeling Language 2.0 (UML 2.0) models. By capturing frequently used or complex structures and processes as design patterns, you eliminate redesign and instead promote reuse and consistency.

Other IBM design and construction products include the following:

- ▶ IBM Rational Rose Technical Developer

This supports the most advanced modeling constructs, such as model execution and fully executable code generation, resulting in the highest levels of productivity.

- ▶ IBM WebSphere Studio Enterprise Developer

This simplifies the skills needed to develop component-based Web applications; helps developers rapidly create well structured business systems that integrate WebSphere software and traditional transactional environments, and promotes the reuse and transformation of existing applications to help reduce costs and shorten the development cycle.

- ▶ IBM WebSphere Studio Asset Analyzer

This maintains and extends existing assets through impact analysis and enhanced application understanding.

Table 2-3 shows the design and construction key products.

Table 2-3 Design and construction key products

Products	Software Architect	App Developer	Web Developer	Traditional Developer
IBM Rational Web Developer for WebSphere Software			x	
IBM Rational Application Developer for WebSphere Software		x	x	
IBM Rational Software Architect	x	x	x	
IBM WebSphere Studio Enterprise Developer				x
IBM WebSphere Studio Asset Analyzer	x			x
IBM Rational Rose Technical Developer	x	x		

Software quality

The IBM Rational Software Development Platform include products for ensuring quality in the development and validation of software products.

Key products

The key products for software quality include the following:

► IBM Rational TestManager

The central console for test activity management, execution, and reporting. It is an extensible platform that supports various test approaches from pure manual test approaches to various automated paradigms, including unit testing, functional regression testing, and performance testing. Rational TestManager is accessible to all members of a project team, ensuring the high visibility of test coverage information, defect trends, and application readiness.

► IBM Rational Manual Tester

This is an authoring and execution tool. Working with IBM Rational Manual Tester, you can create scripts that contain testing instructions to be performed step-by-step by a tester. IBM Rational Manual Tester also supports the following:

- It promotes test step reuse to reduce the impact of software change on testers.
- Imports preexisting Microsoft Word and Excel-based manual tests.
- Exports test results to CSV-based files for analysis in preferred third-party tools.
- Content sharing across distributed test sites.
- Data entry and verification during test execution to reduce human error.

► IBM Rational Functional Tester

Automated functional and regression testing tool. It supports testing of the functionality and Graphical User Interface (GUI) of Java, Microsoft Visual Studio® .NET, and Web-based applications. Its features include:

- Automated capabilities for data-driven testing
- Choice of scripting language and editors for test authoring and customization:
 - Java in Eclipse
 - Microsoft Visual Basic .NET in Visual Studio .NET

Included in the packaging for IBM Rational Functional Tester are two other products:

- IBM Rational ClearCase® LT
This provides version control, automated workspace configuration, and parallel development support of development assets.
- ScriptAssure™
This uses pattern-matching capabilities to enable Functional Tester to attain test script resiliency in the face of frequent application user interface changes.

► IBM Rational Performance Tester

This is a performance test creation, execution, and analysis tool used for validating the scalability and reliability of complex multi-tier applications before deployment. Here are some of IBM Rational Performance Tester capabilities:

- It detects variable data and prepares tests for data-driven load test generation using built-in data correlation filters.

- It supports creation of advanced analysis algorithms through script customization and an extensive library of test functions.
- Its customizable test workload scheduler permits highly accurate models of true user profiles.
- You can generate real-time reports showing up-to-the second response times across users and user groups, exposing system bottlenecks.
- You can correlate detailed server resource data with response times to expose hardware-related performance limitations.

Other IBM software quality products include the following:

► IBM Rational Robot

This has long been the industry leader for automated functional testing, including regression and smoke testing. It is the only tool to provide full native support for functional testing of .NET applications. Rational Robot ships with IBM Rational TestManager for automated management of all testing activities, including manual testing.

► IBM Rational Team Unifying Platform™

This integrates all the testing activities for one application with centralized test management, defect tracking, and version control. It provides an integrated suite of infrastructure tools and processes, including IBM Rational RequisitePro, IBM Rational ProjectConsole™, IBM Rational ClearCase LT, IBM Rational ClearQuest®, IBM Rational TestManager, IBM Rational SoDA®, and IBM Rational Unified Process®.

Table 2-4 shows software quality key products.

Table 2-4 Software quality key products

Products	Business Analyst	Tester	Developer
IBM Rational Manual Tester	x	x	x
IBM Rational Functional Tester		x	x
IBM Rational Performance Tester		x	x
IBM Rational Robot		x	
IBM Rational TestManager	x	x	x

Software configuration management

Software configuration management (SCM) solutions are a core foundation of the IBM Rational enterprise change management portfolio. SCM enables development teams to capture, control, and securely manage software changes and assets.

IBM Rational SCM products provide process-centric change management across the application development life cycle. Integrated with leading IDEs, including Rational Application Developer and Microsoft Visual Studio .NET, IBM SCM products automatic and streamlined control of software assets across geographically distributed development teams and help increase productivity.

Key products

The IBM Rational Software Development Platform software configuration management key products include:

- ▶ IBM Rational ClearCase Change Management Solution
An integrated software asset management, workflow management, and defect and change tracking system that enforces development process for improved responsiveness and efficiency.
- ▶ IBM Rational ClearCase Change Management Solution Enterprise Edition
This provides integrated software configuration management for medium to large development projects and geographically distributed teams.
- ▶ IBM Rational ClearCase
This provides software asset management for medium to large projects.
- ▶ IBM Rational ClearCase LT
This provides entry-level version control for small, local projects.
- ▶ IBM Rational ClearQuest
This provides flexible defect and change tracking across the project life cycle.
- ▶ IBM Rational ClearQuest MultiSite
This is an option to Rational ClearQuest to support defect and change tracking across geographically distributed projects.
- ▶ IBM Rational ClearCase MultiSite®
This is an option to Rational ClearCase to support geographically distributed software asset management for medium to large projects.
- ▶ IBM Rational ClearCase and MultiSite
This provides a software asset management solution for geographically distributed projects.

- ▶ IBM Rational ClearQuest and MultiSite
This is a defect and change tracking solution for geographically distributed projects.
- ▶ IBM Rational ClearCase Change Management Solution
An integrated software asset management, workflow management, and defect and change tracking system that enforces development process for improved responsiveness and efficiency.
- ▶ IBM Rational Team Unifying Platform
This integrates all the testing activities for one application with centralized test management, defect tracking, and version control.

Table 2-5 shows software configuration and management key products.

Table 2-5 Software configuration and management key products

Products	Project Manager	Developer	Tester	Analyst
IBM Rational ClearCase	x	x	x	
IBM Rational ClearQuest	x	x	x	x
IBM Rational Team Unifying Platform	x	x	x	x

Process and portfolio management

In organizations that recognize software development as a business process or have adopted business driven development, senior IT leaders and project managers need to drive consistency, predictability, and performance to meet business needs. To achieve this, there must be clear definition of responsibilities and a shared understanding of the processes and goals among members of the development teams. IBM Rational Software Development Platform provides integrated tools that enable development teams to work together more effectively, gain visibility across project portfolios, and make faster and better decisions. The tools range from a software development process platform based on proven best practices, to tools for prioritizing, planning, managing, and measuring progress of IT projects.

Key products

The IBM Rational Software Development Platform process and portfolio management key products include:

- ▶ IBM Rational Portfolio Manager

This provides comprehensive IT governance and executive level visibility into IT projects.

- ▶ The IBM Rational Team Unifying Platform

This provides the infrastructure tools, processes, and integrations development teams need to work together more effectively. It unifies teams by providing common access to development assets, communication alerts, and processes to maintain project momentum and focus. The IBM Rational Team Unifying Platform is an integrated suite of infrastructure tools and processes that include:

- IBM Rational RequisitePro

This is a requirements management solution.

- IBM Rational ProjectConsole

This provides managers and team members with access to complete project information through a single Web site.

- IBM Rational ClearCase LT

This provides version control for small, local projects.

- IBM Rational ClearQuest

This enables flexible defect and change tracking across the project life cycle.

- IBM Rational TestManager

This enables management of all testing activities for an application, including manual, regression, functional, performance, and runtime analysis.

- IBM Rational SoDA

This automates the creation and maintenance of comprehensive project documentation and reports.

- IBM Rational Unified Process (RUP®)

This unifies the team by providing common access to development assets, communication alerts, and workflow processes.

► IBM Rational Unified Process (RUP)

This is a flexible software development process platform that helps deliver customized yet consistent process guidance to the project team. The RUP platform includes tools for configuring RUP for specific project needs, developing internal knowledge into process components, and Web-based deployment, as well as an online community for exchanging best practices with peers and industry leaders.

► The IBM Rational Suite®

This provides a full life cycle solution of analyst, developer, and tester products to unify cross-functional teams and support enterprise software projects from requirements to release. Specifically, Rational Suite unifies cross-functional teams in a Windows environment through key product integrations and workflow; accelerates development through visual modeling, code-generation, and reverse engineering capabilities; finds and eliminates runtime errors, memory leaks, and performance issues; includes best practices, market leading tools, and configurable process; provides all the tools needed in one box; and offers access to product-specific discussion forums, white papers, and re-usable assets as a member of the IBM Rational online development community.

Products included with Rational Suite are:

– IBM Rational Rose XDE™ Developer Plus

This provides a rich set of model-driven development and runtime analysis capabilities for building software applications that address the needs of both J2EE-based and .NET-based systems.

– IBM Rational PurifyPlus™ for Windows

This provides a complete set of runtime analysis tools designed for improving application reliability and performance in a Windows environment.

– IBM Rational Functional Tester for Java and Web

This virtually eliminates script maintenance by creating resilient, reusable test scripts in Java with ScriptAssure.

– IBM Rational Robot

This automates functional, regression, and configuration testing for a wide range of application types, including .NET.

– IBM Rational Team Unifying Platform

This manages development across the life cycle.

– IBM Rational Developer Network®

This provides Rational users with targeted articles, Web-based training, reusable assets, and discussions to help them be successful with software application development projects.

Table 2-6 shows process and portfolio management key products.

Table 2-6 Process and portfolio management key products

Products	Project Manager	Analyst	Tester	Developer
IBM Rational Portfolio Manager	x	x	x	x
IBM Rational Team Unifying Platform	x	x	x	
IBM Rational Unified Process	x	x	x	x
IBM Rational Suite	x	x	x	x

The WebSphere platform is the IBM implementation of the Java 2 platform Enterprise Edition (J2EE) specifications. It implements the complete set of J2EE architecture elements, including application components, containers, resource adapters, databases accessibility, and the set of standard services. J2EE is the Java-based platform created by Sun™ Microsystems™ (together with its industry partners, which include IBM).

In this section, we introduced the IBM WebSphere platform for developing and deploying enterprise scale multi-tiered distributed Web applications.

Introduction to .NET platform

The intent of this chapter is to enable readers with knowledge of J2EE and WebSphere to gain an understanding of the .NET platform. It provides an overview of the .NET framework, its capabilities, and technologies.

This chapter contains the following sections:

- ▶ The .NET initiative
- ▶ The .NET suite
- ▶ Developing .NET applications
- ▶ Deploying and managing .NET applications

3.1 The .NET initiative

Mention .NET and most people will immediately conclude that you are referring to programming with .NET framework. While this may be true most of the time, .NET is also a brand, development tools, and technologies. More strategically, .NET is Microsoft's push for use of Web Services to develop software systems to connect information, people, systems, and devices.

The introduction of .NET framework marked a new beginning for Windows' applications runtime environment. It was also a major evolutionary step in the advancement of Windows development technologies and tools. You can trace this evolutionary advancement to the introduction of Microsoft Windows Distributed interNet Applications (DNA) and beyond. DNA introduced the framework for developing multi-tier, scalable, and distributed applications over the Internet.

3.1.1 Windows DNA

Microsoft introduced DNA architecture in 1999, a framework for building what was then a new class of distributed computing solutions integrating the Internet, client/server, and PC computing. At the center of the framework is the COM+ integrated component programming model. COM+ is the latest evolution of the Microsoft Component Object Model (COM). It incorporates COM, Microsoft Transaction Server (MTS), Microsoft Message Queuing (MSMQ), and new component services to create a runtime environment for scalable and distributed applications.

N-tier architectures with DNA

One of the most important new features introduced by Windows DNA was the strong separation of solutions into three tiers. It solved a number of problems that were inherent with 2-tier architectures and enabled clear separation of the user interface from the application logic and resource management. The three-tier architecture model had the advantage of enabling modular software solutions with well defined interfaces that allow independent upgrade, replacement, or technology change of any of the three tiers.

- **Presentation tier**

The presentation tier or user access layer is the user access point to DNA applications. This layer presents data to the user and accepts user inputs for validation before sending it to the middle tier where the business logic resides. The presentation tier in DNA supports both the traditional stand-alone application client and Web based thin clients.

- ▶ **Business logic tier**

The middle tier or the business layer receives user inputs from the presentation tier, interacts with the data tier to process the user request, and sends back responses to the presentation tier. This tier is where COM+ services and business logic components are deployed. The middle-tier components are not tied to specific clients; rather, they run on server machines that can be clustered to provide load balancing and scalability.

- ▶ **Data resource tier**

In the 3-tier Windows DNA architecture model, the data services layer that can be accessed from the middle tier and, on occasion, from the presentation tier, provides a level of abstraction from persistent data. It manages all interaction with data stored in databases and in file systems.

Technologies supporting the DNA architecture

The first implementation of Windows DNA was made up of a set of products that included:

- ▶ Component Object Model (COM)
- ▶ Windows NT®
- ▶ Microsoft Message Queue (MSMQ), Microsoft Transaction Server (MTS), and Internet Information Server (IIS)
- ▶ Site Server 3.0 Commerce Edition
- ▶ SNA Server
- ▶ SQL Server 6.5
- ▶ Exchange Server 6.5
- ▶ Visual Studio 6.0

And development technologies that included:

- ▶ Access Data Object (ADO)
ADO offered the developer an easy way to make use of several data sources, including databases through standardized API.
- ▶ Active Server Pages (ASP)
Active Server Pages are a server side scripting derivative of the Visual Basic language targeted on Web servers. ASP is comparable to JSP in J2EE.

To get a full picture of the Windows DNA technologies, you will have to consider the Windows DNA 2000 platform starting with the release of the Windows 2000 operating system. With the Windows 2000 release, Microsoft delivered an encompassing technology infrastructure for building and deploying Windows DNA solutions. The Windows DNA 2000 platform delivered three sets of services and technologies designed to make programming DNA solutions easier and scalable. These technologies and services included COM+ and a set of component services, built-in core services from Windows 2000 operating systems, and the Windows DNA 2000 suite of servers.

COM+ services

COM+ incorporates Component Object Model (COM), Distributed COM (DCOM), and the Microsoft Transaction Server (MTS). It also incorporates a number of component services to create a managed services environment making it easier for developers to develop and use components. COM+ manages tasks such as thread allocation, security, and distributed transactions previously handled by developers. Microsoft has since added to the number of available COM+ services; the initial set of COM+ services included the following:

- ▶ **COM+ Events service**

This is a loosely coupled publish/subscribe event system. Publishers store event information in a COM+ catalog and subscribers query the catalog for events of interest. The COM+ Events service simplify programming for both publishers and subscribers by maintaining subscription data in the catalog and handling all event semantics.

- ▶ **COM Just-in-time Activation service**

This service enable your components to hold active references to objects while COM+ deactivates the object so it will not tie up valuable server resources. COM+ reactivates the object just in time when your component calls a method on the object.

- ▶ **Object pooling service**

This service enables the developer to configure your component to be kept active in a pool, ready to be used by clients of the component. You achieve significant performance improvement by configuring your components in this manner.

- ▶ **COM+ Queued Components service**

The queued component service enables your components to invoke and execute other components asynchronously. You make calls to target components without regard as to their availability or accessibility. Queued component service takes care of delivering messages to the target component when it is ready to receive messages.

- ▶ **COM+ Security services**

These services provide security features such as declarative role-based security and authentication that you can use to protect your COM+ applications. It includes services you configure administratively and APIs you can call from within your code.

- ▶ **COM+ Transactions service**

This service provides automated transactions processing framework you can use and rely on when you design your COM+ components.

Windows 2000 native services

Windows 2000 integrated previously separate applications into a core set of services for building distributed Web applications into the operating system. The set of services included COM+, the significance of which we have already pointed out and how crucial the component object model is to the Windows environment. The initial set of services that were part of the Windows DNA 2000 platform included:

- ▶ **Internet Information Server (IIS)**

This Windows 2000 Web Server application is implemented as a set of system services. It supports the common Internet protocols such as HTTP, FTP, Simple Mail Transfer Protocol (SMTP), and Network News Transfer Protocol (NNTP). IIS can serve up static as well as dynamic Web pages. IIS also includes an extension API (the Internet Server API - ISAPI) through which applications running on the Web server could extend its functionality (Active Server Pages (ASP) is an example of an extension built using ISAPI).

- ▶ **Active Server Pages (ASP)**

ASP is a system service that makes use of ISAPI to provide a server side scripting environment. It allows for the embedding of script commands, and COM components into HTML pages to create dynamic and interactive Web applications.

- ▶ **Message Queuing**

Message Queuing is a messaging service infrastructure for creating distributed, loosely-coupled applications. Applications can use the Message Queuing service to communicate across heterogeneous networks and with applications that may be offline. Message Queuing provides security, transaction support, and guaranteed delivery of messages.

► ActiveX® Data Object (ADO)

This service provides an easy-to-use API used for communicating with data sources. It interfaces to OLE DB, which provides access to the underlying data repositories. ADO is one of the three technologies in the Microsoft Data Access Components (MDAC) (the other two being OLE DB and ODBC), which implements the Universal Data Access a key element of Windows DNA.

3.1.2 Evolution of .NET

While Microsoft was promoting DNA as the Internet Application Architecture of the future, changes were occurring in the computing industry. Significant among these were emergence of Web Services and Java 2 Enterprise Edition. So, in order to unify the different programming models and evolve Microsoft Windows for the future platform, Microsoft took steps and evolved the DNA platform and the underlying technologies to .NET. The .NET platform is built on the underlying technologies and products that make up DNA, including the Windows Operating System and COM+ technology.

The key values that .NET provides are as follows:

- It encapsulates the Windows operating system and its Quality of Service mechanisms within industry standard protocols, such as those involved in Web Services and Web applications.
- It provides a runtime environment for application software with services like garbage collection, exception management, and namespace support.
- It provides enhanced programming language independence and a new programming language: C#.
- It provides a Framework of pre-built classes that perform functions many applications need.

As shown in Figure 3-1 on page 97, .NET provides these values through the implementation of a Common Language Runtime and a Framework on top of COM+ and the operating system. The purpose of the Common Language Runtime is to provide language independence and code execution management. The Framework provides access to the underlying Quality of Service mechanisms, such as COM+, and simplifies many tasks, such as building and consuming Web Services.

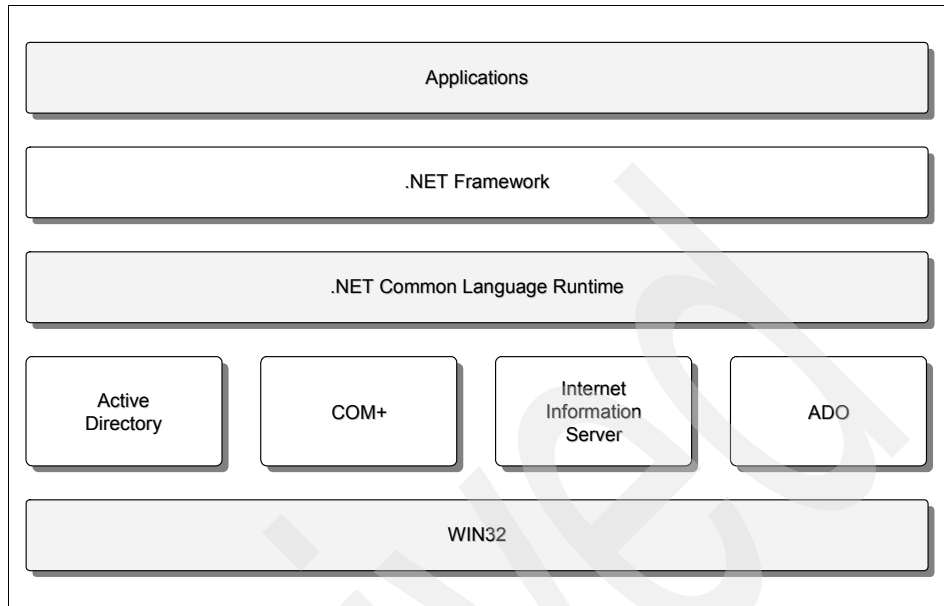


Figure 3-1 Overview of the Microsoft .NET architecture

Various underlying products in Figure 3-1 were rebranded for .NET. They however continue to perform the same functions as in Windows DNA but have been enhanced to provide additional features. For example:

- ▶ A new version of the OS has been released called Windows Server® 2003.
- ▶ COM+ was upgraded from V1.0 to V1.5 and has been rebranded as Enterprise Services under .NET.
- ▶ ADO has been enhanced and rebranded ADO.NET.
- ▶ Internet Information Services was also enhanced to Version 6.0.

N-tier application architecture

Microsoft describes how to architect applications for .NET using published patterns. These patterns are described in a highly consistent manner and are available online at:

<http://www.microsoft.com/resources/practices/default.asp>

See Figure 3-2.

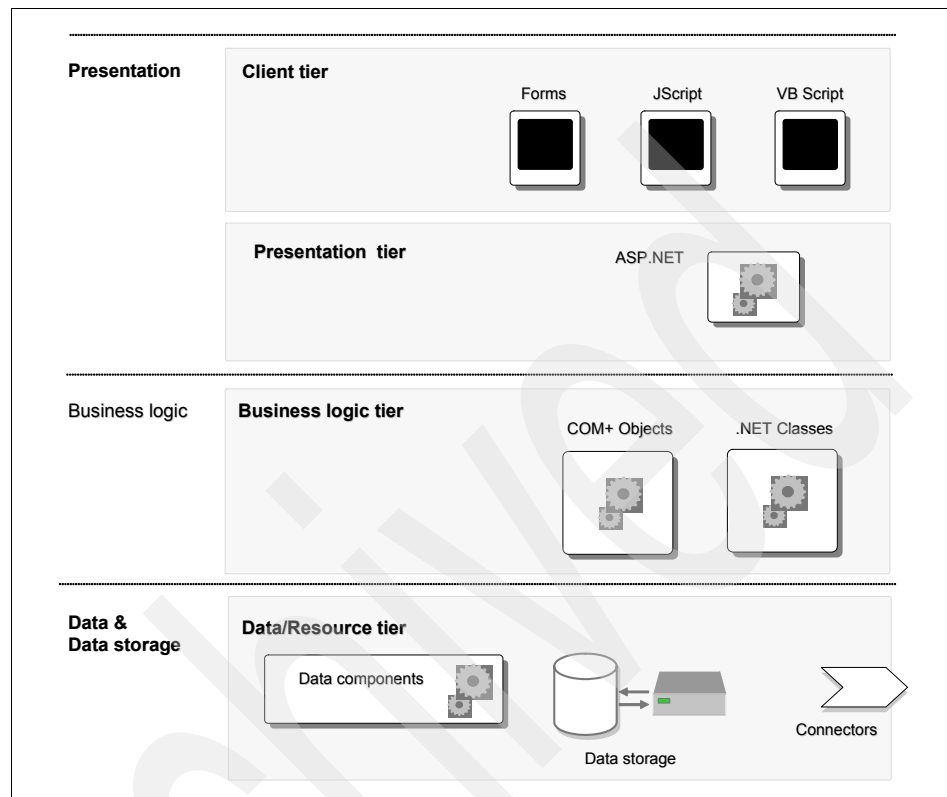


Figure 3-2 Layered Services Architecture advocated by Microsoft Patterns

The design patterns describe an n-tier model that is not specific about what the tiers contain, but rather extolls the benefits of layered design. They provide a simplified three-tier version of the n-tier model as a starting point for designing your own solutions. This three-tier model is similar to that advocated for Microsoft DNA. In Figure 3-2, we see that the Presentation tier of Microsoft's basic pattern encompasses both the client and Presentation tiers.

We will discuss each tier of Microsoft's model in more detail and place the components that make up each tier for clarity.

Presentation tier

The Presentation tier provides the user experience. For thin clients, it consists of ASP.NET (Active Server Page.NET) components that execute under Internet Information Services (IIS). Although the .NET Presentation tier may contain client-side scripting via VBScript or JScript®, for thin clients the ASP.NET components do the vast majority of the presentation work (see Figure 3-3).

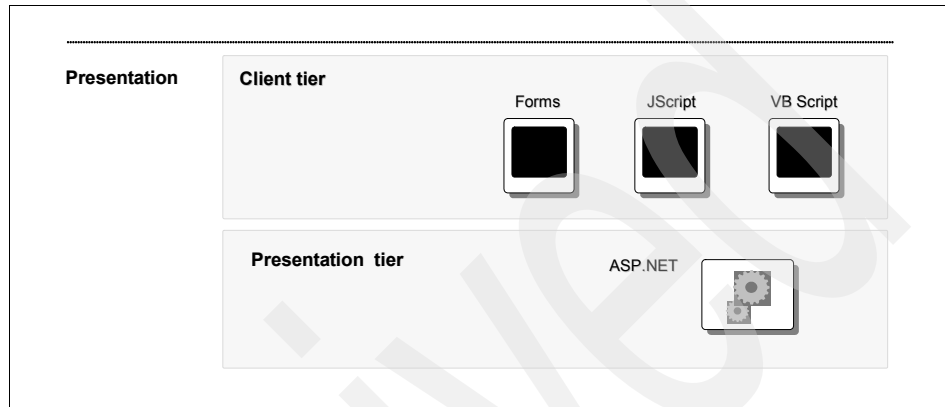


Figure 3-3 Presentation tier

ASP.NET can include other objects, and the interactions often follow the well-known model-view-controller pattern.

Business tier

The business tier provides reusable objects that perform discrete business functions. The business tier contains objects running under the .NET Common Language Runtime, known as “managed code”, and COM+ components, which do not run under the CLR, known as “unmanaged code” (see Figure 3-4).

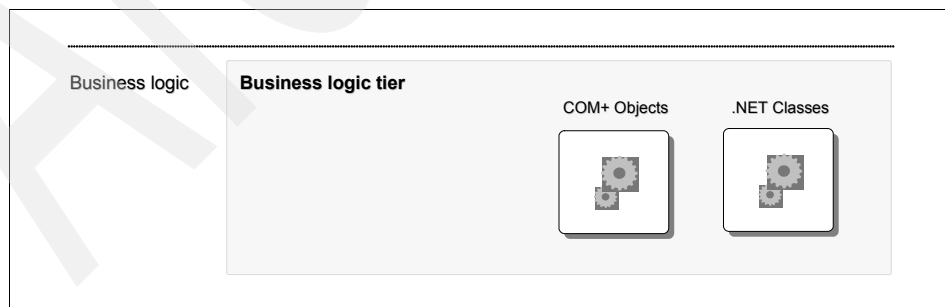


Figure 3-4 Business tier

The existence of these components in the business tier does not imply location. They may or may not run on the same machine as the Web server, application server, or the rich client application. They may run on dedicated application servers or even on the same machines as the resources they utilize.

When objects do not live on the same machine, .NET facilitates remote invocation between these objects using the so called “.NET Remoting”, which is analogous to RMI under J2EE and provides for remote invocation of .NET components via SOAP or a proprietary binary over TCP protocol.

Data/Resource tier

The Microsoft design patterns refer to this tier as the data tier; however, it can include services that provide access into any resource required by the application business components (see Figure 3-5).

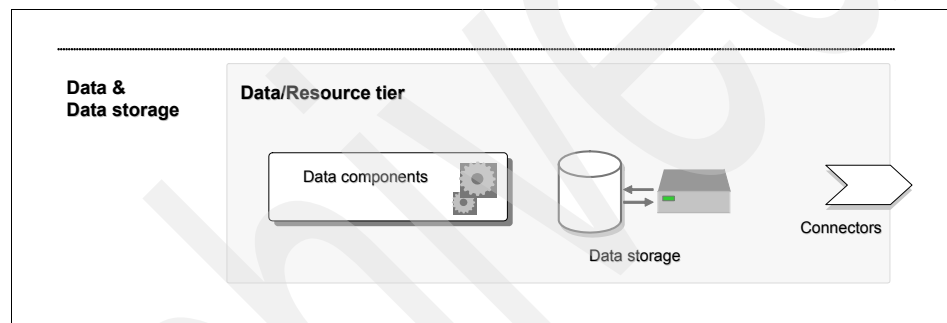


Figure 3-5 Data / Resource tier

The Microsoft patterns advocate abstraction of data and other resources with .NET Classes designed to provide access to Business tier objects. These components isolate the access code into a single location and abstract the data representation and access patterns from the business objects to minimize the impact to business objects of changes in data structure or providers.

3.2 The .NET suite

The Microsoft broad vision for Web Services encompasses a complete suite that includes the .NET Framework, a component of the Microsoft Windows operating system used for building and running Windows-based applications, Visual Studio .NET, which provides the application development environment for creating Web Services and applications, and .NET enterprise servers, which support application deployment, maintenance, management, and security.

3.2.1 The .NET Framework

The .NET Framework is the platform for creating and running .NET Web Services and Windows applications. It consists of three key components: the Common Language Runtime (CLR), the .NET Framework Class Library (FCL), and ASP.NET.

The .NET Framework is built on top of the Microsoft Windows operating system. Running any .NET application requires .NET Framework Version 1.1 Redistributable Package, which is similar to the Java Runtime Environment in a WebSphere environment. The .NET Framework Version 1.1 Redistributable Package is freely available from the Microsoft site. The following is a list of the various Windows operating system client and server platforms:

- ▶ Client
 - Microsoft Windows 98 and editions
 - Microsoft Windows Millennium Edition
 - Microsoft Windows NT 4.0 Workstation with Service Pack 6.0a or later
 - Microsoft Windows NT 4.0 Server with Service Pack 6.0a or later
 - Microsoft Windows 2000 Professional
 - Microsoft Windows 2000 Server family
 - Microsoft Windows XP Home Edition
 - Microsoft Windows XP Professional
 - Microsoft Windows Server 2003 family
- ▶ Server
 - Microsoft Windows 2000 Professional with Service Pack 2.0
 - Microsoft Windows 2000 Server family with Service Pack 2.0
 - Microsoft Windows XP Professional
 - Microsoft Windows Server 2003 family

Programming languages

While in J2EE the primary language is Java and applications run on many platforms, Microsoft .NET supports multiple languages that primarily run on the Microsoft Windows operating system platform. The .NET languages follow Common Language Specification (CLS), which is the minimum set of features that compilers must support on the target runtime. Currently, the .NET supports more than 20 languages, including vendor supported languages.

The Visual Studio .NET is shipped with following languages developed and supported by Microsoft:

- ▶ Visual Basic.NET
- ▶ Microsoft C#
- ▶ Microsoft J#
- ▶ Visual C++®

Visual Studio .NET also supports scripting languages, such as JScript.NET and VBScript.

The .NET Framework supports a number of languages. The compiler for each language translates the source code into the Microsoft intermediate language, which is a CPU-independent set of instructions that can be efficiently converted to native code (see Figure 3-6).

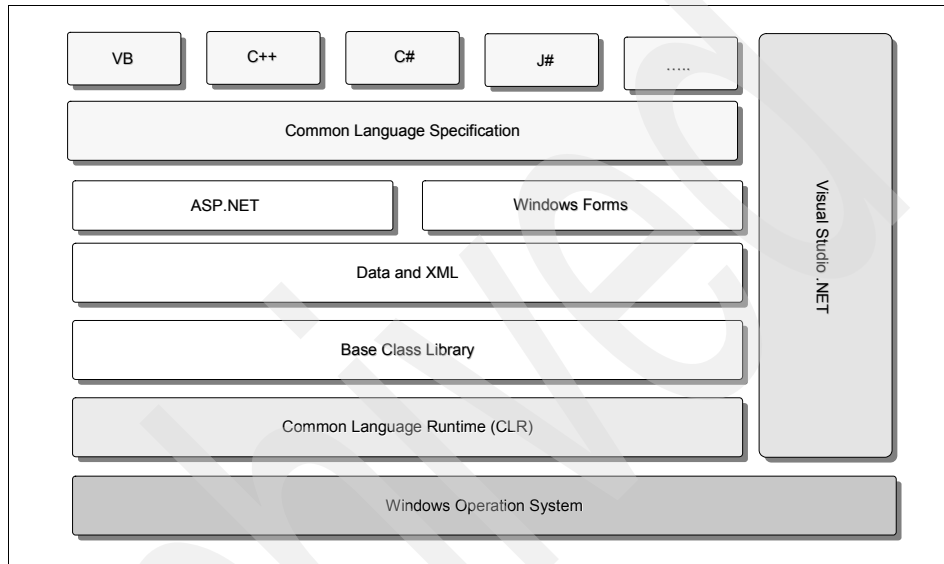


Figure 3-6 .NET Framework languages and tools

Common Language Runtime

The Common Language Runtime (CLR) is the key component of the .NET runtime environment. CLR is Microsoft's implementation of the Common Language Infrastructure (CLI) specification. CLI was initially created by Microsoft for .NET; it is now an ECMA standard (ECMA-335). It defines the structure of executable code and the virtual environment where it runs. The specification consists of parts that include the following four areas:

- ▶ The Common Type System (CTS)
Defines a rich set of types and operations that are supported by a wide range of programming languages. In .NET, CTS defines the common set of data types for all .NET languages, such as Visual Basic .NET and C#.
- ▶ Metadata
The CLI metadata defines the language and platform independent structure for describing types so that it can be referenced and persisted between languages, tools, and execution environments.

► The Common Language Specification (CLS)

CLS is the set of common language features that is a subset of the CTS. It is large enough to include the commonly needed developer constructs, yet small enough that it is supported by most languages. CLS defines features such as class overloading and inheritance, which language, tools, and framework designers agree on and developers rely on.

► The Virtual Execution System (VES)

Defines the software layer between CLI-compatible programs and the underlying operating system. It makes use of the metadata to bind the different modules at runtime, loads, and manages the execution.

The CLR acts as the virtual machine for the programs that execute in .NET. It provides the base on which the other .NET framework components are built. It is comparable to the Java Runtime Environment (JRE™) in J2EE.

CLR includes a number of components, as shown in Figure 3-7. Now let us look at what function each component provides.

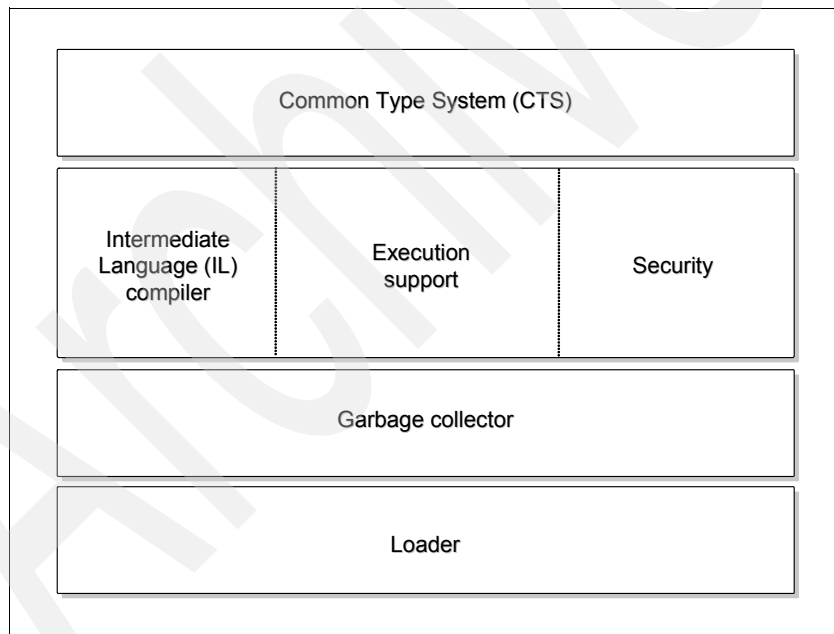


Figure 3-7 Components of the Common Language Runtime (CLR)

The Common Language Runtime loader

The CLR loader is responsible for loading each assembly (the basic unit of deployment in .NET; for more details, see “Assemblies” on page 107) into the appropriate application domain and controlling the memory layout of the type hierarchy within each assembly. The CLR loader component is similar to the class loader in Java.

Garbage collector

The garbage collector performs automatic memory management. Just as in Java, the developer is relieved from tracking and knowing when to free memory. The garbage collector reclaims memory for unreferenced and releases objects.

The garbage collector is able to identify and release objects that have circular references. This occurs when a parent object creates and references a child object which then creates and references another child object. If the second child references the first one, and the parent is being garbage collected, normally both children would be left alive since they still have open references. Fortunately, the .NET garbage collector is able to identify constructs like these and release them correctly.

The garbage collector determines when to run, yet it is also possible to manually initiate garbage collection. Just as in Java, with finalizers, you can influence the steps that are taken when releasing an object from the heap managed by the CLR. The garbage collector will call an object's finalizer method, giving you the opportunity to clean up, for example, to release database connections before the memory for the object is garbage collected.

You should also watch out for pitfalls when working with finalizers:

- ▶ Finalizers have a performance impact on the application.
- ▶ Finalizers are not called in a predictable sequence.
- ▶ Finalizers are not called when an application is being shut down, since all resources will be released anyway.

Note: The CLR relieves the developer from the responsibility of allocating and de-allocating memory. If your application tries to bypass the CLR and handle allocation and de-allocation of memory, it is considered *unsafe*, and the compiler would not compile your code. Using the key words *unsafe* and *fixed*, you can bypass CLR memory management; the resulting code is considered *unmanaged* code, as opposed to *managed* code, where CLR performs all allocation and de-allocation.

Intermediate language

Every .NET compliant compiler compiles the programming language specific code into a generic code, the Microsoft intermediate language (MSIL). The compilation process adds metadata to the IL-code produced. The metadata contains information about the types, references, and members of the code. It is used by the CLR during the execution to decide on the security features and the order to load classes to ensure the application runs correctly (see Figure 3-8).

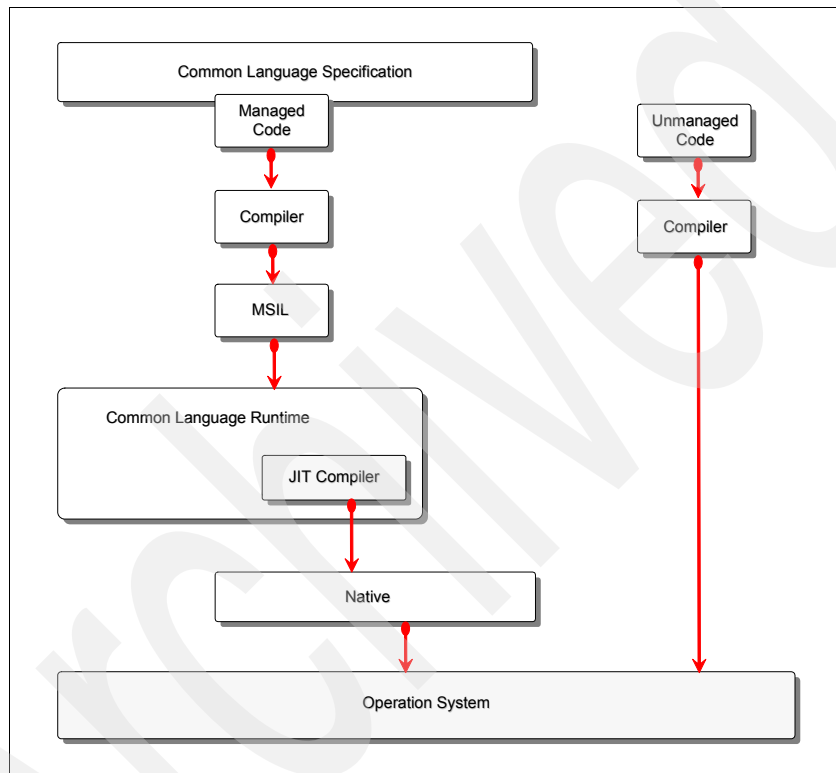


Figure 3-8 Steps in application program compilation

Since the IL-code produced during the compilation process is not machine specific, another core component of the CLR is needed to execute .NET applications. This part is the Just-In-Time (JIT) compiler.

The JIT takes common MSIL (Microsoft Intermediate Language) code and dynamically generates native code (x86 instructions for Intel® processors). The code generated takes advantage of a multiprocessor architecture, such as NetBurst architecture and Hyper-Threading technology. JIT also utilizes runtime information to provide further performance enhancements and sophisticated optimizations.

Execution support

The CLR provides execution support that exploits the architectural structure of processors for better performance. For example, with Intel's introduction of Hyper-Threading technology, a single physical Xeon® processor can appear as two logical processors. CLR takes advantage of Hyper-Threading and multi-processor systems through a thread pool, which optimizes runtime CPU utilization by spawning and blocking threads accordingly. CLR exploits this parallelism without the need for the developer to write additional code.

Note: The CLR server-optimized version (MSCorSvr.dll) is designated for deployment on multi-processor machines, such as those using the Intel Xeon MP processors. It has a garbage collector per CPU, with its own thread, and is participating simultaneously in the collection cycle.

Security

The CLR execution environment enforces a security model and policy that is independent of the host operating system security system. The CLR component-centric security model known as code access security is based on the origins of the code. It grants privileges to the code and not the user executing the code.

The CLR security system uses a security policy to determine what privileges to grant to the code in an assembly when it is loaded. The security policy accepts as input evidence and the in-memory representation of the assembly and produces permissions, which represent the rights to access protected resources, such as files and the rights to perform trusted operations as output.

The idea behind that is that an assembly, loaded from the application directory, has a higher probability of being harmless than code directly accessed over the Internet. This gives the application a different level of control over security, for example, with the user based approach, the code is classified by the user's permissions to access the application, but with the code access security model, the same user has different levels of security contexts, depending on the code location. This means that, for example, a privileged user could execute code from an unsafe source without automatically giving the code the access to all resources the user possibly could access.

The decision on the handling of the implemented permissions is then done on the so called evidences. These evidences are attributes the runtime evaluates that give the application information about the following:

- ▶ The URL (site and application directory where the piece of code is located)
- ▶ The hash value that identifies the piece of code

- ▶ The so called zone, that is, on Windows operating systems, filled according to the zones defined in the Internet Explorer® zone settings
- ▶ A unique signature of the assembly called the strong name
- ▶ The publisher's certificate, if appropriate

Common Type System

The support for multiple programming languages for development in .NET framework is provided by the Common Type System (CTS). The CTS provides a base set of data types to the upper layers of the .NET framework. This ensures interoperability within CLR, since all .NET compliant languages are based on the same set of data types. This feature of defining data types within the runtime itself independent of programming languages is important when it comes to code security. Also, since all types are unique across different languages, conversions and translations are therefore obsolete.

All data types that are exposed by the CTS are represented internally as objects. CTS supports two categories of data type objects:

- ▶ Value types, which directly contain a value
- ▶ Reference types, which contain only references to other objects

Value types and reference types are handled differently within .NET Framework. The differences between the two are as follows:

- ▶ Value types are created on the stack, while reference types are created using heap memory.
- ▶ Value types contain data, while reference types only hold references to other objects.
- ▶ Value types cannot be Null, while reference types can be set to Null.
- ▶ Value types mainly define the primitive types of the .NET Framework, while reference types provide complex types like classes and strings.

Assemblies

Assemblies are deployment units and are basic building blocks in the .NET Framework. When an application is compiled in .NET, the output of the compilation produces an assembly that contains Microsoft intermediate language code in a portable executable (PE) file that the CLR executes. There are two types of assemblies: process assemblies (EXE) and library assemblies (DLL). An assembly forms a single complete deployable unit to which security permissions can be requested and granted, type can be scoped to, and versioned in the CLR.

Associated with each assembly is a manifest that contains the assembly's metadata, which is a file table containing all files that make up the assembly and the assembly reference list for all external dependencies. In general, an assembly consists of four elements:

- ▶ **Assembly metadata**

The assembly metadata that includes the assembly's identity and version information (see Figure 3-9).

- ▶ **Type metadata**

The type metadata is used for resolving types and specifying the types that are exposed outside the assembly.

- ▶ **IL code**

The Microsoft intermediate language (MSIL) code.

- ▶ **Resources**

The resources in an assembly consist of image files like .bmp or .jpg and other files required by the application.

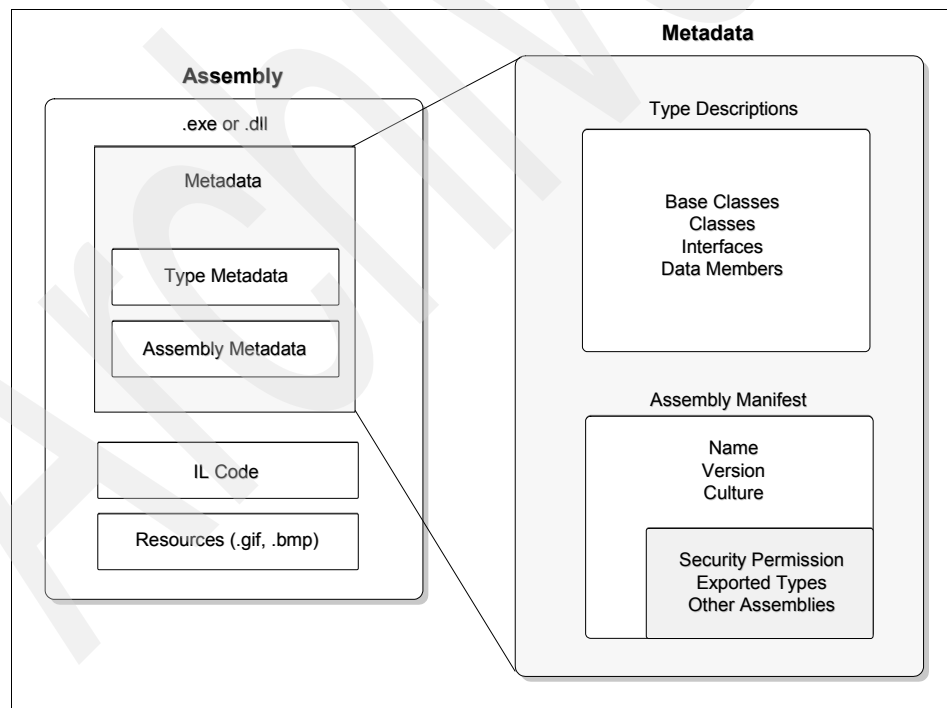


Figure 3-9 Assembly and metadata

The assembly can be either private or shared, depending on the visibility:

- ▶ Private assembly

A private assembly is an assembly that is visible to only one application and deployed within the directory structure of the application. The CLR finds these assemblies through a process called probing. The version information is not enforced for private assemblies, because the developer has complete control over the assemblies.

- ▶ Shared assembly

A shared assembly is used by multiple applications on the machine and stored in Global Assembly Cache. Assemblies deployed in the Global Assembly Cache supports side-by-side execution and must have a strong name to avoid any confliction. Side-by-side execution in the .NET Framework allow multiple versions of an assembly to be installed and running on the machine simultaneously, and allows each application to request a specific version of that assembly.

Configuration files in .NET

The common language runtime locates and binds to assemblies using a configuration file. The configuration file contains information related to application, machine, and security settings, which is stored in XML format. By updating the configuration file, the developer or administrator can change the way in which the application works. The .NET has three basic types of configuration files:

- ▶ Application configuration file

The application configuration file consists of information about the application. The Common Language Runtime uses the application configuration to get the information about assembly binding policy, the location of remote objects, and the ASP.NET runtime settings.

The settings for applications, such as Windows Forms and Console applications, are stored in the [applicationName].exe.config file. And the settings for Web applications, such as ASP.NET and Web Service, are stored in the Web.config file.

- ▶ Machine configuration file

The machine configuration file contains information about the machine that can be applied to the ASP.NET runtime, built-in remoting channels, and assembly binding. This file is located in the CONFIG directory of the runtime install path of the .NET Framework at <Windows install drive>\WINNT\Microsoft.NET\Framework\v1.1.4322\CONFIG.

Note: This file is stored in a different folder for each version of .NET. In the above example, V1.1.4322 refers to Version 1.1 of the .NET Framework.

► Security configuration file

Security configuration files contain information about the code group hierarchy and permission sets associated with a policy level.

The location of the security configuration files depend on policy levels. For example, the enterprise policy file `enterprisesec.config` is stored in the same folder as the `machine.config` file. And the user policy configuration file named `security.config` is stored in the user profile sub tree folder.

Versioning

The .NET Framework provides a robust scheme for versioning assemblies. By versioning assemblies, multiple versions of a same named object, in the same namespace, may be loaded into memory at the same time. This capability provides backward compatibility in cases where an application was built with one version of an assembly and does not work with follow-on versions of that assembly. This also virtually eliminates the possibility of back-leveling an assembly, a common problem with traditional Windows applications.

Figure 3-10 on page 111 gives an overview of object versioning within the .NET Global Assembly Cache.

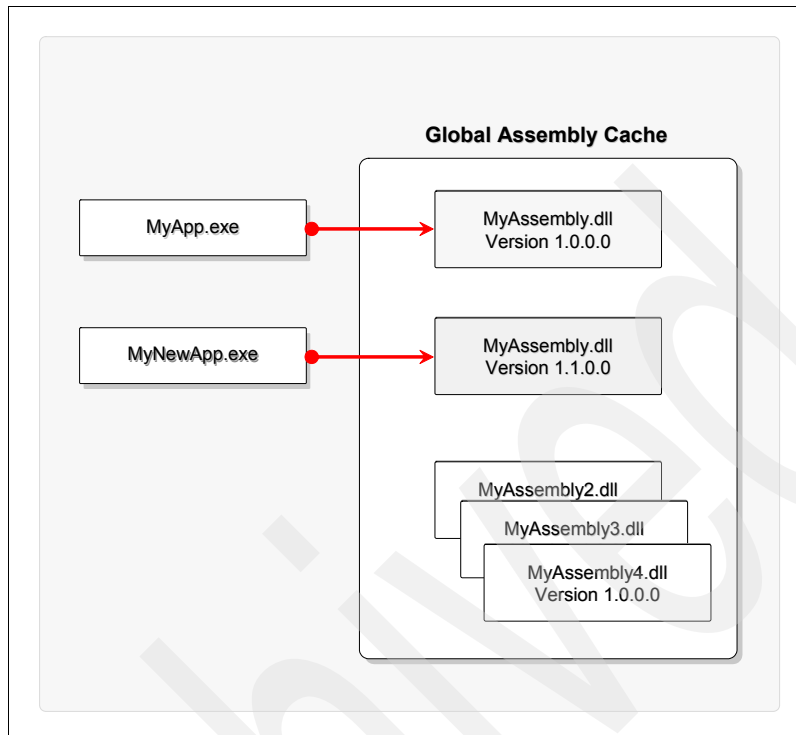


Figure 3-10 Object versioning within the .NET Global Assembly Cache

Assembly versioning is most transparent and effective when an assembly is strongly named and installed into the Global Assembly Cache (GAC). The .NET Global Assembly Cache is a repository for commonly used assemblies. Unless specified by configuration and policy files, the .NET assembly loader loads generically specified assemblies from the Global Assembly Cache before searching for and loading the same named assemblies from other locations. The Global Assembly Cache requires assemblies to be strong named and may contain multiple versions of the same assembly or same versioned assemblies with different culture information. See Figure 3-10 for an example of assembly versioning.

Note: There are other versions of .NET Framework, for example, it is possible to execute .NET applications using Microsoft's reference implementation for UNIX, called Rotor. There is as well a stripped down version of the .NET framework called the Microsoft .NET Compact Framework, which enables the developer to run .NET applications on smartphones and handhelds.

Framework class libraries

A common problem when programming with different languages in the Windows DNA world is the lack of a single widely accepted repository for dealing with common programming tasks. In the effort to unify development and put all languages on a single and solid base, Microsoft introduced a set of classes with ready-to-use functions. These classes are not only bundled with the .NET Framework, but are an integrated part of the framework itself. In fact, the framework uses the functions to accomplish basic tasks.

The .NET Framework Class Libraries (FCL) is a collection of interoperable, object oriented, and extensible types (classes, interfaces, structures, and enumerated types) that provide the foundation on which .NET Framework applications and components are built. You can use the classes as is or extend them by deriving new classes from the base classes. The FCL types are CLS-compliant and hence has the benefit that it can be used from any programming language with a compiler that conforms to CLS; this enables interoperability between the different programming languages.

The FCL is the .NET equivalent of the Java class libraries and just like the Java classes, it provides the following for the .NET platform:

- ▶ Defines base types and exceptions
- ▶ Defines well-known functions to perform common functions
- ▶ Defines abstract interfaces to hardware and operating system tasks

To handle this vast number of classes and give the developer a better orientation, Microsoft introduced the concept of namespaces to the framework. The commonly used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions are defined in the System namespace.

Figure 3-11 on page 113 gives an overview of a sample namespace.

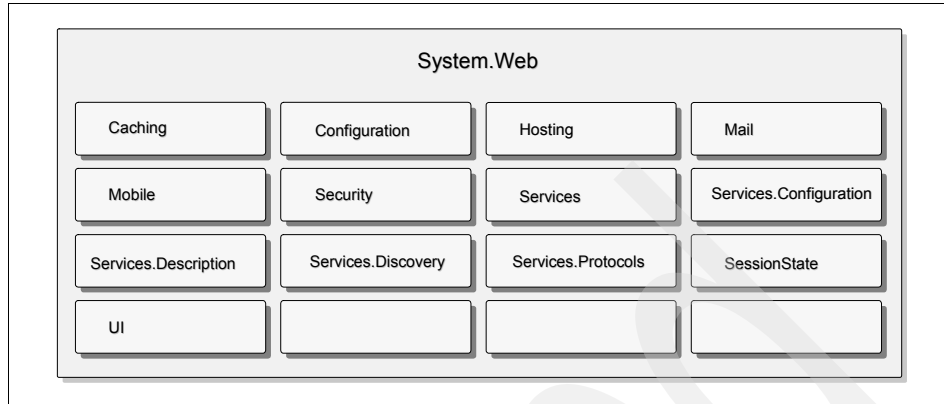


Figure 3-11 Sample namespace: System.Web namespace showing types

FCL uses a dot syntax naming scheme to group related types into a hierarchical structure. The naming scheme consists of two parts. The first part is the namespace name, which extends from the start of the name up to the right-most dot in the namespace's full name. The second part is the type name, which is the part beyond the right-most dot. For example, `System.Web.Services.Description` represents the `Description` type that consists of classes for describing XML Web service and belongs to the `System.Web.Services` namespace.

The following are examples of FCL namespaces and types:

- ▶ `System.Xml`
Provides standards-based support for processing XML.
- ▶ `System.Xml.Schema`
Contains the XML classes that provide standards-based support for XML Schemas definition language (XSD) schemas.
- ▶ `System.Xml.Serialization`
Contains classes that are used to serialize objects into XML format documents or streams.
- ▶ `System.Net.Sockets`
Provides a managed implementation of the Windows Sockets (Winsock).
- ▶ `System.Reflection`
Contains classes and interfaces that provide a managed view of loaded types, methods, and fields, with the ability to dynamically create and invoke types.

A description and complete list of the .NET namespaces are available online at:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/cpref_start.asp

Example 3-1 shows some sample code that uses System namespaces.

Example 3-1 Sample code showing use of System namespaces

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace ItsoClaim
{
    [WebService(Namespace="http://dotnet.claim.examples.itso",
    Name="ItsoClaim")]

    public class ItsoClaim: System.Web.Services.WebService
    {
        public ItsoClaim()
        {
            InitializeComponent();
        }

        #region Component Designer generated code

        private IContainer components = null;

        private void InitializeComponent()
        {
        }

        protected override void Dispose( bool disposing )
        {
            if(disposing && components != null)
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #endregion

        [WebMethod]
        public Boolean findCustomer(String customerID,String policyID )
```

```

    {
        CustomerDataAccess customerObj = new CustomerDataAccess();
        try
        {
            return customerObj.getCustomer(customerID, policyID);
        }
        catch (DataException de)
        {
            throw new ClaimException(de.Message);
        }
    }

    [WebMethod]
    public string registerClaim(String customerID,String policyID, DateTime
accidentDate,
    String accidentDescription, String [] involvedCars)
    {
        ClaimDataAccess claimObj = new ClaimDataAccess(customerID, policyID,
accidentDate,accidentDescription, involvedCars);
        try
        {
            return claimObj.getClaimCode();
        }
        catch (DataException de1)
        {
            throw new ClaimException(de1.Message);
        }
    }
}
}

```

3.2.2 ASP.NET

The ASP.NET technology is the central technology when it comes to Web development under .NET. It provides a scalable infrastructure and programming model for creating dynamic Web applications. ASP.NET is one of the .NET Framework components and hence enables the use of most .NET compatible languages, including Visual Basic, C#, and J#, for developing Web applications.

With ASP, you were limited to using HTML and VBScript (or Jscript) to process and render pages; however, with ASP.NET, you are able to separate code from display, and by the introduction of controls, you are able to do a whole lot more than just display information. The ASP.NET programming model extends to communicating and sharing data over the Internet using Web Services and to mobile devices, such as cell phones and PDAs.

The main new feature of the ASP.NET technology is the use of Web controls. Web controls offer an implementation of the presentation tier that can be used just like Windows Forms classes and objects. It is therefore possible to assemble dialogs and Web pages by using preconfigured building blocks, for example, a calendar control, that is already fully functional. When using Web controls, the user of the control will fire events on the client side that are transferred over the net, causing the server to react. Web controls are therefore essential for rapidly developing Web applications. Note that the use of Web controls causes events and messages to be exchanged with the server.

Another difference between ASP and ASP.NET is in the configuration of Web and Internet Information Server (IIS). With ASP.NET, all important configuration information is in human readable XML files. This enables the modification of configuration information without having to modify the code. The settings for culture and multi-language support can be modified on the fly through the configuration without having to deploy server side applications.

The technologies embraced by the ASP.NET framework include the following:

- ▶ Scripting for Web pages
- ▶ Management of sessions and states
- ▶ Security
- ▶ Server side controls
- ▶ Web forms
- ▶ Caching mechanisms
- ▶ Binding to data sources
- ▶ Configuration framework
- ▶ Multi language capabilities

Model-View-Controller design pattern using ASP.NET

To demonstrate the design of a simple ASP.NET application and to illustrate the Model-View-Controller roles in the ASP.NET environment, we present an example that involves creating an ASP.NET application to access a database.

In our example, we use three files: SimpleForm.aspx, SimpleForm.aspx.cs, and SimpleDataGateway.cs.

SimpleForm.aspx contains the description of our user interface and is defined in HTML with the following ASP.NET declaration, as Example 3-2 shows.

Example 3-2 ASP.NET declaration

```
<%@ Page language="c#" Codebehind="SimpleForm.aspx.cs" AutoEventWireup="false"
Inherits="CodeBehindExample.SimpleForm" %>
```

The Inherits attribute tells the ASP.NET system which class the .aspx file inherits from at runtime. The Codebehind attribute is used by Visual Studio .NET but it is not used at runtime. Note that using the same file name for the .aspx file and the .cs file is a convention used in the Visual Studio .NET Application wizard. To make the user interface SimpleForm.aspx use a different code-behind class, just change the Inherits attribute (and the Codebehind attribute if you are using Visual Studio). For a description of ASP.NET page directives, see:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconpage.asp>

SimpleForm.aspx.cs contains the event handling code for user interface defined by SimpleForm.aspx and ties it to SimpleDataGateway. Because SimpleForm is named as the code-behind class for SimpleForm.aspx, the object must inherit from the .NET class System.Web.UI.Page or a subclass of Page (see Example 3-3).

Example 3-3 Inheritance for SimpleForm

```
/// <summary>
/// The description for SimpleForm
/// </summary>
public class SimpleForm : System.Web.UI.Page {
```

SimpleDataGateway contains all of the code needed to use the database (see Figure 3-12).

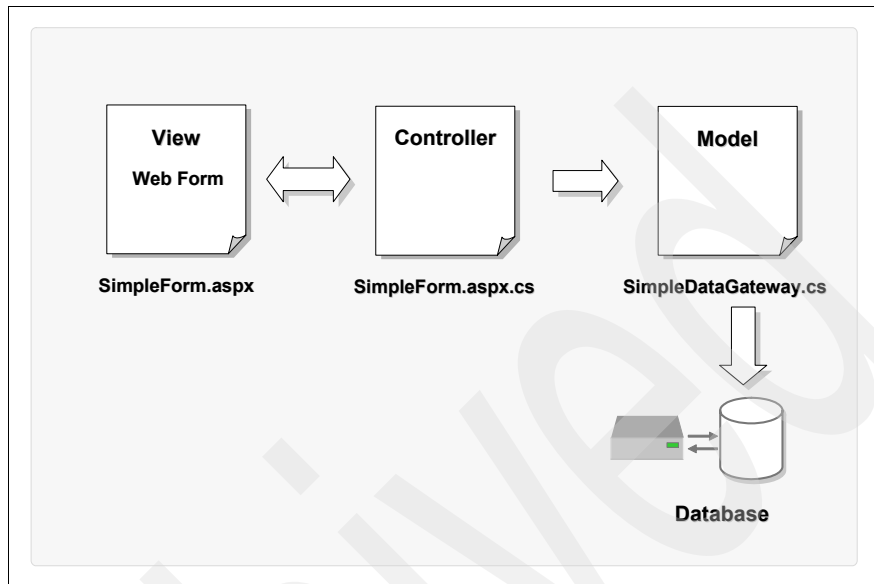


Figure 3-12 Model-View-Controller roles in ASP.NET

When a browser requests SimpleForm.aspx, the ASP.NET runtime looks at the @Page information and loads the precompiled assembly with the class specified in the Inherits attribute. The UI is then displayed and the events denoted by the code-behind object that the page inherits from are handled as they occur.

A common design flaw in ASP.NET applications is embedding the Model code in the Code-behind object filling the role of MVC role of Controller. Combining the Controller and Model makes it difficult to reuse the code filling the role of the Model. In our example, combining the Controller and Model could force us to refactor if we create another page that wants access to the database (see Example 3-4 on page 119).

Example 3-4 Model implementation

```
private void InitializeComponent() {  
    this.TestButton.Click += new System.EventHandler(this.TestButton_Click);  
    this.Load += new System.EventHandler(this.Page_Load);  
}  
private void TestButton_Click(object sender, System.EventArgs e) {  
    // Don't put the database driving code in your codebehind object!!!  
    // This violates the MVC design pattern.  
    SqlConnection AConnection =  
        new SqlConnection("server=(local);database=adb;Trusted_Connection=yes");  
}
```

ASP.NET added a significant amount of features and functionality offered by ASP. If you are developing an application using Web and .NET technologies, you should count on using ASP.NET as a central technology for the Microsoft Windows platform.

3.2.3 .NET enterprise servers

Included in the .NET suite are a set of products and servers that include the Windows operating system and Enterprise servers that support the .NET initiative. The suite of Enterprise servers provide the reliable, manageable, and scalable platform for .NET enterprise solutions.

- Windows 2003 Server

Windows 2003 Server provides the base infrastructure that powers applications, networks, and Web Services for communication, collaboration and connected solutions in workgroups and the data center. For more information, visit:

<http://www.microsoft.com/windows2000/server/evaluation/business/overview/default.asp>

- BizTalk® Server 2002

One of the promises of Web Services is the integration of business processes across the extended enterprise. Making use of Web Services and Web technologies, BizTalk Server 2002 enables organizations to build, execute, and manage business processes across the extended enterprise. For more information, visit:

<http://www.microsoft.com/biztalkserver/>

► Commerce Server 2002

Commerce Server 2002 is Microsoft's highly customizable platform for e-commerce Web sites. Its features include deployment of personalized portals, content targeting, user profiling, and analytics. Commerce Server 2002 extensibility capabilities enable transaction-based sites with capabilities for catalog management, order processing, and merchandising. For more information, visit:

<http://www.microsoft.com/commerceserver/>

► Content Management Server

The Microsoft Content Management Server supports the .NET Framework and provides the environment for creation and publishing of Web content and ASP.NET applications using Microsoft Visual Studio .NET. It also enables users to manage Web content through an Internet browser or the Microsoft Word application. For more information, visit:

<http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28001368>

► Exchange Server 2003

Exchange Server 2003 provides the infrastructure for managing messaging and collaboration. It offers built-in functions, such as e-mail, calendaring, contact and task management, scheduling, and online forms. Exchange Server 2003 is also a development platform for custom collaboration and messaging-service applications accessible on desktops, and mobile devices. For more information, visit:

<http://msdn.microsoft.com/exchange/>

► SQL Server

The Microsoft SQL Server is a relational database management system (RDBMS) that provides advanced data management and functionality, which includes high performance and high availability features to support e-commerce, line-of-business, and data warehousing solutions. SQL Server 2000 supports processing of XML data and capabilities to import and export XML-formatted. For more information, visit:

<http://msdn.microsoft.com/sqlserver/>

► Application Center 2000

Application Center 2000 is the key component for the deployment of scalable and highly available .NET platform. It ensures that software and hardware failures do not disrupt application services. Application Center 2000 actively monitors performance and health of cluster of servers, and automates responses to adverse events and conditions. Built on Windows 2000,

Application Center 2000 collects and presents performance and event-log data for one server or the entire cluster. For more information, visit:

<http://www.microsoft.com/applicationcenter/>

- **Mobile Information 2002 Server**

Mobile Information 2002 Server functions as a gateway enabling mobile users access to enterprise applications, data, and intranet content. Mobile users can securely access e-mail messages, contacts, calendars, tasks, or intranet line-of-business applications while on the go. For more information, visit:

<http://www.microsoft.com/miserver/evaluation/overview/default.asp>

- **Host Integration Server 2000**

Host Integration Server 2000 enables connection to host systems by providing integration components and transactions to host data sources, messaging, and security systems from .NET applications. For more information, visit:

<http://www.microsoft.com/hiserver/default.asp>

3.3 Developing .NET applications

In the Microsoft .NET environment, the development can be done using either a text editor like Notepad or the Visual Studio .NET Integrated Development Environment (IDE). This section covers how to write, compile, and run an application in your development environment.

3.3.1 Writing a C# application using text editor

This is same as writing Java code in a text editor and then compiling using the command prompt. Let us take an example of writing a C# application in Notepad.

The steps are:

1. Write the code in the text editor of your preference:
 - a. Save the code with a .cs extension.
 - b. Compile and run the application.

Figure 3-13 is a very simple C# code written in Notepad.

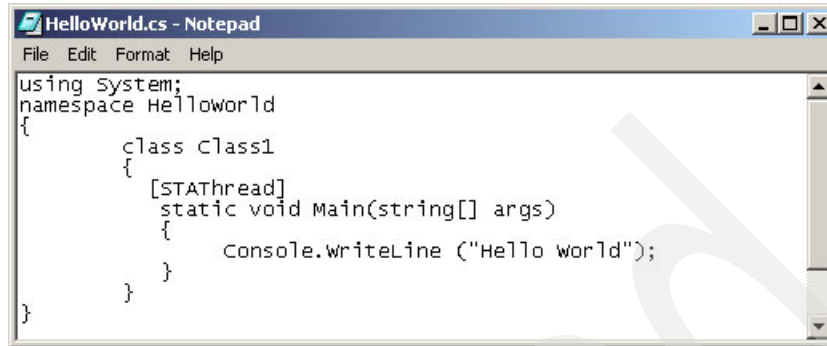


Figure 3-13 C# code in Notepad

2. Compiling the code using a command prompt.

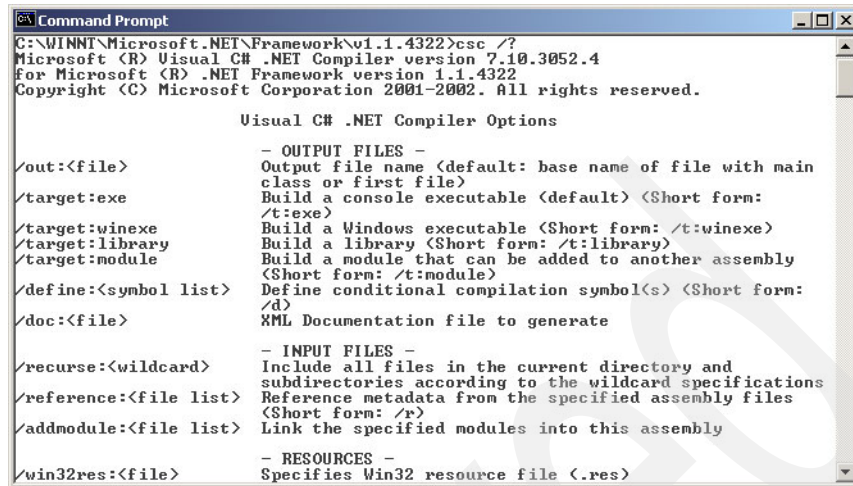
The C# compiler has different options to compile the source file; some of them are listed in Table 3-1.

Table 3-1 C# Compiler options

Compiler Option	Result
csc HelloWorld.cs	Creates HelloWorld.exe file.
csc /target:library HelloWorld.cs	Creates a HelloWorld.dll file.
csc /out:OtherName.exe HelloWorld.cs	Creates an OtherName.exe.

To learn more about compiler options, run the **csc /?** command.

Figure 3-14 on page 123 gives an overview of the available compiler options.



```
C:\WINNT\Microsoft.NET\Framework\v1.1.4322>csc /?
Microsoft (R) Visual C# .NET Compiler version 7.10.3052.4
for Microsoft (R) .NET Framework version 1.1.4322
Copyright (C) Microsoft Corporation 2001-2002. All rights reserved.

Visual C# .NET Compiler Options

- OUTPUT FILES -
/out:<file>          Output file name (default: base name of file with main
                    class or first file)
/target:exe         Build a console executable (default) (Short form:
                    /t:exe)
/target:winexe      Build a Windows executable (Short form: /t:winexe)
/target:library     Build a library (Short form: /t:library)
/target:module      Build a module that can be added to another assembly
                    (Short form: /t:module)
/define:<symbol list> Define conditional compilation symbol(s) (Short form:
                    /d)
/doc:<file>          XML Documentation file to generate


- INPUT FILES -
/recurse:<wildcard> Include all files in the current directory and
                    subdirectories according to the wildcard specifications
/reference:<file list> Reference metadata from the specified assembly files
                    (Short form: /r)
/addmodule:<file list> Link the specified modules into this assembly

- RESOURCES -
/win32res:<file>     Specifies Win32 resource file (.res)
```

Figure 3-14 Available compiler options

3. Running the application from the command prompt.

To run an executable in Java environment, you have to pass the file name with a Java command. In .NET environment, to run an executable, you simply type the name of the .exe file and press Enter. In the following snippet, HelloWorld.cs is first compiled using the **csc** command and then run by just typing the executable file's name (see Figure 3-15).



```
C:\WINNT\System32\cmd.exe

C:\ITS0>csc HelloWorld.cs
Microsoft (R) Visual C# .NET Compiler version 7.10.3052.4
for Microsoft (R) .NET Framework version 1.1.4322
Copyright (C) Microsoft Corporation 2001-2002. All rights res

C:\ITS0>HelloWorld
Hello World

C:\ITS0>
```

Figure 3-15 Compiling and running the C# program

3.3.2 Developing applications using Microsoft Visual Studio .NET

Developing applications in the Visual Studio .NET Integrated Development Environment is quite simple, fast, and allows for Rapid Action Development (RAD). The Environment is integrated with the Visual Source Safe (VSS) version control software for easy management of code. The editor in Integrated Development Environment has some common features with the IBM Rational Application Developer for WebSphere Software, such as IntelliSense® (type-ahead) and drag-drop design.

The Visual Studio .NET has different editions, which include the Professional, Enterprise Developer, and Enterprise Architect editions. In addition to these editions, special language specific editions are available, such as Visual Basic.NET Standard Edition and Visual C#® Standard Edition, primarily for hobbyist, students, and beginners.

When you open the Microsoft Visual Studio, you get the Start Page with three panes; the Projects pane has details about the history of the projects, the Online Resources pane to get the online samples, and the My Profile pane, to set the personalized settings (see Figure 3-16).

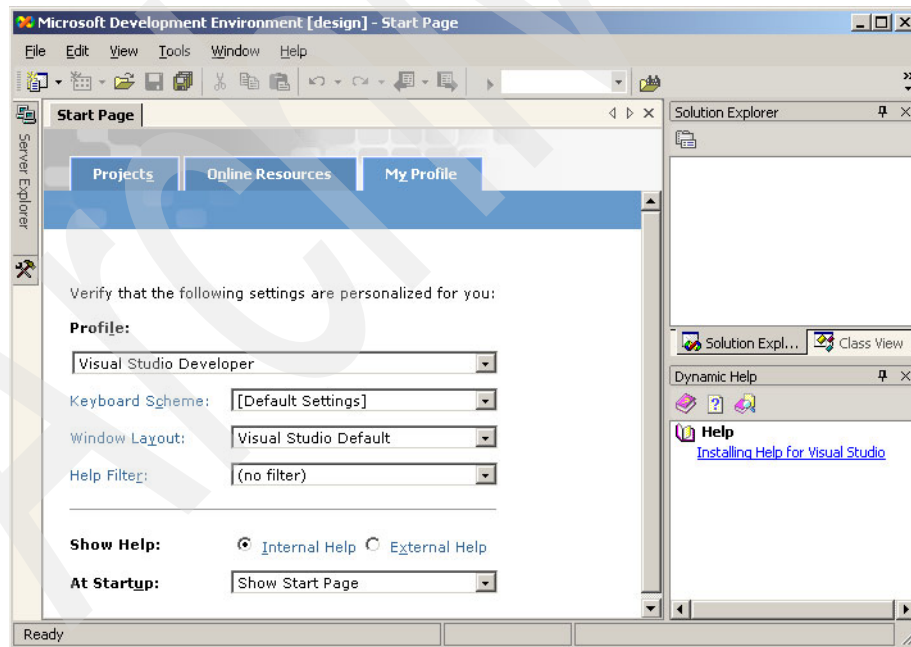


Figure 3-16 Start Page in Visual Studio .NET

The Integrated Development Environment has different windows, such as the Tool window, Code window, Properties window, and Solution Explorer window.

Except for the Code window, the windows can be hidden using the Auto Hide feature. For example, in Figure 3-17, the Server Explorer and the Toolbar windows were hidden using the Auto Hide feature.

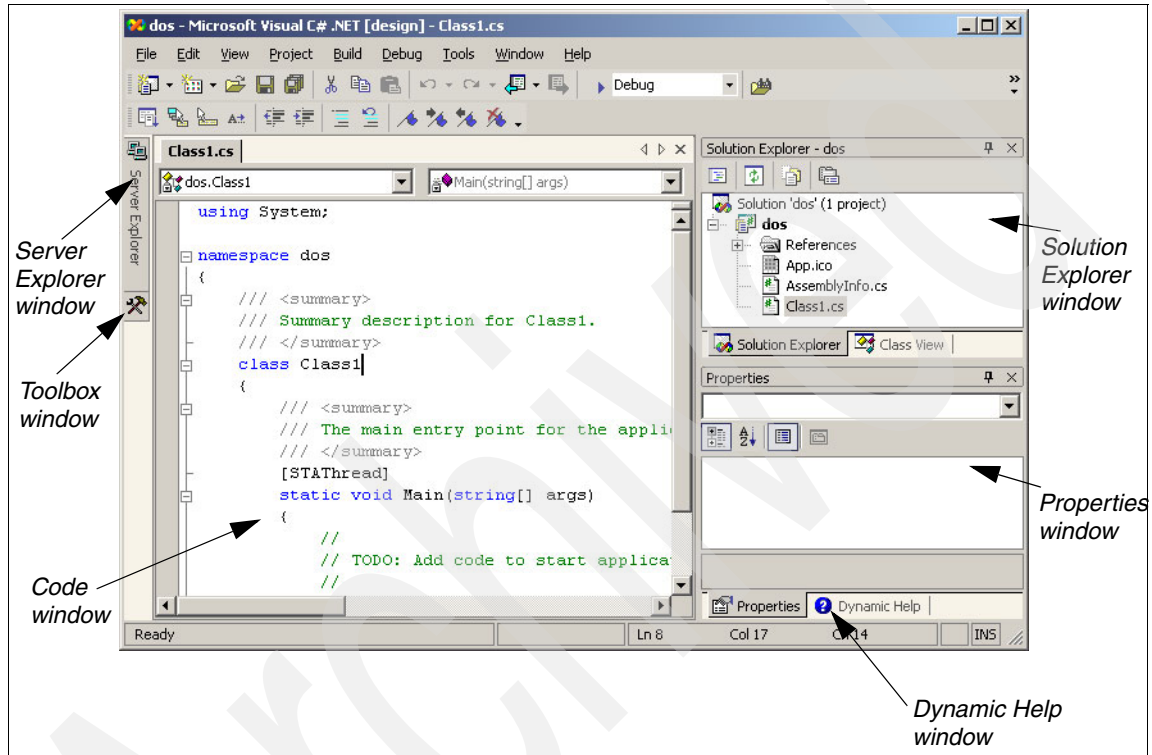


Figure 3-17 The Visual Studio Integrated Development Environment

To start a new project in Visual Studio, select **File** → **New** → **Project** from the main menu.

The New Project window appears, as shown in Figure 3-18.

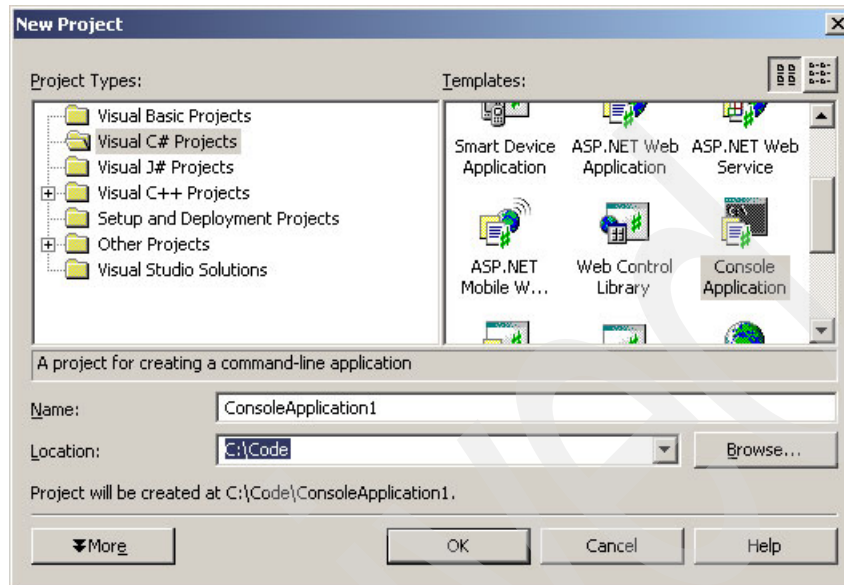


Figure 3-18 The New Project window in Visual Studio .NET

You can create different types of projects in the IDE by simply selecting the appropriate template. You can work with more than one project at a time. This feature make debugging Class Libraries with another application possible.

Table 3-2 describes various templates available for projects.

Table 3-2 Project templates

Template	Description
Windows Application	Creates a new Windows or Desktop project that refers to traditional rich client applications.
Class Library	Creates a new DLL Project or creates components that typically encapsulate some business logic.
Windows Control Library	Creates a new control project that can be used on a Windows application as a traditional ActiveX control.
Smart Device Application	Creates a new smart device project, for example, PDAs.
ASP.NET Web Application	Creates a new ASP.NET Web application project (dynamic and data driven browser based applications).
ASP.NET Web Service	Creates a new Web Service project.

Template	Description
ASP.NET Mobile Web Application	Creates a mobile application project.
Web Control Library	Creates a Web control project that can be used on a Web form, allowing code reuse and modularization.
Console Application	Creates a console application project to run on a DOS command prompt.
Windows Service	Creates a Windows service project that runs in the background.
Empty Project	Creates an empty project or solution. You can avoid creation of default files and can insert your own files.
Empty Web Project	Creates an empty Web project.

To create an application, select the language for coding from Project Types and then select the application type from Templates.

To build the project, various options are available in the Build menu; to run the project, click the **Run** button on the tool bar.

Source code management

Source code management in the Visual Studio .NET environment can be realized by using The Microsoft Visual Source Safe (VSS). The Microsoft Visual Source Safe is a repository for managing project files. The VSS is tightly integrated with Visual Studio and therefore has some advantages over other third party source code management products.

A user can see the latest version of any file, make changes to the file, and save a new version in the VSS source code management database. Users can lock files for modification or save and release the lock on the files by performing check-out or check-in operations respectively on the files.

3.3.3 Testing

Application testing and debugging involves various activities on individual deployment units, as well as across deployment units (applications). The goal is to ensure that deployment units work, individually and collectively, as designed.

Debugging and unit testing

Functional testing that occurs during the development and initial debugging process primarily involves the IDE and it is often an integral part of developing the classes and methods that make up the deployment unit.

Visual Studio.NET provides the essential tools for the software developer. Applications are developed under Visual Studio.NET as Projects within a Solution file. Each Project results in a separate assembly. Debugging options and features are, therefore, set at the project level. This allows the developer to control which debug features are enabled for each assembly being tested.

Debug settings

Several features of the Visual Studio.NET debugger is customizable, allowing the user to control such things as warning levels, setting of breakpoints, and how data members and variables are displayed, resulting in a personalized environment for each developer.

Building a project for debugging: debug builds

The build configuration for each project is selectable, allowing the developer to build release versions of some components while continuing to debug others. Generic Debug and Release configurations are provided by default. Additional custom configurations can also be created for each project.

Any assembly built from a .NET project built with the Debug configuration set can be fully debugged via Visual Studio.NET.

Figure 3-19 shows the configuration manager.

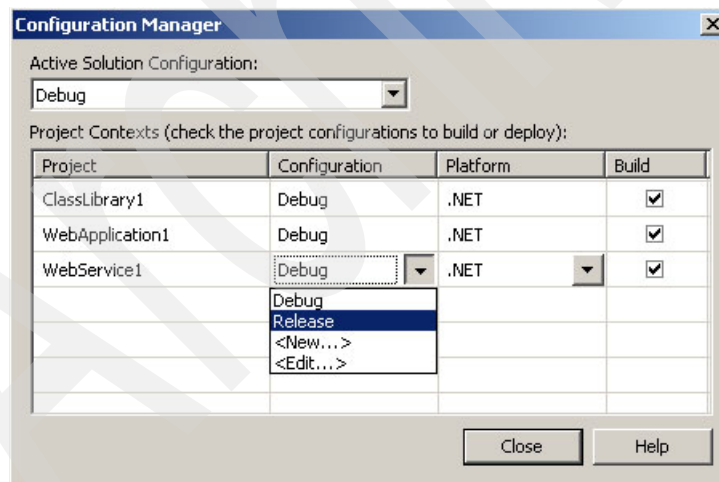


Figure 3-19 Configuration manager

Code animation

Visual Studio.NET IDE provides debugging animation support for a large majority of the code that is developed within the environment. This includes the following types of animation:

- ▶ Multi-language animation: ASP.NET, VB.NET, C#, C++, and C
- ▶ Support for managed (CLR) and unmanaged (binary) code and their interoperation via .NET Interop
- ▶ Services, Server-side COM+ objects, DLLs, and other unmanaged objects
- ▶ The ability to animate multi-threaded code
- ▶ Animation of Class libraries
- ▶ Animation of .NET assemblies that take advantage of Remoting, both from the client and server perspectives
- ▶ Animation of Web Services as they are invoked
- ▶ Remote debugging of code running on other physical nodes
- ▶ Animation of SQL Stored Procedures for SQLServer

Visual Studio.NET IDE allow you to step through all execution regardless of threading or language concerns as long as the project was built with debug options. Features available during animation include stopping execution, stepping through, over, and into code, running to cursor, and resetting the current line. In addition, you can view and modify variables, view registers, and the memory space in which your process is operating. You can debug one or multiple processes simultaneously.

With symbols provided by Microsoft as part of the Platform and Framework SDKs, you can also step through framework and operating system code executed by your application.

The debugger allows you to decide how to handle exceptions during debugging and provides an edit and continue feature that allows you to edit your code in place during a debugging session and continue without rebuilding and starting over.

Just-in-time debugging

The bugs that escape unit testing are usually the ones that occur in special circumstances. These are often difficult to duplicate in a debugging environment. A unique feature of Visual Studio.NET is the ability to turn on Just-In-Time Debugging for a given assembly.

With just-in-time debugging enabled, an exception within an assembly that was started and is running outside the IDE will cause a dialog box to appear, providing an opportunity to enter the debug mode with any debugging tool you specify. This enables a developer to easily identify the location of the problem and the current execution state.

Note: In Visual Studio.NET, just-in-time debugging across systems (remote debugging) was disabled due to security issues.

Tracing and debugging instrumentation

The Microsoft .NET provides a set of classes that allow the developers to monitor the execution of the application while it is running. This process of enabling tracing and debugging is called *instrumenting*.

Once the code is instrumented, you decide whether the tracing and debugging statements will actually be included in the assembly by setting the Trace and Debug conditional attributes in your project build settings. This prevents release versions of the application from writing to debug listeners. Further control of trace output is also available by including a Trace Switch object in your source code to set severity levels and control which trace statements will produce output.

Instrumented applications can make information available about a wide variety of execution and behavior parameters that can be used to better understand the executable.

Other debugging tools

Microsoft also provides some other utilities that are useful for debugging various parts of a .NET application. Many of these utilities have been enhanced for the .NET environment. Here is a brief list of other debugging tools:

- ▶ **ISAPI Web Debug Tool**
Allows debugging of ISAPI filters commonly used in Web applications.
- ▶ **Spy++**
Tracks controls, windows, threads, and processes during execution to aid in debugging.
- ▶ **Visual Studio command prompt**
Most things that can be done in Visual Studio.NET can also be done from the command line. This simply opens a command-line session with all of the proper environment variables set for Visual Studio.NET command-line operations.

Performance and load testing

An application is only as useful as its ability to perform in terms of speed and number of users it supports. This section discusses the performance tools, techniques and practices commonly used for .NET applications.

Performance Monitor and counters

The Performance Monitor is a tool provided by the Microsoft operating system for selectively measuring or recording the performance of any application, process, or thread running on a local or remote system. Performance Monitor works by listening to “performance counters” that are installed with the Common Language Runtime and .NET Framework Software Developer’s Kit.

A performance counter is an object that reports performance information. The Windows operating system and .NET applications automatically inherit a default set of performance counters. These performance counters provide a large array of performance information, such as memory, handles, threads, locking, loading of code, and networking. Performance Monitor can be used to get a detailed view of the internal performance of any .NET application regardless of where it is running.

Figure 3-20 shows the Performance Monitor.

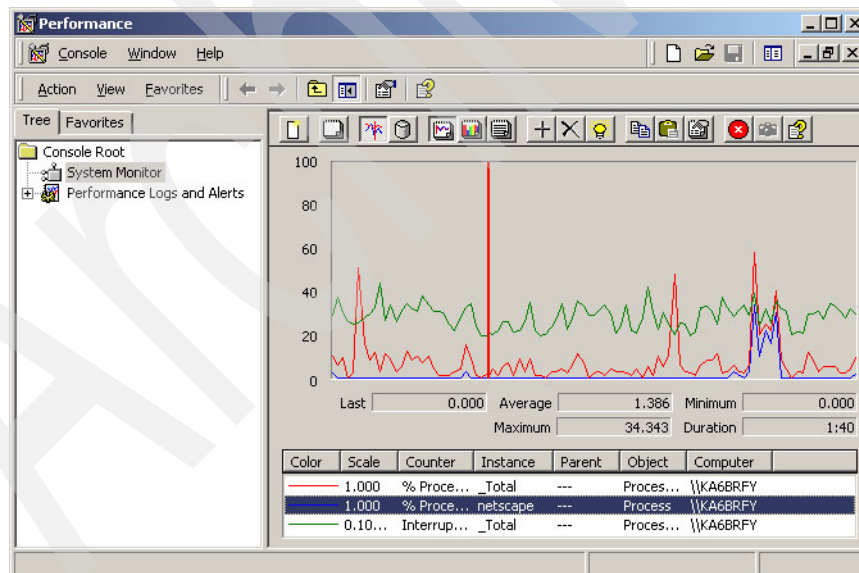


Figure 3-20 Performance Monitor

Classes provided in the System.Diagnostics namespace within .NET Framework can be used to instrument your application with custom performance counters or to consume performance counter information programmatically in the same way Performance Monitor does.

By instrumenting your application using these classes, you can implement useful counters in specific areas of your application where performance might be an issue and the existing counters do not supply this information, for example, when counting the number of people who use a specific feature of your application on an ongoing basis.

Once implemented, these counters can then be viewed by anything that can consume these counters, such as Performance Monitor or a custom listener you create within your application.

The Microsoft Application Center Test

Included with Microsoft Visual Studio.NET for Enterprise Architects is a tool for simulating several users simultaneously accessing various parts of your Web application from browsers. This tool is called Application Center Test (ACT) (see Figure 3-21 on page 133).

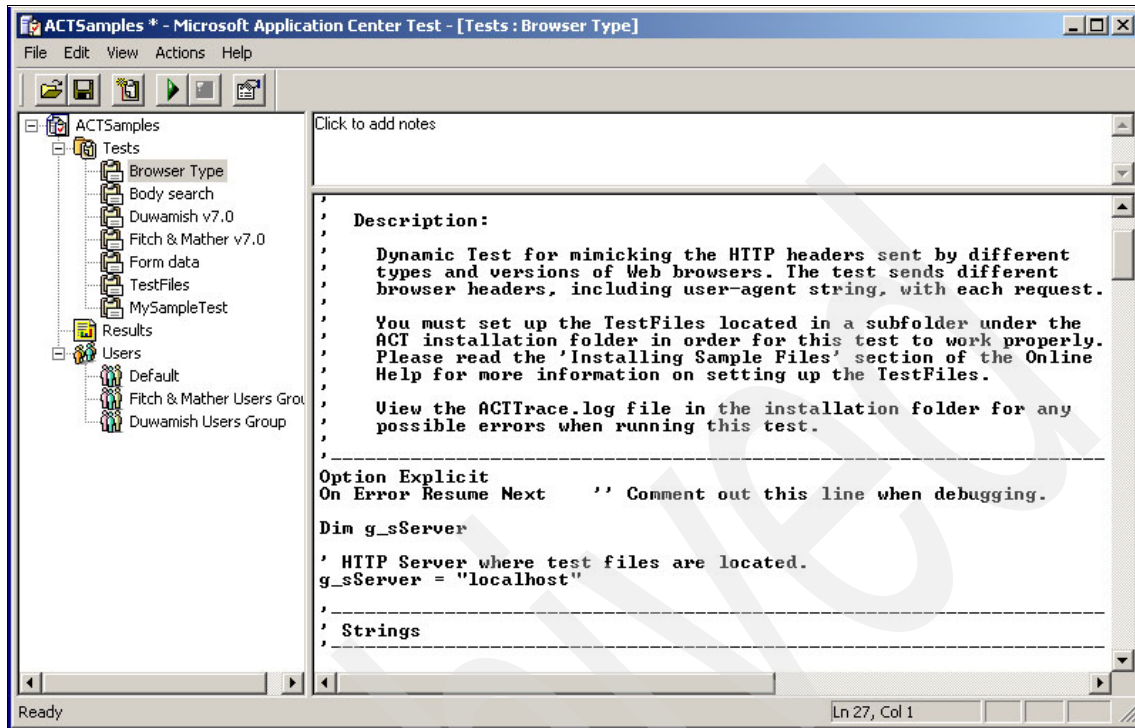


Figure 3-21 Application Center Test multi-browser-type sample

ACT allows you to observe various parts of the application while it is running with a simulated load. There are various methods for creating tests, including recording real-time browser sessions. You can simulate multiple groups of users simultaneously accessing a Web site using different types of Web browsers and different versions of Web browsers. Test sessions are scripted using either VBScript or JScript and the tool comes with several sophisticated examples. A large variety of third-party testing facilities are available.

3.4 Deploying and managing .NET applications

A key aspect of .NET applications is the distributed architecture and dependency on network infrastructure. Effective management and administration in .NET environment requires net aware tools. The Windows operating system, .NET Framework, and Visual Studio .NET provide the support and tools for deploying, running, and administering applications in the .NET distributed networked environments.

3.4.1 Deployment

Deployment in .NET is quite easy and different from the traditional model. It does not require registration and hence does not have the DLL and COM registration and versioning issues. Deployment in .NET consists of packaging and distribution, which can be performed in any one of a number of ways.

By copying files

This is the simplest way of moving a .NET application from one location to another location. This can be done by using either the **Copy Project** command available on the Project menu or by using the **XCOPY** DOS command.

Note: This method has limitations. Copying files does not register or verify the location of assemblies, and for Web projects, it does not automatically configure IIS directory settings.

By using Setup and Deployment programs

A Setup and Deployment program is the automated installation approach of deploying the applications in a work environment. The advantages of using Setup and Deployment projects are as follows:

- ▶ You can avoid overwriting of files that could cause other applications to break.
- ▶ Registering and deploying of COM components or assemblies can be implemented.
- ▶ The program shortcut can be added in to Windows' Startup menu or onto the Windows desktop.
- ▶ Interaction with the user, for example, storing user information, licensing, and so on.

The Visual Studio .NET has several ways of creating a Setup and Deployment project. Moreover, third party Setup and Deployment programs can be used for deploying the .NET applications.

Figure 3-22 on page 135 shows various Setup and Deployment projects available in Visual Studio .NET.

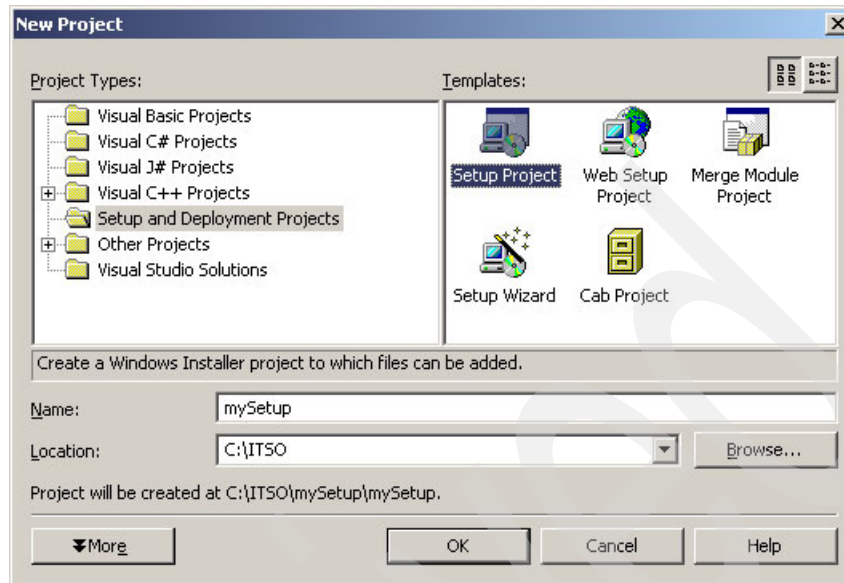


Figure 3-22 Setup and Deployment Projects in Visual Studio .NET

Table 3-3 describes various project types available in Visual Studio .NET.

Table 3-3 Project types available in Visual Studio .NET

Project type	Purpose
Merge Module Project	The Merge Module Project allows you to share setup code between Windows Installers and avoids versioning problems. This creates a merge module (.msm file), a single package that contains all files, resources, registry entries, and the setup logic necessary to install a component.
Setup Project	The Setup Project builds an installer for a Windows-based application in order to distribute an application. The resulting Windows Installer (.msi) file contains the application, any dependent files, information about the application such as registry entries, and instructions for installation. The Setup project installs the files into the file system of a target computer.
Web Setup Project	The Web Setup Project builds an installer for a Web application. The setup installs files into a virtual directory of a Web server.
Cab Project	The Cab project builds a cabinet file for downloading to a Web browser.

3.4.2 Runtime

The .NET Framework includes the Common Language Runtime, which provides the runtime for Windows-based applications, and a set of class libraries, which .NET applications use for common services, including access to operating system features such as the Windows.Forms namespace.

At the time of this writing, the current version of the .NET Framework is V1.1. The .NET Framework Version 1.1 package provides all the necessary support needed for developing and running applications using the .NET Framework. It was first included as part of the Windows operating system with the release of Windows Server 2003. It is also integrated with Visual Studio .NET 2003. You can also obtain it by itself as the .NET Framework Version 1.1 redistributable package.

For more information about the .NET Framework Version 1.1, please see the following Web location:

http://msdn.microsoft.com/netframework/downloads/framework1_1/

3.4.3 Administration

The various products that provide the features underlying .NET, such as the operating system, Enterprise Services, COM, Internet Information Services, SQL Server, and other products, all have their individual management capabilities and tools. These tools are available remotely in nearly every case and are conveniently collected together within the Control Panel under Administrative Tools, as shown in Figure 3-23 on page 137.

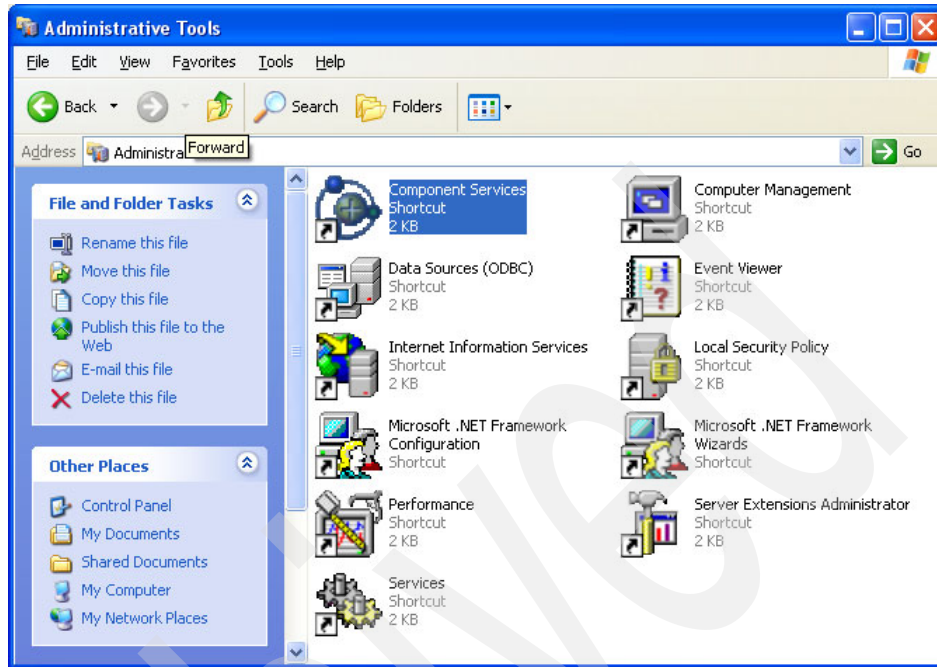


Figure 3-23 Windows administrative tools

It is beyond the scope of this redbook to address each of these tools. However, we will briefly discuss a few key Windows management components.

Microsoft Operations Manager

The Microsoft Operations Manager provides event-driven management for the Windows platform and applications that implement Windows Management Instrumentation (WMI). It allows consumption and utilization of information provided through WMI and provides a platform on which to generate alerts or take action based on specific events. For more information about Microsoft Operations Manager, see the following Web site:

<http://www.microsoft.com/mom/default.mspx>

Internet Services Manager

The Internet Service Manager is the main administrative console for Internet Information Services (IIS). It is used to administer and manage IIS nodes and Web sites, and works for both local and remote nodes.

The .NET Framework Configuration Tool

The .NET Framework 1.1 Configuration tool is a Microsoft Management Console (MMC) snap-in used for the following:

- ▶ Manage and configure assemblies in the Global Assembly Cache
- ▶ Adjust code access security policy
- ▶ Adjust remoting services

Active Directory

The Active Directory® provides a number of critical functions for the Windows operating system, including providing a common directory for distributed components, acting as a central authentication provider, managing users, groups and individual machines. Active Directory provides a Group Policy feature that enables administrators to define a host of policies for groups of users or nodes. A technology known as *IntelliMirror*® uses Group Policy to enable software distribution and configuration management capability for groups of machines and people.

For more information about the use of Active Directory in the administration role, please refer to:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vstchdeployingvsusingactivedirectory.asp

3.4.4 Windows Services

Windows Services were formerly known as NT Services. They have been around a long time. Windows Services are applications that run in the background. They can be started and stopped by an administrator locally and remotely whether a user is logged on the system or not.

Windows Services often provide background “daemon-like” functions in the Windows operating systems and typically do not interact with the screen. .NET applications may function as NT Services.

The execution life cycle of Windows Services is managed by the Service Control Manager, which uses methods implemented by the application developer to manage the execution life cycle.

As before, Windows Services differ significantly from other types of applications written for .NET. For a more complete description of both the life cycle management process and the differences between Windows Services and other applications, please see the Microsoft Web site at:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbc_onintroductionontserviceapplications.asp

3.4.5 Object pooling

Object pooling can be used to improve application performance in cases where object instantiation is a relatively time consuming operation. When a poolable object is instantiated (typically by an object factory), it is placed into a pool. When an object is needed, an unused object may be retrieved from the object pool. Not only does connection pooling help to reduce instantiation costs, it also helps to reduce the cost of garbage collecting additional objects at runtime. In general, object pooling is most beneficial when objects can be and are frequently reused.

There are two primary types of object pooling provided by the .NET Framework. The most common type is pooling of ADO.NET database connection objects. This type of pooling is very similar to JDBC connection pooling. In ADO.NET, pooling occurs on a per-provider basis. For example, SQL Server and DB2 providers each provide their own connection pooling facility. ADO.NET connection pooling is typically controlled by an option on the connection string.

The second type of object pooling is provided through interfaces and classes provided by the System.EnterpriseServices namespace. This namespace provides .NET components access to COM+ enterprise services, including object pooling. Objects pooled in this manner must inherit from the ServicedComponent class. By applying the ObjectPooling attribute to a serviced component, values such as the maximum and minimum pool size may be specified to tune performance.

3.4.6 Remote invocation

Similar to Java Remote Method Invocation, .NET provides a mechanism called Remoting. Remoting provides an object framework allowing .NET objects to run and communicate in a distributed environment. Remoting in .NET does not use Microsoft's DCOM or COM+ facility. Instead, it is exposed through a set of .NET classes. It offers greater flexibility than DCOM, including the ability to communicate over multiple types of channels. By default, the .NET Framework provides two types of channels: TCP and HTTP. These channels have both advantages and disadvantages that make them better or less suited for specific types of applications.

The .NET TCP channel uses a binary socket connection, similar to DCOM. The TCP channel is used in conjunction with a binary object formatter. The main advantage of using TCP is that it is typically faster, since data is transmitted in binary format. There is little to no additional parsing involved in packaging and unpackaging data to be sent over the network. The main disadvantage of using a TCP channel is that it requires additional TCP ports to be accessible and is typically only suited for an intranet environment.

The .NET Framework provides an additional channel, the HTTP channel, for providing remoting capabilities. The HTTP channel uses the XML SOAP format for remote method invocation. The advantage of using the HTTP channel is that it uses the standard HTTP protocol so it can be used over the Internet without opening additional TCP/IP ports in a firewall. The disadvantage of using a HTTP channel is that it is typically slower due to the added processing required when formatting requests and replies to and from XML and .NET objects.

3.4.7 Web Services

The Microsoft .NET Web Services Framework is similar to IBM WebSphere in that it uses standard protocols and services to provide and locate Web Services.

In addition, the .NET Framework provides components for building Web Services with these standard protocols and services:

- ▶ Simple Object Access Protocol (SOAP)
The .NET Framework supports both RPC and Document style SOAP.
- ▶ Web Services Discovery Language (WSDL)
- ▶ Universal Description, Discovery, and Integration (UDDI)

In addition to using automated features of Visual Studio .NET to facilitate creating Web Services, Web Services can also be created in a less convenient, but more flexible manner. The .NET Software Development Kit (SDK) contains several tools for working with Web Services:

- ▶ soapsuds.exe
The soapsuds tool generates runtime assemblies capable of accessing a remoting service provider given a XSD definition.
- ▶ wsdl.exe
The wsdl tool generates Web Services and Web Service client proxies based on a WSDL definition.
- ▶ disco.exe
The disco tool searches a specified Web server for Web Services discovery documents and saves them locally. Typically, disco.exe is used to discover Web Services and then wsdl.exe is used to generate client code to access the service.

All these tools are built based on classes provided by the .NET Framework. Therefore, the same functionality is available at runtime and may be used within an application for dynamic Web Services discovery and consumption.

3.4.8 Transaction management

Transactions are a key aspect of building remote and distributed applications. By using transactions, remote and distributed applications can run multi-step operations and roll back any or all of these operations if there is a failure. The .NET Framework provides support for the following transactional components:

- ▶ Microsoft Message Queuing (MSMQ)

Microsoft Message Queuing support is provided directly through objects in the .NET System.Messaging namespace. This namespace contains the object MessageQueueTransaction, which provides MSMQ level transactional support. This is typically called a one-phase or level one transaction, since it encapsulates one level of processing.

- ▶ ADO.NET data transactions

ADO.NET data providers typically include database level transaction support. The larger ADO.NET database providers, including IBM DB2 UDB, Microsoft SQL Server, and Oracle, also provide the ability to allow database connections to automatically participate in distributed transactions. Like Microsoft Message Queuing, non-distributed ADO.NET transactions are a one-phase transaction.

- ▶ Serviced components

The .NET System.EnterpriseServices namespace, as introduced in 3.4.5, “Object pooling” on page 139, also includes the ability to provide distributed transaction support. .NET Serviced components are built on top of COM+ and must be registered as a COM+ component. A registered component may participate in distributed transactions involving other transactional components. Wrapping a Microsoft Message Queuing or ADO.NET transaction in a distributed transaction is generally referred to as multi-phase commitment.

The level of transactional support required varies greatly from application to application and must be determined on a per-application basis. Generally, transactions involve a considerable amount of overhead and should only be used where they are required. Non-distributed transactions typically have less overhead but are limited to their respective component, while distributed components require COM+ services and must coordinate transactions through a third entity. On Microsoft platforms, this entity is the Distributed Transaction Coordinator (DTC). A complete discussion of the Distributed Transaction Coordinator is beyond the scope of this redbook.

3.4.9 Security

Security is a critical component of enterprise applications. The ability to effectively provide authentication, access control, data integrity, privacy, non-repudiation, and auditing within an enterprise application are some of the requirements of building a secure system.

The .NET Framework contains a multi-layered security approach to meet these requirements. Each layer provides various services that may be used to secure applications at different levels. At the most general level, .NET provides the following security services:

- ▶ Operating system level security

At the lowest level, the operating system controls access to the file system and other system level resources.

- ▶ Runtime code level security

The .NET Common Language Runtime performs runtime type verification and code validation. These two features help to eliminate code execution problems caused by type mismatches, bad function pointers, memory bounds overruns, and many other runtime problems.

- ▶ Tamper-proof assemblies

Assemblies can be strong named by signing them with a public/private key pair. When the assembly is signed, a hash is generated based on the contents of the assembly. This hash is then encrypted with the private key. If the contents of the assembly are changed and the assembly is not re-signed, the hash will no longer match, thus the assembly will be marked as corrupt. Signing assemblies can be extremely important in keeping them from being injected with malicious code.

- ▶ Role-based security

The role-based security allows for application-level security within the .NET Framework. Role-based security can be configured at the enterprise, machine, and user levels. This is accomplished by creating permission sets based on attributes, such as site, URL, and publisher, and then applying them to code groups. Policies can be configured directly with XML configuration files or by using the .NET configuration snap-in, available from the Windows control panel.

Note: Role-based security is configured using XML files located on the file system. Without access restrictions on these files, they can be easily modified to allow unauthorized access to resources.

► Secure Sockets Layer (SSL)

Secure Sockets Layer is the industry standard for network data encryption. It may be used transparently by various .NET components to provide secure network communications.

► Data encryption components

The System.Security.Cryptography namespace of .NET Framework includes classes for cryptography, hashing, and authentication. Many standard symmetric and asymmetric cryptography providers are supported, including RC2, RSA, and TripleDES.

► Authentication services

Several methods of authentication are available for use by .NET applications and services. Authentication comes into play when connections are initiated with a remote service. The most common users of authentication services are:

– ASP.NET

Supported authentication methods include Integrated Windows, Forms, and Microsoft Passport. However, authentication can also occur at the IIS Web server level. IIS allows anonymous, basic, digest, certificate, and integrated authentication.

– ADO.NET

Authentication methods vary from provider to provider. For example, the Microsoft SQL Server provider supports both integrated and SQL Server authentication.

– Remoting

When hosted by IIS, remote objects can use IIS and ASP.NET Windows authentication. If a TCP channel is used, there is no built-in authentication service available.

– Messaging

Built-in authentication, authorization, and encryption services are available when using messaging services.

– Web Services

IIS and ASP.NET services are also available to Web Services. Other custom approaches, such as passing them as part of the SOAP header, are also available.

3.4.10 Load balancing and failover

Load balancing and failover capabilities under .NET come from the underlying and supporting technologies upon which it is built.

Server clustering

Server clustering has been around since the days of Windows NT and has been enhanced to provide additional features. With clustering, a multi-server Web application can provide service despite hardware failures on individual services.

Although clustering is very powerful, it is a non-trivial implementation that must be properly planned in advance. Shared disk storage is ideally used between the clustered servers to hold common configuration and state information. The application is written to react to lost connectivity and services by re-establishing connections to database and other resources when they suddenly become unavailable.

For more information about Microsoft Server Clustering, see the Microsoft Web site at:

<http://www.microsoft.com/windows2000/technologies/clustering/default.asp>

Network Load Balancing

The Windows Server provides a feature known as Network Load Balancing, which is designed to evenly distribute Web-based traffic between servers and, should a server become unavailable, reroute traffic to another server.

In practice, Network Load Balancing is also a function of the network itself. Any considerations of this should involve careful selection of the technology to be used based on the application requirements (HTTP, FTP, SMTP, ports, and others).

3.4.11 Application logging

Application logging includes the ability to track and monitor performance, provide auditing capabilities, and debug problems. The .NET Framework provides several technologies that can be used to provide standard logging features.

These technologies are:

- Performance counters

The .NET Framework provides components for creating, updating, monitoring, and grouping performance counters.

- Trace class

The .NET Framework includes a standard class named Trace for tracing program execution. Trace may be enabled using online precompiler directives, compiler options, or at runtime using the configurable TraceSwitch class.

- Windows Management Interface (WMI)

A .NET application can monitor and record system status using the Windows Management Interface. This resource may be accessed through classes in the System.Management namespace.

- Windows event log

Complete access to the Windows event log is available through the EventLog component in the System.Diagnostics namespace. Using this component, applications can create new events, read existing events, and respond to events requiring input. See Figure 3-24 for an example of entries created in the Windows event log using .NET.

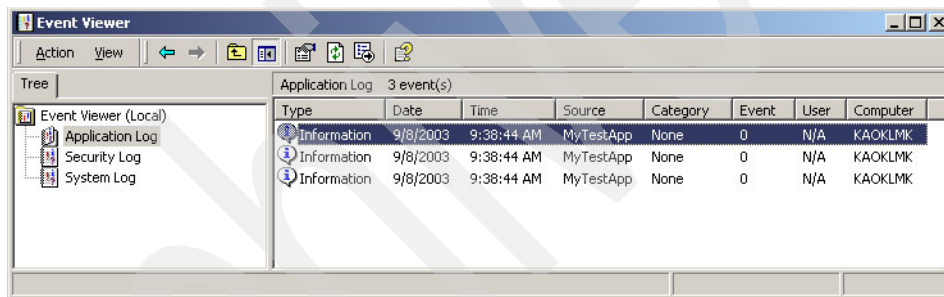


Figure 3-24 Creating entries in the Windows event log using .NET

In addition to these facilities, additional logging for Web applications is provided by Internet Information Services (IIS) and ASP.NET. Internet Information Services and ASP.NET provide the following additional features:

- ASP.NET component trace

This type of logging can be enabled on a per-application or per-page basis. ASP.NET trace allows trace data to be displayed on the current page or in an external document.

- Internet Information Services Web site and application-level configurability

The majority of logging configuration capability resides as site level configuration options. At the site level, options such as a client IP Address, user name, and host port may be chosen. At the application level, the option whether to log visits may be changed.



Part 2

Component interoperability

Archived

Introduction to component interoperability

This chapter introduces interoperability between WebSphere/J2EE and .NET applications at the component or class level.

This chapter contains the following sections:

- ▶ Components overview
- ▶ Introduction to component interoperability
- ▶ Why choose component interoperability?

4.1 Components overview

The term *component* has been loosely used to refer to the different objects that are present in any object oriented application. Both J2EE and .NET architectures build on the object oriented paradigm. They both have different classes of components, each one of them having a different behavior and functionality. Some of these component classes are inherently designed to be server side component classes, as they need services provided by the server side containers. On the other hand, some of the component classes are essentially client classes, while some others can be used both on the server side as well as the client side. The following sub-sections describe these different component classes.

4.1.1 Client side components

Client side components are incorporated into Graphical User Interface (GUI) programs that execute on the desktop or in the Web browser running on a user's machine. This means that the client components usually run on a client machine separate from the application server.

Both J2EE and .NET clients can be classified as Web browser based (thin clients), smart, or rich clients. Web browser based clients are usually thin clients, as they communicate to a server where most of the business logic is implemented. Web clients access the server side components over HTTP, while stand-alone clients may make use of many protocol, such as SOAP over HTTP, SOAP over JMS, HTTP, or any custom protocol over TCP to access the server side components.

J2EE

In the Java/J2EE world, Web clients typically consist of Web pages containing various markup languages (HTML, XML, and so on) generated by Web components running in the Web container on the server and embedded scripts, such as Javascript, DHTML, and Java Applets. Rich and smart clients include plain old Java objects or J2EE client applications, which run in the application container on the client machine.

Java applets

A Java Applet is a component (Java class) that typically executes in a Web browser but can also run in a variety of other applications or devices. An applet must be loaded, initialized, and run by its container, for example, the Web browser. It is used to process presentation logic, and it provides a powerful user interface for J2EE applications. Applets embedded in an HTML page are deployed and managed on a J2EE server, although they run in a client machine. Applets execute inside a sandbox and the browser enforces constraints on what

applets can do. Typically, applets are not allowed to access the local file system or make network connections. They can consist of all kinds of plain old Java objects, including Java Beans. Even though Java applets typically access the server side components over HTTP, they are not restricted to HTTP alone and can access the server side components over any protocol.

Stand-alone client components

Stand-alone client components could either be a part of a J2EE application client or a plain old Java client. A stand-alone application client is a program that can be launched from the command line or desktop and runs on a client machine. For example, it can access business logic implemented in EJBs running on the J2EE application server. Although a stand-alone application component is a Java class, it differs from a Web client in that it runs in the client application container. This means that it has access to all the facilities on the client machine. The container provides the runtime support for the stand-alone application. Furthermore, it runs in its own Java Virtual Machine (JVM). The stand-alone client components provide a way for users to handle tasks that require a richer user interface than can be provided by a markup language. They typically have a graphical user interface (GUI) created from Swing or Abstract Window Toolkit (AWT) APIs, but a command line interface is certainly possible. They can also make use of Java Beans.

Figure 4-1 gives an overview of J2EE client components.

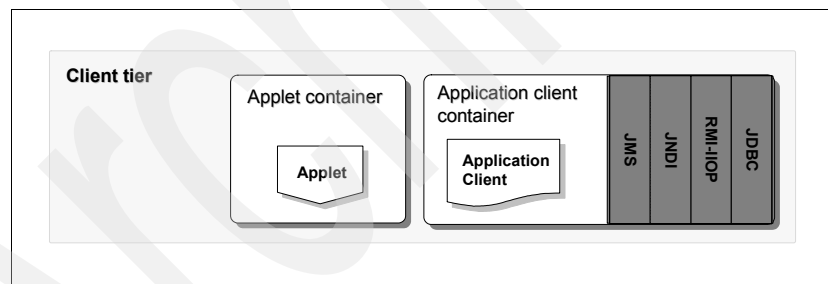


Figure 4-1 J2EE client components

.NET

Web clients in the .NET world consist of Web pages that can have embedded controls, components, and scripts (usually written in VBScript or JScript). Stand-alone clients include the .NET console clients and the graphical user interface clients based on Windows forms.

Web client components

There are two Web client components:

- Web client controls

Web client components include the Web pages written in markup languages like HTML and XML and the embedded scripts and objects. Scripting languages like VBScript and JScript typically provide enhanced access to objects within the browser interface and HTML document, as well as many generic language features. VBScript is based on the Visual Basic programming language, but it is simpler, and JScript is Microsoft's implementation of JavaScript. Both scripting languages enable you to manipulate controls, such as buttons and scrollbars on Web pages.

- An ActiveX control

An ActiveX control is similar to a Java applet. It can be downloaded and executed by a Web browser. Another similarity with Java applets is that ActiveX controls also run in containers; the Microsoft Internet Explorer, for example, is a container for ActiveX controls. However, unlike Java applets, ActiveX controls can be developed in a variety of languages, including C, C++, C#, Visual Basic, and Java. ActiveX controls also have more access to the Windows operating system, where Java applets are sandboxed. Be aware that users may disable support for ActiveX controls.

Note: ActiveX controls and other COM components are not .NET assemblies; however, wrapped appropriately, interoperability between COM and .NET is possible.

For ActiveX components, the Windows Forms ActiveX Control Importer (Aximp.exe) converts type definitions in a COM type library for an ActiveX control into a Windows Forms control.

The .NET SDK provides Runtime Callable Wrapper (RCW), which wraps the COM components and exposes them for interaction from .NET. The .NET SDK also provides COM Callable Wrapper (CCW), which wraps .NET components so COM clients can identify and interact with them.

Stand-alone client components

.NET offers a console-based client that can run from any Windows command line and take advantage of the full range of Windows APIs and .NET libraries. Console clients can be either rich or thin depending on the use of technologies such as Web References (to call Web Services), .NET Remoting (to call .NET code on a remote server), classic COM, and so on.

.NET also offers a new GUI technology library in Windows Forms. Windows Forms are very similar to the look, feel, and coding style of previous versions of Visual Basic. GUI clients are able to take advantage of the same technology choices as console applications.

4.1.2 Server side components

The server side components in J2EE and .NET include the presentation layer components that execute on the server side and produce the client side HTML pages, as well as the business layer components that represent the core business functionality. Typically, the business layer components communicate with each other over distributed object protocols, such as RMI/IIOP (in case of J2EE) and .NET Remoting (in case of .NET).

J2EE components

In J2EE, the presentation layer components include Java Server Pages (JSPs) and Java Servlets that support invocations from a Web browser over HTTP. These run inside a Java enabled server or application server known as the Web Container according to the J2EE specification. The business layer components include Enterprise Java Beans (EJBs), which run inside an EJB container. Both the presentation layer and business layer components can make use of any plain old Java Objects (POJOs).

Presentation layer components (Web Container Components)

The presentation layer components are:

- **Servlets**

Servlets are server-side software components written in Java, and because of that, they inherit all the benefits of the Java language, including a strong typing system, object-orientation, modularity, portability, and platform independence. They run inside a Java enabled server or application server, such as WebSphere Application Server. Servlets are loaded and executed within the Java Virtual Machine (JVM) of the Web server or application server, in the same way that Applets are loaded and executed within the JVM of the Web client.

The Java Servlet API is a set of Java classes that define the link between a hosting server and servlets and a Web Servlet. Client requests are made to the hosting server, which then invokes the Servlet to service the request. A client of a Servlet-based application does not usually communicate directly with the Servlet, requests for the Servlet's services are sent through the hosting server that invokes the Servlet through the Java Servlet API.

► **JavaServer Pages (JSP)**

JSP technology enables you to easily create Web content that has both static and dynamic components.

A JSP is mostly a XML or HTML document with special embedded tags. It runs in a Web container, and at runtime, the JSP is parsed and compiled into a Java Servlet and may call JavaBeans or Enterprise JavaBeans to perform processing on the server.

The JSP tags enable the developer to insert the properties of a JavaBean object and script elements into a JSP file, providing the ability to display dynamic content within Web pages.

In order for the JSP container to understand and execute a custom JSP tag, the code implementing the tag and information about how to translate the tag and find and invoke the code must be available to the container. Tags for a particular function are normally aggregated into a tag library. In effect, a tag library is a collection of custom tags. Tag libraries, or taglibs, are normally packaged as JAR files.

JSP technology supports the use of JSTL (Java Server Pages Standard Tag Library), which defines a standard set of tags and is used to optimize implementation.

Business layer components (the EJB container components)

The business layer components are:

► **Enterprise Java Beans (EJB)**

EJBs are server-side components that encapsulate the business logic of an application. EJBs simplify the development of large, distributed applications by providing automatic support for system-level services, such as transactions, security, and database connectivity, allowing the developers to concentrate on developing the business logic.

According to the EJB specification:

“Enterprise JavaBeans is an architecture for component-based distributed computing. Enterprise beans are components of distributed transaction-oriented enterprise applications”.

More details about the specification can be found at:

<http://java.sun.com/products/ejb/>

Basically, the EJB environment can be described as follows: The EJB components run inside the EJB container of an J2EE-compliant application server. An EJB client can access the EJBs from the same Java Virtual Machine (JVM) or from another JVM over remote interfaces.

RMI over IIOP (Internet Inter ORB Protocol) is the protocol used when invoking remote EJBs. More information about RMI over IIOP can be found at the following location:

<http://java.sun.com/products/jdk/rmi/>

More information about the Inter-ORB Protocol (IIOP) may be found at:

<http://www.omg.org>

More information can be found in the IBM Redbook *EJB 2.0 Development with WebSphere Studio Application Developer*, SG24-6819.

There are three types of EJBs:

- Entity beans

Entity beans are modeled to represent business or domain objects. They usually represent data (entities) stored in a database or in any persistent data. Entity beans may employ either container managed persistence (CMP) or bean managed persistence (BMP). BMP provides additional flexibility by allowing the developer to fully manage the persistence of the bean. Any additional complexity involves manually writing the necessary SQL code. The persistence code is generated automatically in the case of CMP. The advantage of using container managed persistence is that the entity bean can be logically independent of the data source in which the data is stored.

- Session beans

A session bean is the simplest form of EJB you can create. Session beans are not persisted to a datastore, but rather, are transient objects that may or may not hold state during a series of client invocations in the context of a single user session. A session bean may, in fact, choose to save or retrieve data directly from a database or some other persistence mechanism (although the state of the bean itself is not saved). There are two types of session beans: stateful and stateless. The first type is dedicated to a single client and maintains conversational state across all the methods of the bean. The latter type can be shared across multiple clients, so any information kept in instance variables should not be visible to a client.

- Message-driven beans

These are similar to session beans. Message-driven beans (MDBs) may also be modeled to represent tasks. However, a message-driven bean is invoked by the container on the arrival of a JMS message (asynchronous message). They typically represent integration points for other applications that need to work with the EJB. Since they act as the gateway for client applications (and probably do not implement any business logic),

they can also be thought of as belonging to the presentation layer, even though they execute inside the EJB container.

Figure 4-2 shows the J2EE server components.

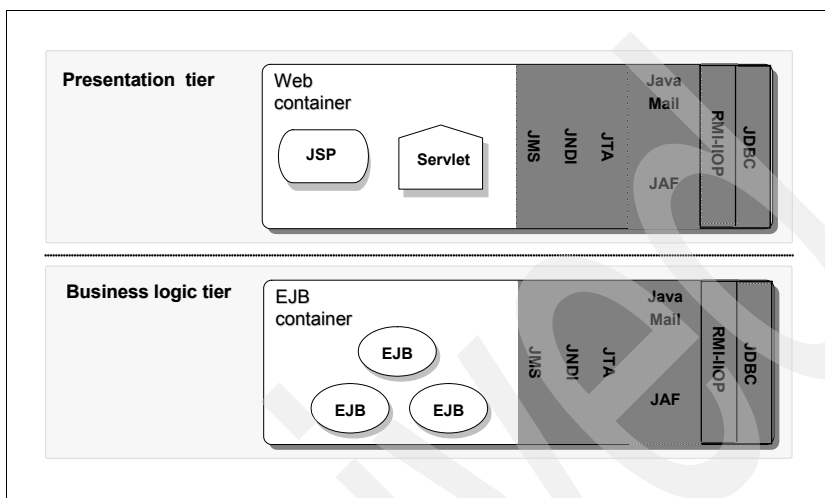


Figure 4-2 J2EE server components

.NET components

The basic component in the .NET framework, which can be reused, is the assembly. An assembly is a collection of files, typically .dll files and any other files that are related to the assembly, such as resource files. The assembly manifest contains metadata relating to version information, security attributes, and external code references. It also contains information about how the pieces in the assembly relate to each other. The assembly manifest, therefore, constructs a logical DLL around the assembly elements.

In .NET, the presentation layer components include ASP.NET running inside Microsoft's Internet Information Services (IIS), and the business layer includes serviced components that are remotely accessible over .NET Remoting.

Presentation layer components

ASP.NET is the main presentation layer component in .NET. Beyond a means of dynamically building Web pages, ASP.NET is a programming framework for building highly scalable Web applications. Normally, ASP.NET applications run on Internet Information Services (IIS), but they can also run on other servers such as the Apache 2.0-based Covalent Enterprise Ready Server. ASP.NET provides enhanced functionality, such as code-behind and event driven Web

controls. One can create ASP.NET Web applications in any language that the .NET Framework CLR supports.

Business layer components

.NET does not have the exact equivalent of server side J2EE components like EJBs that are managed by the container or the server itself. All the services that are provided by a server (such security, messaging, transactions, and logging) are provided by the Windows operating system itself and can be used by any class in the .NET framework. All these services are distributed across various parts of the Windows operating system and each one has its own interface.

.NET Remoting

In the .NET Framework, Microsoft developed a new, .NET specific technology for remote method invocation called .NET Remoting. Simply put, Remoting allows remote access to objects designed to run on the .NET Framework.

Remote objects may be hosted by Internet Information Services, a Windows .NET-based service, or by a stand-alone application. Flexibility comes from the layered component based approach of the Remoting architecture. Figure 4-3 on page 158 shows a high-level overview of .NET remoting. The components in .NET Remoting include:

- ▶ **Formatter**

The formatter takes a remote request and formats it into something that can be sent over the network. The .NET Framework contains a binary formatter and a SOAP formatter. Custom formatters may be added.

- ▶ **Channel**

A remoting channel creates the physical channel between the remoting client and server. The .NET Framework provides two channels: a TCP channel and an HTTP channel. The TCP channel provides a straight socket connection, while the HTTP channel uses the HTTP protocol to send and receive messages. These channels build the appropriate headers and package the data to be sent. The channel interface is extendable. Therefore, custom channels may be created to allow remoting over protocols, such as IIOP or FTP.

- ▶ **Transport**

The transport performs the actual sending and receiving of data. Transports are tied directly to channels and currently cannot be interchanged.

Figure 4-3 gives a high-level view of .NET Remoting.

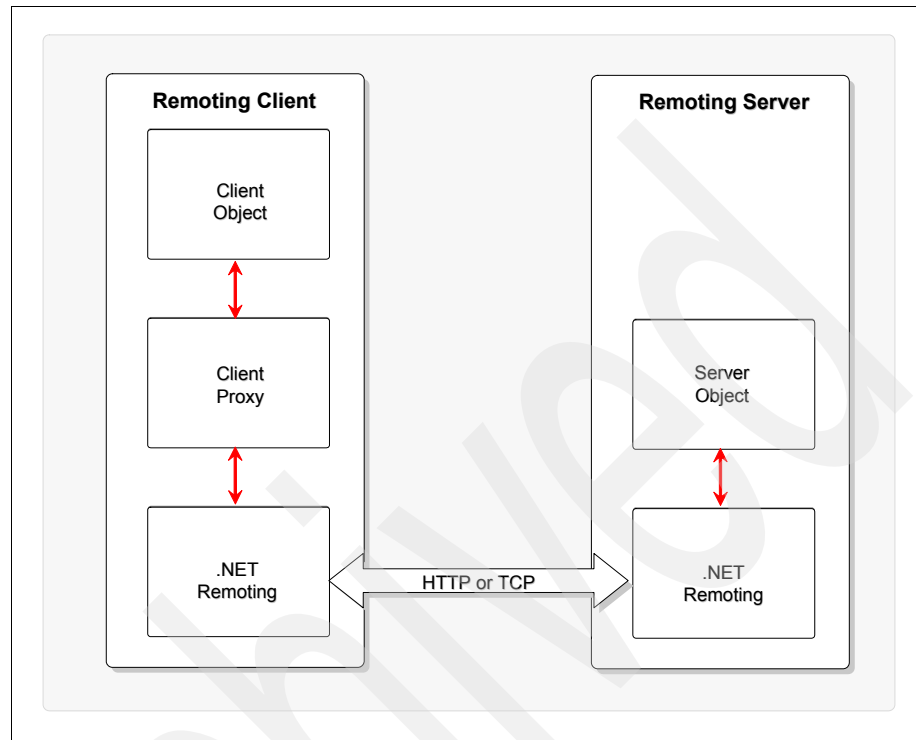


Figure 4-3 High-level view of .NET Remoting

Remoting activation

Activation is making an object available for use by instantiating the object. There are two distinct types of activation for .NET Remote objects. The type of activation scheme you choose will depend on the type of application you are building. The two types of activation are:

- ▶ **Client activation**

In client activation, the remote object is instantiated immediately when the proxy is created on the client. Client activation is used in situations where the life of the object needs to be controlled by the client.

- ▶ **Server activation**

Server activation means that remote object instantiation is controlled by the server. There are two types of server activation: singleton and single call. When a remote object is created as a singleton, a single instance of the object is created for all clients of the object. If the server object is created as a single call object, each time the object is accessed remotely results in a new

server object servicing the call. A singleton object is effective in maintaining state across multiple clients at the expense of requiring the object to be thread safe. In contrast, a single call object need not be thread safe but cannot maintain state. For more information, refer to:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconnetremotingoverview.asp>

4.2 Introduction to component interoperability

The different components in the Java and .NET environments conform to different component object models. However, they all share basic traits, such as the encapsulation of functionality with clearly defined interfaces for accessing the functionality. The runtime environments support common features that include the object life cycle management from the object creation (both local and remote) to automatic garbage collection of objects when all references are freed, and the passing of parameters either by reference or by value (see Figure 4-4).

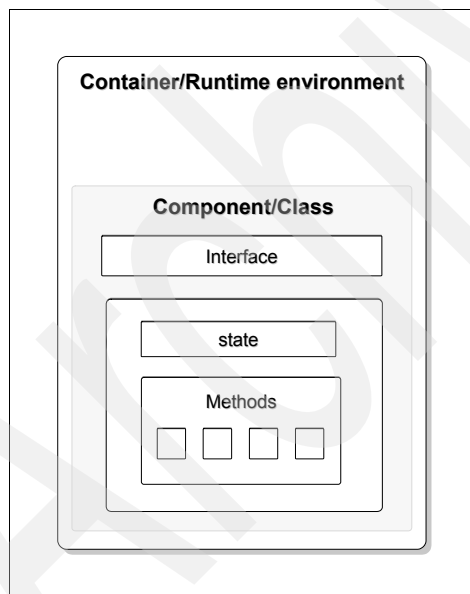


Figure 4-4 Component

Our definition of component interoperability between J2EE and .NET components is essentially about being able to have J2EE components in the Common Language Runtime as though they were a .NET assembly, be able to invoke methods in the interface, and pass parameters and return appropriately typed data from .NET classes. Similarly, you will have .NET assemblies running in J2EE component containers and Java Runtime Environment, with Java classes invoking methods and passing data seamlessly with the .NET classes (see Figure 4-5).

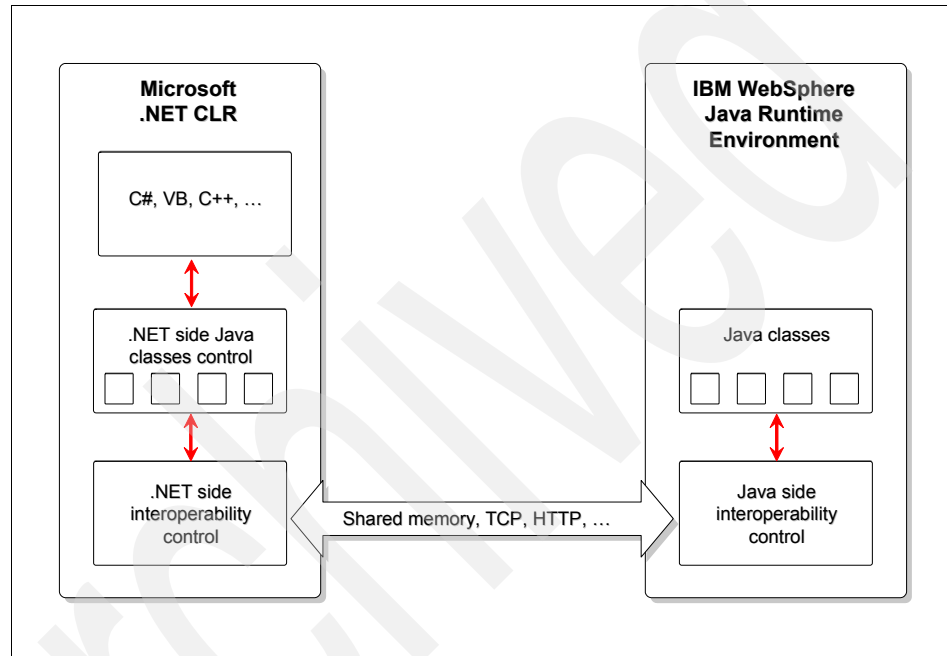


Figure 4-5 Java components interoperating in .NET

For .NET interoperability with Java classes, you in effect need to create and manage the life cycle of proxies to the Java objects in the .NET Common Language Runtime, including garbage collection on the proxies when the Java objects are garbage collected. The .NET side will create the appropriate data types, including collection types that are passed as parameters and return values to the Java classes. Similarly, for Java interoperability with .NET classes, proxies are created and managed in the Java Runtime Environment. The .NET and Java runtime environments can run in the same runtime environment or different environments on the same machine, or on different machines across the network.

The different approaches to component level interoperability include porting, cross compilation, remote procedure call, bridging, and messaging. The three

most practical and widely used approaches are remote procedure call, bridging, and messaging. Although the approaches are different, each provides support that map to the application interoperability stack. A Windows Forms, console client or an Active X component in a .NET client can access a Servlet or a JSP deployed in WebSphere.. Similarly, a Java Applet could access a .NET serviced component or ASP.NET. The application interface will provide support for synchronous and asynchronous interactions depending on the application design. Data is mapped to the appropriate type and passed between the components using standard communication protocol and transport (see Figure 4-6).

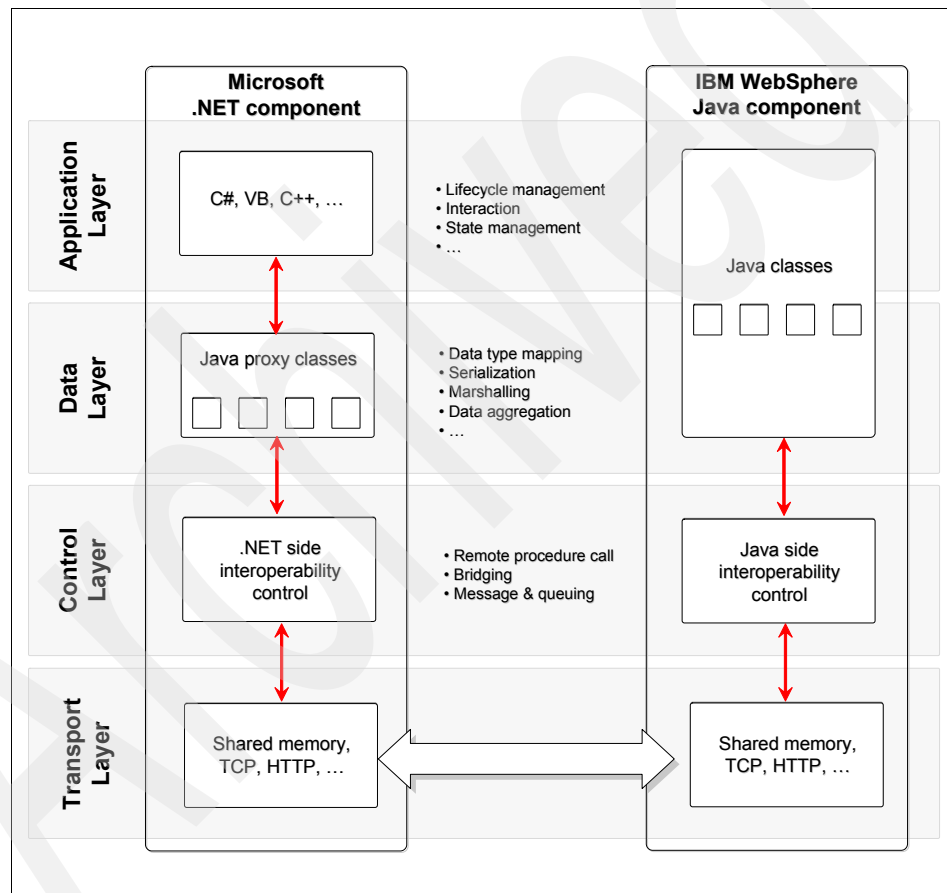


Figure 4-6 Component interoperability overview

4.3 Why choose component interoperability?

Interoperability is one of the main motivations behind the evolution of Web Services. Web Services provide a nice loosely coupled message oriented interoperability. It provides the required set of technologies and standards that are needed to make any two components interact with each other using standard transport protocols and message formats that both components can understand.

However, Web Services is not always the ideal option for interoperability. There are circumstances where component level interoperability is the most viable option.

4.3.1 Interface

Web Services use of text based XML protocol for its service interface results in intrinsic execution overhead. The processing overhead make use of Web Services appropriate for “chunky” interfaces. On the other hand, the entire interface of components are exposed across Java and .NET, resulting in much finer-grained or “chatty” integration. In general, Web Services interoperability are better suited for extending functionality beyond the organizational boundary and for exchange of data across the Internet. The tight integration of component interoperability from activation to garbage collection make it appropriate for intranet solutions.

4.3.2 Transaction

Applications requiring a high transaction rate or high throughput typically avoid Web Services because of the time consuming SOAP/XML message parsing involved. These applications can tolerate tight coupling in return for the high performance they need. For such applications, it makes more sense to interoperate with other components at a level lower than what Web Services provide. By making use of runtime bridges that work at the protocol level and provide conversion between RMI/IIOP and .NET Remoting to support optimized distributed object invocations, component interoperability is able to provide a much higher level of performance than existing Web Services technologies.

4.3.3 Interaction

Web Services are a good candidate for synchronous and asynchronous stateless interactions. Support for stateful (synchronous or asynchronous) interactions is still in its nascent state and not all vendors support out of the box solutions for stateful interactions.

4.3.4 Events

The publish-subscribe and asynchronous notifications support for Web Services is almost non-existent and the various standards are still being defined. Applications that require using publish-subscribe mechanisms cannot use Web Services. Examples of these kind of applications include stock portfolio applications that can subscribe to a set of stock symbols and get notifications when the price for a particular stock symbol changes. Usually, these applications rely on a reliable messaging transport like WebSphere MQ to get these notifications. Component interoperability can make use of different messaging APIs for .NET and J2EE and provide elegant solutions for such applications where publishers and subscribers reside on different platforms.

Designing component interoperability

This chapter provides an overview of the different aspects of component interoperability that you need to consider as you design an interoperability solution. We make use of the application interoperability stack as a guide for identifying the considerations.

This chapter contains the following:

- ▶ Application considerations
- ▶ Data considerations
- ▶ Control considerations
- ▶ Transport considerations

5.1 Application considerations

One of the major decisions you have to make with regard to component interoperability between WebSphere and .NET is the control technology for realizing interoperability. We identified five different approaches for component interoperability in 1.3, “Approaches for interoperability” on page 25. However, only three of these approaches are widely adopted. The choice of any one of these approaches does impact not only your design but also the considerations. Enumerating all considerations for the different application interoperability layers is beyond the scope of this redbook. We will identify considerations that you can apply to your selected interoperability approach to help you design your WebSphere and .NET component interoperability solution.

The application considerations are concerned with such functions as application interaction, state management, and life cycle management, which are part of the application layer of the application interoperability stack (see 1.4.1, “Application interoperability stack” on page 27). The programming language and actual method calls to implement these functions vary based on the approach and the product used for realizing the approach.

5.1.1 Interaction and state management

The potential interaction dynamics between WebSphere’s Java and J2EE components and Microsoft’s components (.NET assemblies, COM, and ActiveX) include the following:

- ▶ Stateful synchronous interaction
- ▶ Stateful asynchronous interaction
- ▶ Stateless synchronous interaction
- ▶ Stateless asynchronous interaction

Stateful synchronous interaction

Stateful interaction: The called code (the service) holds information (conversational state) on behalf of the calling code (the client) across multiple method or function invocations.

Synchronous interaction: The calling code (the client) waits for (blocks) the called code (the service) to complete its operation before continuing with any further processing itself.

Stateful synchronous interaction: The interaction between the calling code (the client) and the called code (the service) is both stateful and synchronous.

Example 5-1 illustrates a pseudo interface definition that implies a stateful interaction semantic between the client code that drives the interface and the service code that implements the interface.

Example 5-1 Pseudo interface definition with a stateful semantic

```
interface iCalculator1
{
    void setArg1([in]float arg1);
    void setArg2([in]float arg2);
    [retval]float add();
    ...
}
```

Example 5-2 illustrates pseudo client code implementation driving the iCalculator1 interface from Example 5-1.

Example 5-2 Pseudo client code to invoke the interface from Example 5-1

```
...
float arg1 = 1;
float arg2 = 2;
float result = 0;
iCalculator1 objCalculator = new calculator();
objCalculator.setArg1(arg1);
objCalculator.setArg2(arg2);
result = objCalculator.add();
delete objCalculator;
assert(3 == result);
...
```

The client code makes three method invocations of the service implementation (setArg1(...), setArg2(...) and add()). In order for the service implementation to correctly service the client's request, it must maintain state on behalf of the client across all three invocations. This implies that the service implementation must be associated with a single client (or must be able to distinguish between clients) until the client no longer requires its services (the implication is that the service implementation is stateful).

Stateless synchronous interaction

Stateless interaction: The called code (the service) does *not* hold information (conversational state) on behalf of the calling code (the client) across multiple method or function invocations.

Stateless synchronous interaction: The interaction between the calling code (the client) and the called code (the service) is both Stateless and Synchronous.

Example 5-3 illustrates a pseudo interface definition that implies a stateless interaction semantic between the client code that drives the interface and the service code that implements the interface. This interface differs from the interface in Example 5-1 on page 167, because all the arguments to the `add()` method are passed as input parameters to the `add()` method itself. As a consequence, the service implementation can service the client's request in a single method invocation. Consequently, the service implementation has no requirement to maintain any client specific state beyond the scope of a single service request. Example 5-4 illustrates the pseudo client code implementation driving the `iCalculator2` interface from Example 5-3.

Example 5-3 Pseudo interface definition with a stateless semantic

```
interface iCalculator2
{
    [retval]float add([in]float arg1, [in]float arg2);
    ...
}
```

Example 5-4 shows the pseudo client code to invoke the interface in Example 5-3.

Example 5-4 Pseudo client code to invoke the interface from Example 5-3

```
...
float arg1 = 1;
float arg2 = 2;
float result = 0;
iCalculator2 objCalculator = new calculator();
result = objCalculator.add(arg1, arg2);
delete objCalculator;
assert(3 == result);
...
```

Example 5-5 on page 169 illustrates an alternative pseudo interface definition. This definition also implies a stateless interaction semantic between the client code that drives the interface and the service code that implements the interface. This differs from Example 5-3 in that it passes a *state* object instance (of type `iCalculator3Args`) as the single input parameter to the `add()` method. Example 5-6 on page 169 illustrates pseudo client code implementation driving the `iCalculator3` interface from Example 5-5 on page 169.

Example 5-5 An alternative pseudo interface definition with a stateless semantic

```
interface iCalculator3Args
{
    void setArg([in]float arg);
}

interface iCalculator3
{
    [retval]float add([in]iCalculator3Args args);
    ...
}
```

Example 5-6 shows the pseudo client code to invoke the interface in Example 5-5.

Example 5-6 Pseudo client code to invoke the interface from Example 5-5

```
...
float arg1 = 1;
float arg2 = 2;
float result = 0;
iCalculator3Args objCalculatorArgs = new calculator3Args
objCalculatorArgs.setArg(arg1);
objCalculatorArgs.setArg(arg2);
iCalculator3 objCalculator = new calculator();
result = objCalculator.add(objCalculatorArgs);
delete objCalculator;
assert(3 == result);
...
```

We have assumed that the implementation of interface iCalculator3Args in Example 5-7 on page 170 is local to the client; do not assume this in real scenarios.

These interfaces *imply* a stateless interaction, and note that stateless is really a classification of an implementation, not a classification of an interface. Do not assume an implementation is stateless just because its interface implies that it is.

Stateless asynchronous interaction

Asynchronous interaction: The calling code (the client) does *not* wait for (block) the called code (the service) to complete its operation before continuing with any further processing.

Stateless asynchronous interaction: The interaction between the calling code (the client) and the called code (the service) is both Stateless and Asynchronous.

Stateful interactions are generally synchronous (this is not always the case, but frequently so). That is, if client code needs to make several invocations of a service, and state needs to be maintained by the service implementation on behalf of the calling client between these invocations, and the order (sequence) of these invocations is important, then the client is generally synchronized with the service. This normally means that the client invokes the service, then waits (blocks) until the service responds and returns logical flow back to the client before the client invokes the service again.

Because stateless interaction (generally) communicates all the states for a given task in one invocation, stateless service implementations (generally) have more potential to be implemented as asynchronous (non-blocking) implementations than stateful service implementations.

Example 5-7 illustrates a pseudo interface definition, which implies a stateless asynchronous interaction semantic between the client code that drives the interface and the service code that implements the interface. The implication is that the `delegateTask()` method takes some abstracted state (a message or document), acknowledges receipt of the message (returns logical flow to the client), and then processes the message in its own time.

Example 5-7 Pseudo interface definition(s) with a stateless asynchronous semantic

```
interface iBusinessProcess
{
    void delegateTask([in]iMessage arg1);
    ...
}
```

Example 5-8 shows the pseudo client code to invoke the interface in Example 5-7.

Example 5-8 Pseudo client code to invoke the interface in Example 5-7

```
...
iMessage objMessage = new Trade('sell Microsoft stock, buy IBM stock');
iBusinessProcess objTrader = TradeProcessor();
objTrader.delegateTask(objMessage);
...
```

A stateless asynchronous interaction does not necessarily imply messaging middleware (such as WebSphere MQ). For example, a service proxy or a service façade could deliver an asynchronous solution using threads.

Asynchronous processing is useful when a consumer requires independent services from more than one service provider, especially when the elapsed time for processing for any particular service provider is unpredictable or simply takes too long. Since information provided back to the consumer at the time of service initiation is scant, asynchronous operations typically are not used for real-time queries, but are often used to initiate a request or action on the part of a service provider.

Because stateless interaction communicates all the states for a given task in one invocation, stateless service implementations have a greater potential to be implemented asynchronously than do stateful interactions. Figure 5-1 illustrates an asynchronous request between a consumer and service provider.

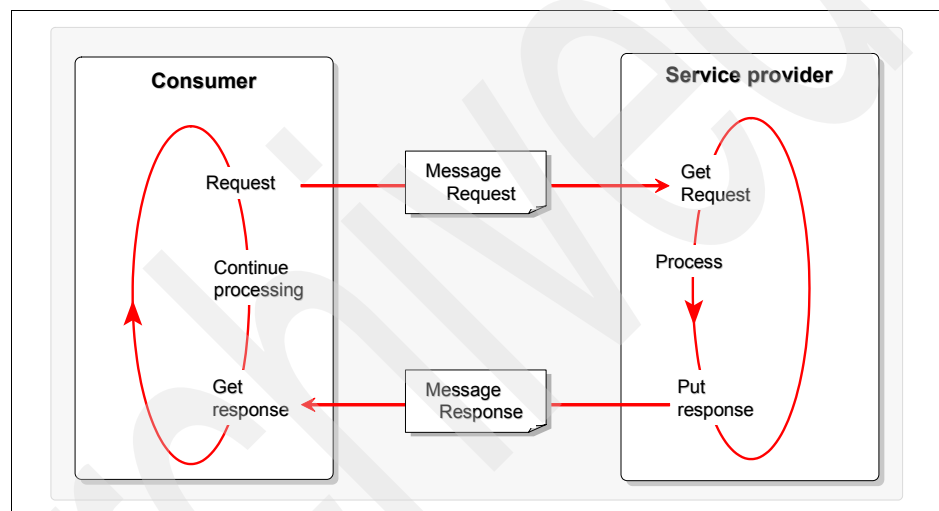


Figure 5-1 Asynchronous interaction between consumer and service provider

The service provider is designed to consume incoming requests and provide responses. The consumer makes requests and continues processing until the requests are satisfied and then retrieves the responses. Although we show this entire activity taking place within a single consumer execution, this is not a requirement. The only requirement to be considered asynchronous is that the consumer not block execution between the request and the response.

Although we represent the information being passed between the artifacts here as messages, a stateless asynchronous interaction does not necessarily imply messaging middleware. A service proxy or a service façade can also deliver an asynchronous solution using threads. In the same way, the use of messaging middleware does not automatically imply asynchronous invocation either. Many applications that use messaging middleware are written using a synchronous request/response paradigm.

Stateful asynchronous interaction

Stateful asynchronous interaction: Interaction between the calling code (the client) and the called code (the service) is both stateless and asynchronous.

Stateful interactions are usually synchronous. That is, if a consumer needs to make several invocations of the service, and the state needs to be maintained by the service implementation between these invocations, and the order (sequence) of these invocations is important, then the consumer must be synchronized with the service. This normally means that the consumer invokes the service, then waits (blocks) until the service responds before invoking the service again.

Consider, then, how the management of state is simply the storing of state information. Under the asynchronous paradigm, the state can be stored by the service provider in the same way, but this often makes little sense because asynchronous service providers are usually engineered so that they do not have to remain running. A better way to store state information for asynchronous interaction is within the messages passed between the consumer and provider.

Although there are limitations to the type and amount of state information it is possible to store, this makes for some interesting and useful new paradigms regarding state where asynchronous operations occur, as represented in Figure 5-2 on page 173 and Figure 5-3 on page 174.

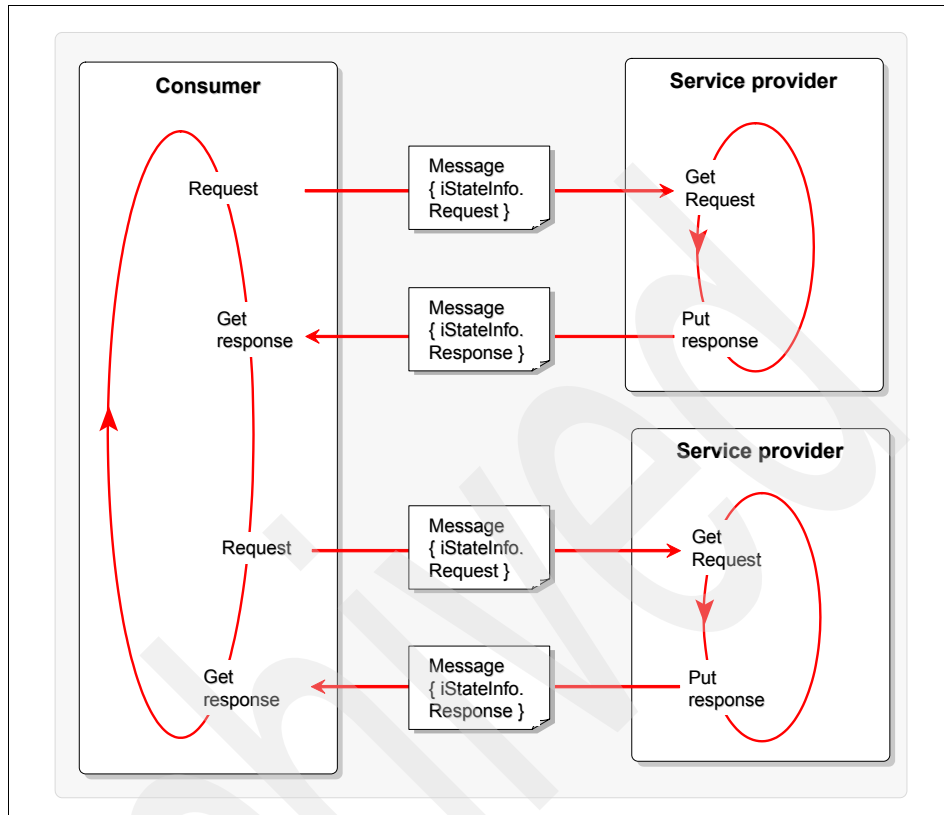


Figure 5-2 Stateful interaction using multiple service providers

In Figure 5-2, we see that the messages contain a mutually agreed-upon `iStateInfo` object containing state information. Because the message maintains state, the service providers are capable of performing state-dependent operations while the interaction remains asynchronous in nature.

In Figure 5-3, we see that we can easily aggregate services as well using the asynchronous paradigm while maintaining state information. Properly engineered messages between the artifacts can help maintain the loosely-coupled nature of these interactions.

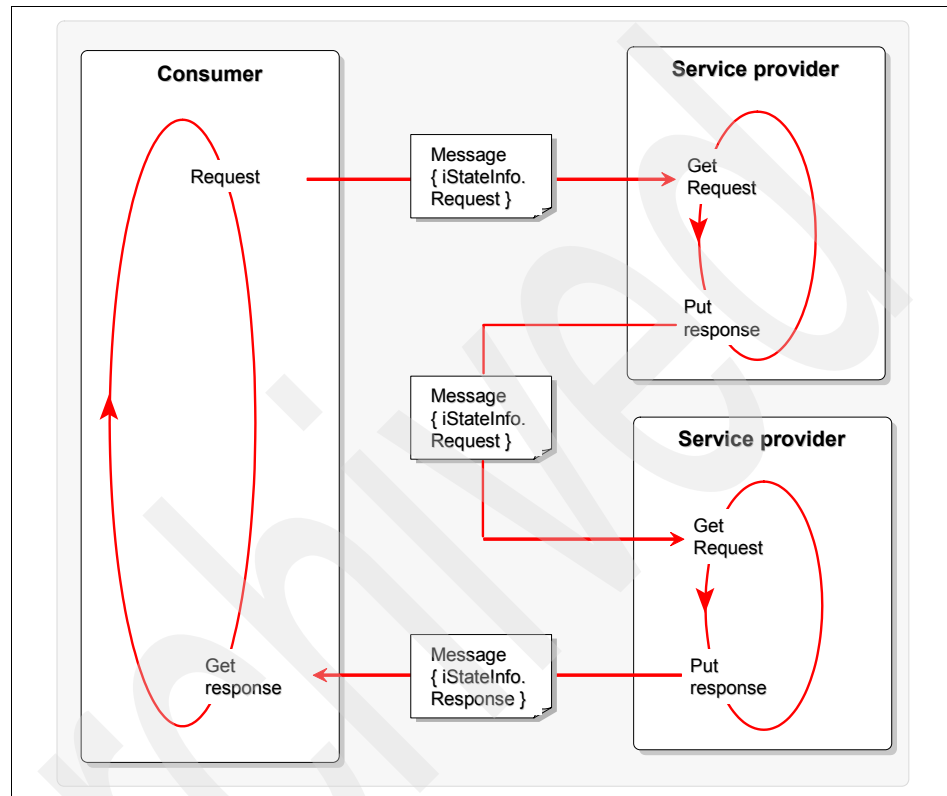


Figure 5-3 Stateful asynchronous service aggregation

Finally, asynchronous processing allows the introduction of other powerful middleware tools such as WebSphere Integrator. Integrator provides sophisticated message transformation and routing capability, allowing the creation of highly-sophisticated message flows between consumers and providers.

5.1.2 Message format

The request message can be formatted following a simple format that both the sender and the receiver can easily encode and decode. This could simply mean sending the ASCII data delimited by some special characters. The text data is then parsed by the receiver to retrieve the meaningful data. For example, to add two integer numbers (for example 12 and 25), the data can be sent as:

```
"12$25$Add"
```

where \$ is a delimiter.

The above format has severe restrictions. It requires detailed understanding of the order and meaning of the data arguments by both the sender and the receiver. However, it is also a very compact format that requires very little extra characters. For simple applications with only one or two external communication interfaces, this is a usable format.

The above format can be enhanced to remove the restriction on explicit ordering of data values. Thus, the data can be sent as:

```
"Input1=12$Input2=25$Command=Add"
```

where \$ is a delimiter.

Here the data values need not be in any given order, as each data value has a name associated with it. However, it still requires both the sender and the receiver to know the names of the arguments to expect in the message.

Any large scale, enterprise application cannot work with the above formats because of their obvious limitations. Extended Markup Language (XML) is the industry standard text format for exchange of information between communicating parties. The structure, the content, and to some extent, semantics of XML documents are defined using XML schemas, which can either be exchanged or made available via a repository for all communicating parties. Using XML, the above data values can be represented as:

```
<?xml version="1.0" encoding="utf-8"?>
<CalculatorInputElement xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://tempuri.org/XMLSchema.xsd">
  <Input1>28</Input1>
  <Input2>3</Input2>
  <Command>ADD</Command>
</CalculatorInputElement>
```

Note that the parsing overhead for the above text is much more than with the previous simpler formats. Fortunately, there are a number of XML parsers available that can be applied here. For example, the Java Architecture for XML Binding (JAXB) provide an API and tools that automate the mapping between XML documents and Java objects. For more information about JAXB, refer to the following URL:

<http://java.sun.com/webservices/jaxb/>

5.1.3 Life cycle management

Usually, you do not need to concern yourself with when an object is created. You are simply interested in making calls to methods on the object. When dealing with interoperability, you not only need to know when the new object is created, but you also need to know how your interoperability control product creates and initializes the new object. You also need to know where it is activated; here you need to consider if the WebSphere and .NET components are running in the same runtime environment, same machine, or different machines across the network.

When an object is no longer accessible, the object is automatically garbage collected. In an interoperability scenario, the proxy object is usually automatically garbage collected as well. However, there are situations where manual control is necessary. Take, for example, a Java object with a .NET proxy; the triggering of garbage collection on the Java side and the .NET side does not happen simultaneously. Making a call through the .NET proxy object that has yet to be garbage collected whose Java object is already garbage collected will result in an exception condition. So finer control is necessary to ensure control of the garbage collection on both the interoperating object and its proxy.

Note: Consult the product documentation for the interoperability approach for details for configurations and activation of objects, and the garbage collection of objects and their proxies.

5.2 Data considerations

Perhaps the most important aspect of interoperability is the exchange of data between the interoperating components. Data is constantly exchanged between interoperating components. The exchange must match, syntactically and semantically, the method interfaces.

One of the functions that is handled in the data layer of the application interoperability stack is the mapping of types between the interoperating components. There are differences in the Java and .NET types. These

differences are some of the data considerations that you need to take into account for your solution design. The other functions include serialization and deserialization, marshalling and unmarshalling, and data performance functions, such as aggregation.

5.2.1 Type mapping

Not all types in Java map directly to .NET types. For example, .NET has unsigned integral (byte, ushort, uint, and ulong) and decimal types that do not exist in Java (see Table 5-1).

Table 5-1 Java and .NET direct primitive type mapping

Java	.NET
int	int (System.Int32)
short	short (System.Int16)
long	long (System.Int64)
float	float (System.Single)
double	double (System.Double)
char	char (System.Char)
boolean	bool (System.Boolean)

Both Java and .NET support a rich library of collection types, although the collections appear to be similar and even when the names are practically the same (java.util.Hashtable in Java and System.Collections.Hashtable in .NET), the implementations are different and they are sources of interoperability issues. In Java, for example, you have java.util.Hashtable, Vectors, Hashmap, Set, and ArrayList, and in .NET there are System.Collections.Hashtable, SortedLists, Queue, Stack, and ArrayList.

Note: The mapping and translation of Java collection classes to .NET and vice versa and the details of how they are handled when passed as parameters is beyond the scope of this redbook. It is dependent on the approach and product you use for your solution. Please consult the product documentation for more information.

5.2.2 Pass by value or reference

When a parameter is passed by value, the called code (the service) receives a copy of the argument. Any changes made to the value of a pass by value argument by the called code do not affect the value of the caller's original copy, and are not visible to the calling code.

When a parameter is passed by reference, the called code (the service) receives a reference to the caller's instance of the argument. Any changes made to the value of a pass by reference argument by the called code do affect the value of shared argument instance, and are visible to the calling code (the client).

In-only arguments: Pass by value arguments are often referred to as in-only arguments. For example:

```
method([in]arg1); is equal to... method([byValue]arg1);
```

In/out arguments: Pass by reference arguments are often referred to as in/out arguments. For example:

```
foo([in,out]arg1); is equal to... foo([byReference]arg1);
```

The Java language has two categories of types: primitive types and reference types. The .NET type system includes value, reference, and pointer types. Arrays, classes, and interfaces are reference types in Java language, hence `java.util.Date` and `java.util.Calendar` classes are reference types. In .NET, any type that is derived from `System.ValueType` either directly or indirectly is considered value type and types that do not derive from `System.ValueType` are considered reference types.

In .NET, all types (even primitive types) are fundamentally objects. For example, an `int` is actually an alias to the CTS `System.Int32` type. `System.Int32` derives from `System.ValueType`, so it has *pass by value* behavior.

Boxing is a .NET technique that allows a *pass by value* type to be treated as a *pass by reference* type (see Example 5-9).

Example 5-9 Boxing

```
//--- i is by default a 'pass by value' object ---  
int i = 99;  
//--- boxed_i is a 'pass by reference' copy of i ---  
object boxed_i = i;  
//--- unboxed_i is a 'pass by value' copy of boxed_i ---  
int unboxed_i = (int)boxed_i;
```

Java and all .NET programming languages support the argument by value paradigm in a distributed environment. So, in a proxy pattern class level

interoperability, a call by reference in Java that passes a reference data type such as an array becomes a pass by value in the .NET proxy, creating a corresponding type in the .NET proxy.

While this has performance benefit because pass by reference results in a chatty interface requiring a round trip to the Java side for each access to the array element, it could result in unexpected behaviors. You should be fully aware when you are assigning values to the proxy array, and when passing the proxy array by reference.

Note: The details of how call by value or reference is handled and what the solution implementer must do depends to a large extent on the approach and product you choose for class level interoperability. Consult the product documentation for more detail.

5.2.3 Callbacks

.NET supports callbacks through the use of delegates and events. Delegates are commonly defined as Type Safe Pointers. A delegate hold references to functions that can be defined and used at runtime. Java classes can participate in .NET callback functionality. The implementation and registration of Java classes as delegates and event handlers in Java to .NET interoperability will depend on the approach and product you choose for class level interoperability.

Note: A full discussion on delegates and events and the implementation in Java to .NET interoperability is beyond the scope of this redbook. Please refer to Microsoft .NET documentation for discussion on delegation and events and to the product documentation you are using for your solution to the implementation of delegates and events implementation in Java to .NET interoperability.

5.3 Control considerations

To a large extent, the choice of control approach and the product of choice for implementing your solution depends on a number of factors. In this section, we will discuss these factors. We will also like to note that this section is not intended to be prescriptive, but it is intended to be indicative of considerations leading to solutions for different WebSphere and .NET component interoperability scenarios.

5.3.1 Factors

The following are some of the factors that influence the choice of solution for a given WebSphere and .NET component interoperability scenario.

Solution environment

The solution environment:

- ▶ What is the existing infrastructure for the targeted deployment environment? What middleware is already deployed?
- ▶ What is the strategic and long term technical direction of the solution environment? What weight does the direction constraints have on the choice of control approach and product?
- ▶ What are the qualities of service requirements? How do the different solution approaches compare to the stated performance, security, scalability (and so on) requirements?

Project plan

The project plan:

- ▶ What is the available time for the project? Are there time constraints that will require a quick solution?
- ▶ What is the skills and resource availability? Are there design and development resources with skills for any of the approaches?

Financial resources

The financial resources:

- ▶ Is there available budget for the initial implementation?
- ▶ How does the deployment and maintenance costs for the different approaches fit into a projected budget?

Technical solution

The technical solution:

- ▶ What is the optimal solution for the environment? Is it the case that more than one approach is applicable in the scenario?
- ▶ Technical solution maturity

Your ideal (strategic) technical solution may still be maturing. It may be deficient in a required characteristic, but on the cusp of delivering all your requirements. In this scenario, if the interoperability integration layer is well designed, it will normally be possible to totally abstract the integration technology from both the client implementation code and the service implementation code. This implies that, if it is appropriate, you should be able

to choose a tactical short term solution, then, with a minimum of disruption, substitute it for the ideal (strategic) solution at a later date.

The intent of these factors is to help in the solution rationalization. The goal is to arrive at a set of characteristics for mapping to the different component interoperability approaches and products for a given scenario.

5.3.2 Approaches

We identified five different approaches for accomplishing class level interoperability between WebSphere and .NET (1.3, “Approaches for interoperability” on page 25). These approaches include:

- ▶ Porting

You port the entire .NET frame work to Java or vice versa. The effort required for the initial port and for keeping the port up-to-date is quite involved and the task is very tedious. This is reflected in the lack of available products pursuing this approach.

- ▶ Cross compilation

This approach cross compiles Java classes into Microsoft Intermediate Language (MSIL) for execution in the Common Language Runtime (CLR) or .NET languages (C#, Visual Basic, C++, and so on) to Java bytecodes for execution in the Java Runtime Environment. While in theory it is technically feasible to cross compile for MSIL and Java bytecode, the practice is, however, not forth coming.

- ▶ Remote procedure calls

You take advantage of the remote procedure call mechanism supported by both Java and .NET.

- ▶ Bridging

Implementation of proxy pattern where Java proxy runs in .NET CLR, invoking and is invoked by .NET classes and vice versa with .NET proxy in Java.

- ▶ Messaging and queuing

Interoperating using messaging and queuing with client support classes for Java and .NET.

5.3.3 Products

In this section, we identify products that use three of the five approaches we have identified. The three approaches are remote procedure call, bridging, and messaging and queuing. The identified products are simply that and not an endorsement or recommendation, nor is the set of products an exhaustive list of products for realizing component interoperability control.

IBM Interface Tool for Java

The IBM Interface Tool for Java is a tool that allows Java programs to communicate with ActiveX objects. It leverages .NET COM interop features to provide stateful Java proxies to .NET assemblies. It allows integration of ActiveX objects into a Java Environment. Using the Java Native Interface and COM technology, the IBM Interface Tool for Java allows an ActiveX object to be treated just like a Java object (see Figure 5-4).

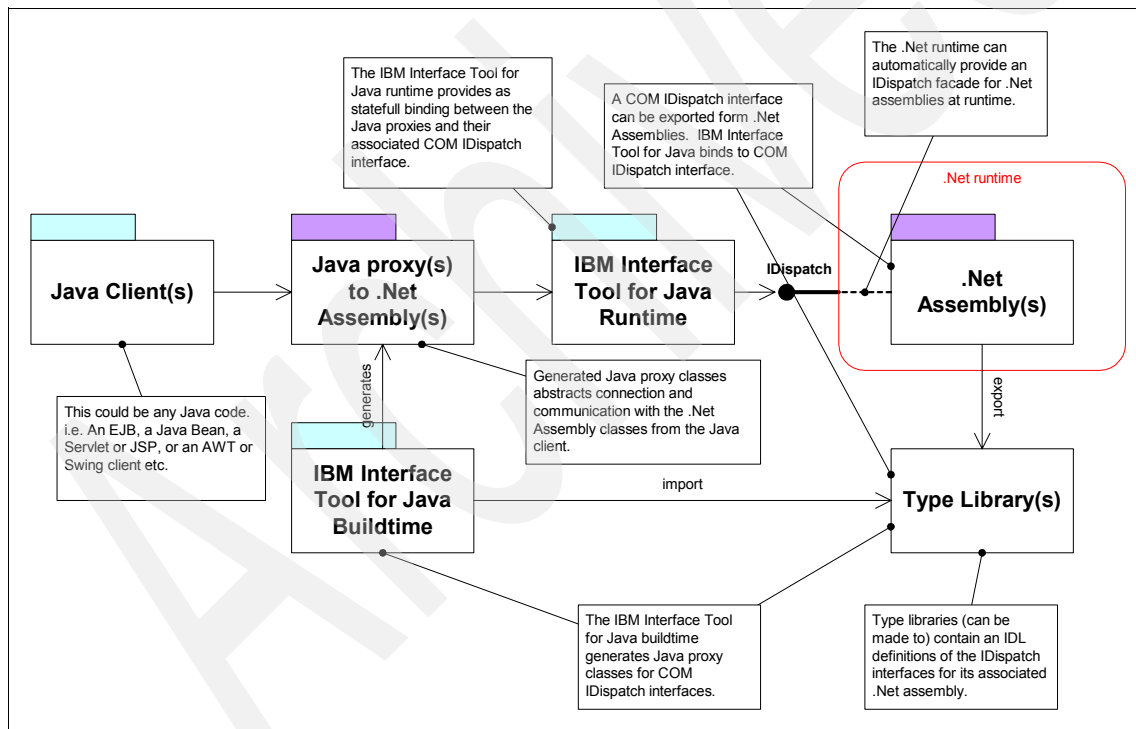


Figure 5-4 IBM Interface Tool for Java overview

Using the IBM Interface Tool for Java requires simply running a proxy generating tool that will create a Java proxy from the Active X control's typelib. These proxies

can then be used to allow a Java program to communicate with the ActiveX object.

Note: The IBM Interface Tool for Java is the technology formerly known as Bridge2Java. See:

<http://www.alphaworks.ibm.com/tech/bridge2java>

The IBM Interface Tool for Java provides in-process binding to .NET assemblies.

As Figure 5-5 shows, the simplest solution candidate loads the IBM Interface Tool for Java runtime, and the assembly (and its CLR instance) into the Java client's JVM process.

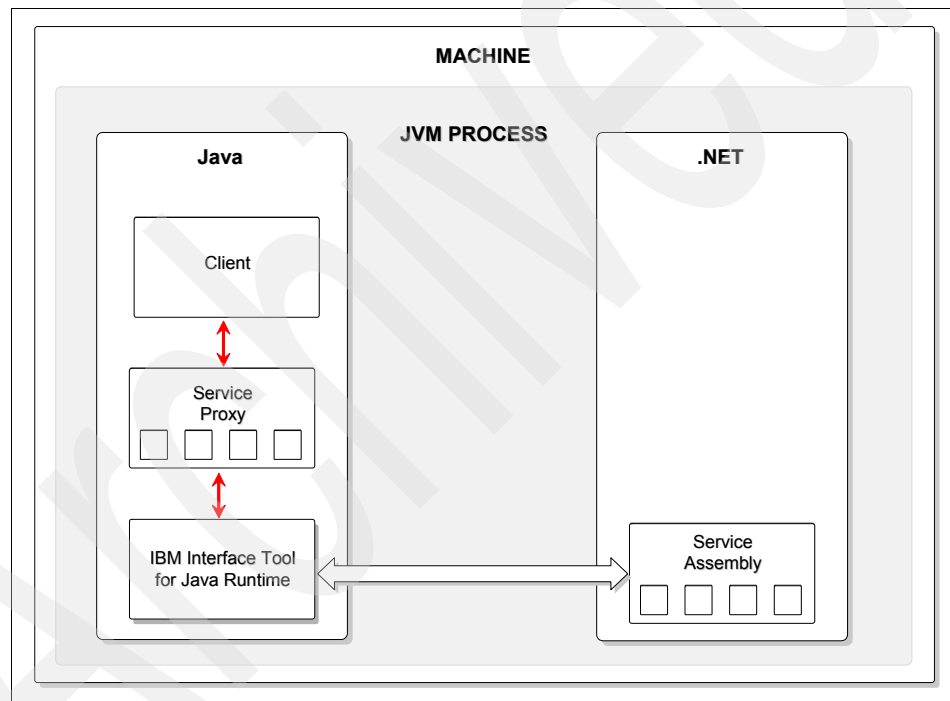


Figure 5-5 IBM Interface Tool for Java: in-process solution scenario

Other configurations are possible with the IBM Interface Tool for Java. The .NET assembly can run on the same machine outside of the JVM process and also on a separate machine. Some sort of transport is required when the .NET assembly is not local to the JVM. We discuss the different configuration and communication mechanisms for component interoperability in 5.4, “Transport considerations” on page 187.

IBM WebSphere ActiveX Bridge

IBM WebSphere ActiveX Bridge provides a COM interface to the WebSphere Client Container. We can leverage .NET COM interop features to provide stateful .NET proxies to WebSphere Services and applications (see Figure 5-6).

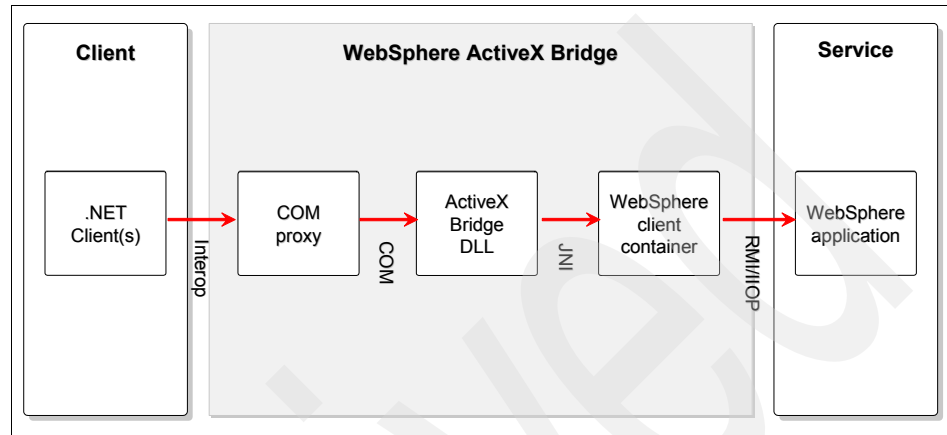


Figure 5-6 IBM WebSphere ActiveX Bridge overview

Figure 5-7 on page 185 illustrates a candidate solution model using IBM WebSphere ActiveX Bridge as the integration solution between .NET client code and an application deployed in WebSphere. This solution model uses a COM service proxy to abstract IBM WebSphere ActiveX Bridge from the .NET client code. The .NET client code binds to the COM service proxy using .NET COM interop. As you can see, with this solution, both a CLR instance and a JVM instance live inside the client process

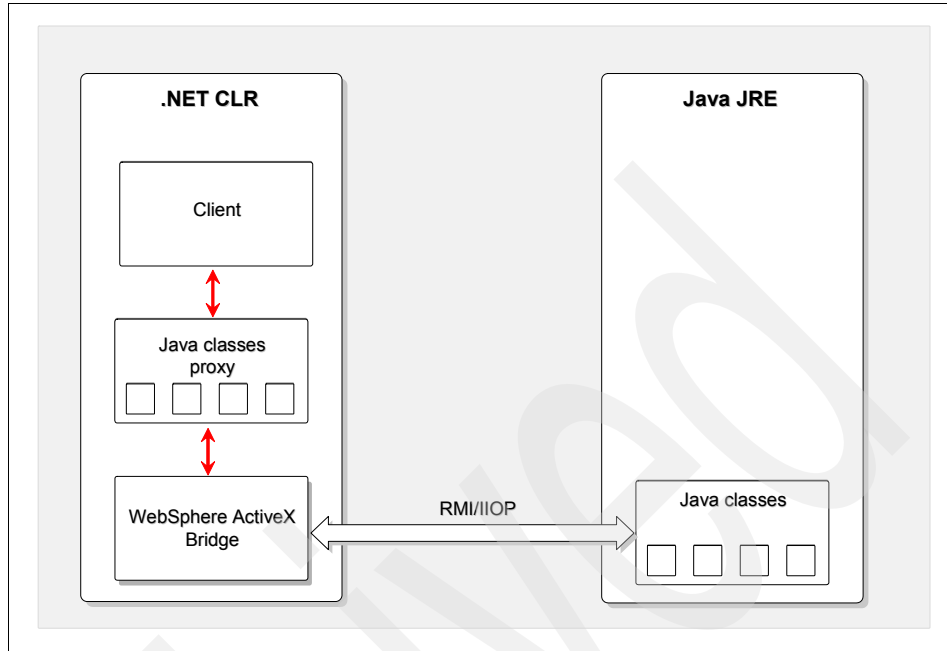


Figure 5-7 IBM WebSphere ActiveX Bridge as the integration solution between .NET client code and an application deployed in WebSphere

Figure 5-8 illustrates an alternative candidate solution model that put the .NET CLR and COM into separate processes. As usual, separating these technologies into different processes can potentially have both benefits and disadvantages. Once again, you may need to trade one feature for another to arrive at a satisfactory solution.

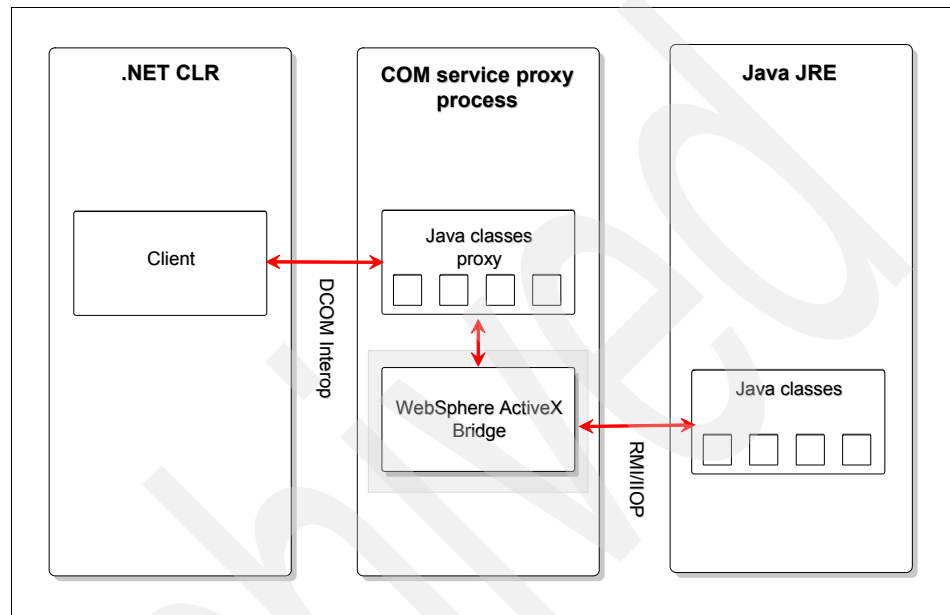


Figure 5-8 Alternative candidate solution model, separating the .NET CLR and COM processes

IIOP.NET

IIOP.NET is a SourceForge.net open source project that provides bidirectional interoperation between .NET, CORBA, and J2EE distributed objects. IIOP.NET implements a CORBA/IIOP remoting channel for the .NET Framework.

For more information about IIOP.NET, visit:

<http://iiop-net.sourceforge.net/>

IIOP.NET is released under the LGPL license:

<http://www.gnu.org/copyleft/lesser.html>

J-Integra

J-Integra is a commercial product from Intrinsic. J-Integra includes a .NET remoting implementation for .NET clients to bridge to J2EE and CORBA implementations over IIOP. For more information about Ja.NET, visit:

<http://j-integra.intrinsyc.com/>

JNBridge

JNBridge is a commercial product from JNBridge. JNBridge includes a .NET remoting implementation for .NET clients to bridge to J2EE and CORBA implementations over IIOP. For more information about JNBridge, visit:

<http://www.jnbridge.com>

Janeva

Janeva is a commercial product from Borland. Janeva includes a .NET remoting implementation for .NET clients to bridge to J2EE and CORBA implementations over IIOP. For more information about Janeva, visit:

<http://www.borland.com/janeva/>

5.4 Transport considerations

Flexibility in the products configuration make various deployment options possible. This ranges from having the two sides running on the same machine in the same process to having the Java and .NET sides running across the network. The transport considerations include the different options for communication between the Java and .NET sides.

5.4.1 Run on the same machine

You have two options when you configure the Java and .NET components to run on the same machine.

Same process

The transport option is the use of shared memory for communication between the Java and .NET sides (see Figure 5-9).

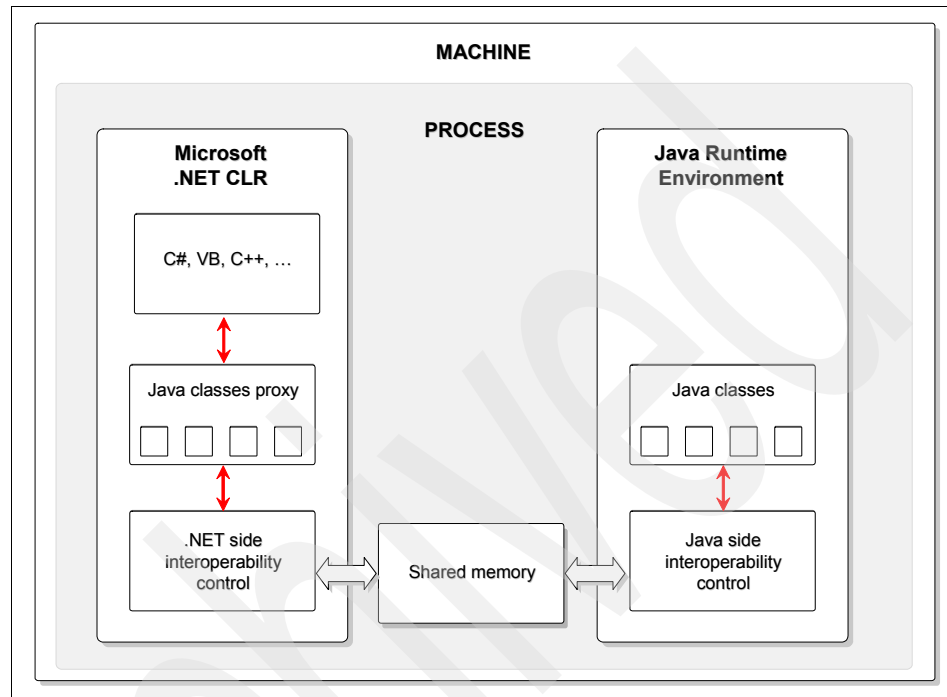


Figure 5-9 Shared memory communication on same machine and process

Different processes

Additional options include socket communication between the two processes and messaging and queuing (see Figure 5-10 on page 189).

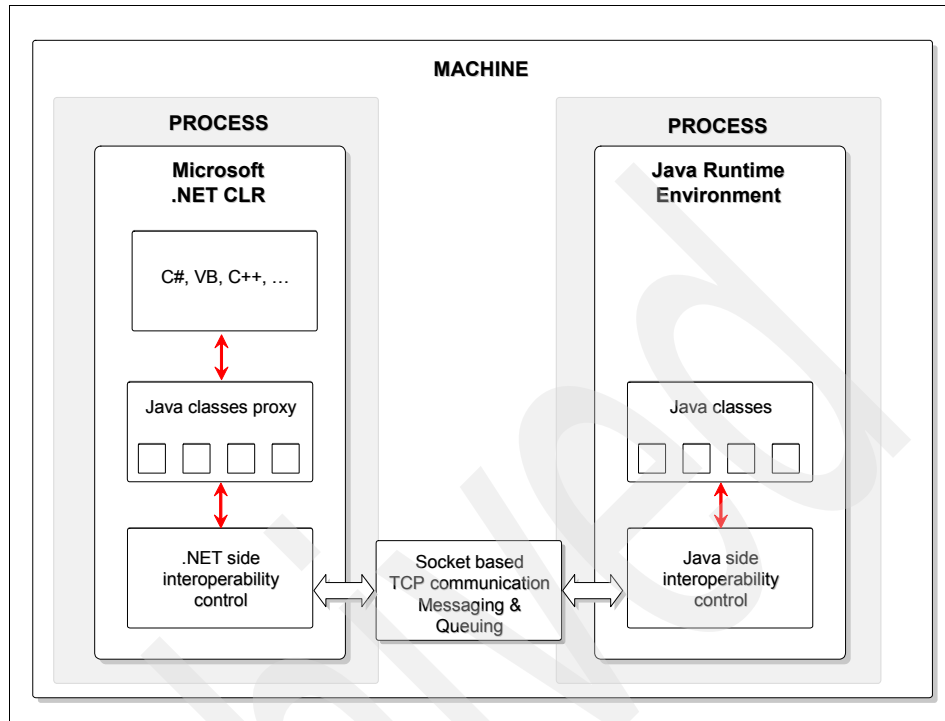


Figure 5-10 Same machine different processes

5.4.2 Different machines

When you configure the Java and .NET side on different machines, you have options ranging from TCP within your intranet to HTTP/SOAP across the Internet. You can also use messaging and queuing for reliable and guaranteed delivery communication when running on different machines (see Figure 5-11).

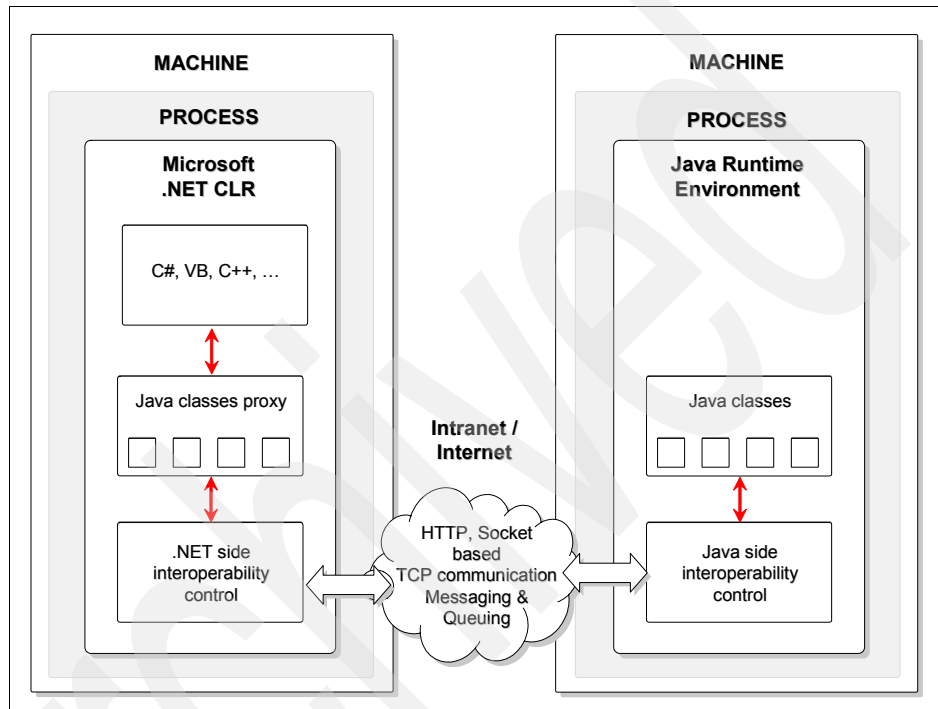


Figure 5-11 Separate machines over the intranet

Component interoperability scenario

This chapter describes a component interoperability scenario. The scenario implements a simple calculator where the interoperating components consist of a .NET Windows Form client application component and a WebSphere calculator service component. The scenario makes use of WebSphere MQ messaging to provide component level interoperability.

This chapter discusses the following:

- ▶ Scenario description
- ▶ Queues installation and configuration
- ▶ Create messaging resources, schemas, and classes
- ▶ Developing the .NET Windows Form client application
- ▶ Developing the WebSphere calculator component
- ▶ Testing the sample application

6.1 Scenario description

The scenario essentially illustrates asynchronous component level interaction between a .NET Windows Form client and a WebSphere EJB calculator component using WebSphere MQ. The scenario makes use of messaging for the component interoperability control and transport.

Note: Use of messaging and WebSphere MQ is just one of the many possible interoperability technologies for this scenario; a component bridging technology could easily be substituted for this component interoperability scenario.

6.1.1 Basic interaction outline

The scenario illustrates component interoperability between a .NET Windows Form based rich client application and a WebSphere Calculator service that is exposed through a J2EE message-driven bean (MDB). The .NET client application acts as a consumer and invokes various methods on the calculator component.

The client component

The basic functions of the client component, the .NET Windows Form application can be summarized, as follows:

- ▶ Accepts input data from the user using a graphical user interface.
- ▶ Creates a formatted request message using the input data.
- ▶ Sends the request message.
- ▶ Receives the response message.
- ▶ Interprets the response message and format the data for display.
- ▶ Displays meaningful data to the user.

The calculator component

The basic functions of the calculator component, the J2EE message-driven bean (MDB), can be summarized as follows:

- ▶ Receives message from the client application component using the message queue.
- ▶ Extracts the requested function and parameters from the message.
- ▶ Executes the calculation function.
- ▶ Formats the response message properly.
- ▶ Sends the response message.

6.1.2 Message format

The request message can be formatted following a simple format that both the sender and the receiver can easily encode and decode. This could simply mean sending the ASCII data delimited by some special characters. The text data is then parsed by the receiver to retrieve the meaningful data. For example, to add two integer numbers (for example 12 and 25), the data can be sent as:

"12\$25\$Add"

where \$ is a delimiter.

The above format has severe restrictions. It requires a detailed understanding of the number, order, and meaning of the data arguments by both the sender and the receiver. However, it is also a very compact format that requires very little extra characters. For simple applications with only one or two external communication interfaces, this is a usable format.

The above format can be enhanced to remove the restriction on explicit ordering of data values. Thus, the data can be sent as:

"Input1=12\$Input2=25\$Command=Add"

where \$ is a delimiter.

Here the data values need not be in any given order, as each data value has been associated with a name. However, it still requires both the sender and the receiver to know the names of the arguments to expect in the message.

Any large scale, enterprise application cannot work with the above formats because of their obvious limitations. Extended Markup Language (XML) has been an industry standard text format for exchange of information between communicating parties. The structure, the content, and to some extent, semantics of XML documents are defined using XML schemas, which can either be exchanged or made available via a repository for all communicating parties. Using XML, the above data values can be represented, as shown in Example 6-1.

Example 6-1 Data representation using XML

```
<?xml version="1.0" encoding="utf-8"?>
<CalculatorInputElement xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://tempuri.org/XMLSchema.xsd">
  <Input1>28</Input1>
  <Input2>3</Input2>
  <Command>ADD</Command>
</CalculatorInputElement>
```

Note: The parsing overhead for the above text is much more than with the previous simpler formats. Fortunately, there are a lot of third-party XML parsers available. Some, like the Java Architecture for XML Binding (JAXB), provide an API and tools that automate the mapping between XML documents and Java objects.

In this sample scenario, we use XML formatted data and JAXB for automatic mapping between XML and Java objects. For more information about JAXB, refer to the following URL:

<http://java.sun.com/webservices/jaxb/>

6.1.3 The calculator service

In this sample component interoperability scenario, the business function provided by the WebSphere service is a simple calculator that is contained in a JAR file (Calculator.jar).

Calculator.jar contains a class called CalculatorService.java with the methods shown in Example 6-2.

Example 6-2 CalculatorService methods

```
...
public void setCurrentTotal(float arg)
public float getCurrentTotal()
public float add(float arg1)
public float add(float arg1, float arg2)
public float subtract(float arg1, float arg2)
public float add(ICalculator3Args args)
...
```

The only methods that we use in this scenario are:

```
public float add(float arg1, float arg2)
public float subtract(float arg1, float arg2)
```

Figure 6-1 on page 195 shows the class hierarchy of the Calculator component implementation.

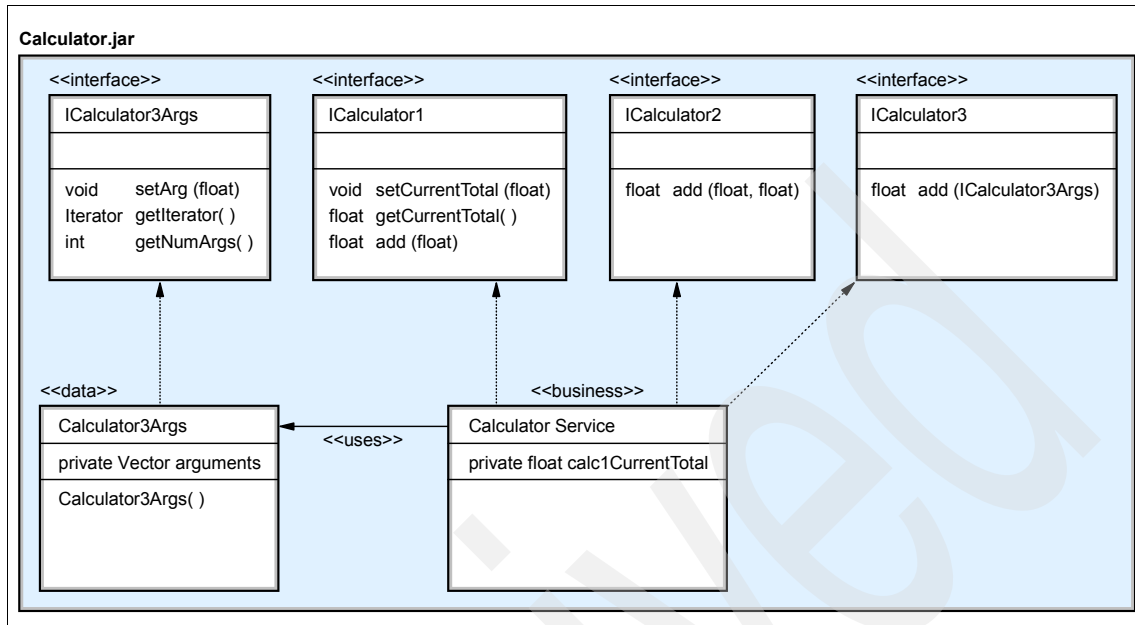


Figure 6-1 Calculator component class diagram

To view the actual implementation of the code within Calculator.jar, simply create a new Java project in IBM Rational Application Developer for WebSphere Software, and import the Calculator.jar file (select **File** → **Import** → **Zip file**, and select **Calculator.jar**), as shown in Figure 6-2.

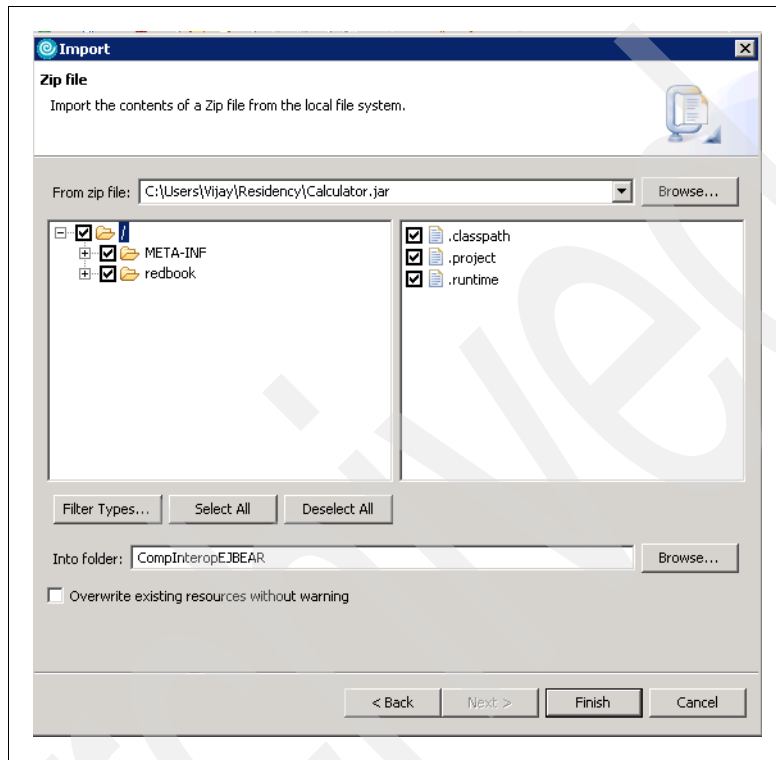


Figure 6-2 Importing the calculator service code into the current project for browsing

6.1.4 Messaging using WebSphere MQ

Message-oriented middleware has been used to provide the mechanism for multi-platform integration. WebSphere MQ messaging middleware platform has been used to realize numerous business-critical integrations.

A typical integration involving interaction between a service provider and a client service consumer using messaging is shown in Figure 6-3 on page 197.

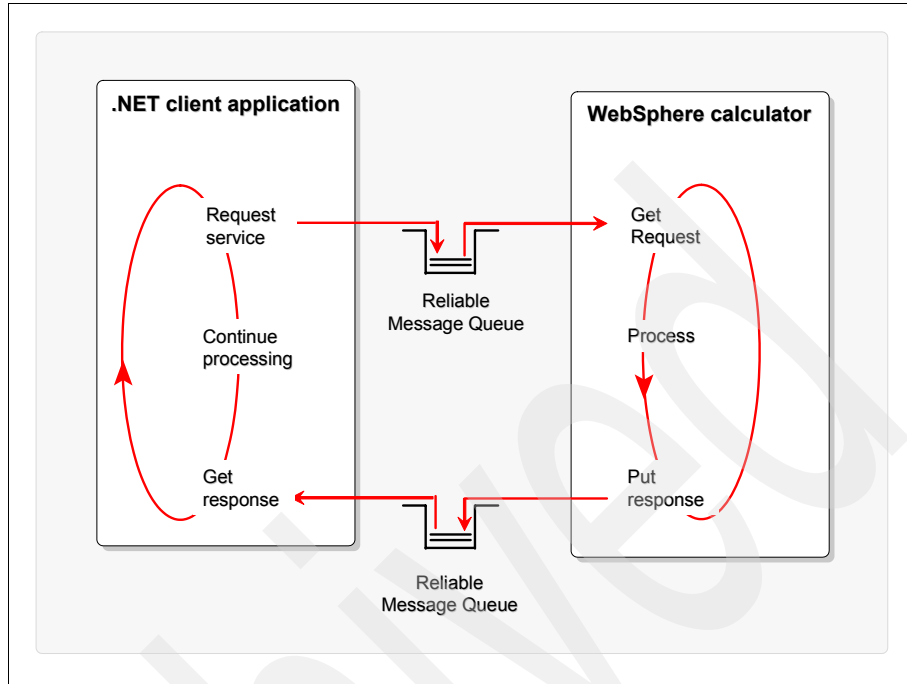


Figure 6-3 Interaction between a service provider and a consumer using messaging

WebSphere MQ implements highly reliable message queuing and ensures that messages sent from one program to another arrive at the destination exactly once without loss or corruption of the message data.

The service and client applications normally reside on different machines, linked by a network, and are connected by WebSphere MQ servers installed on each machine. Alternatively, you can install the WebSphere MQ server on one machine and a WebSphere MQ client on the other machine, as shown in Figure 6-4.

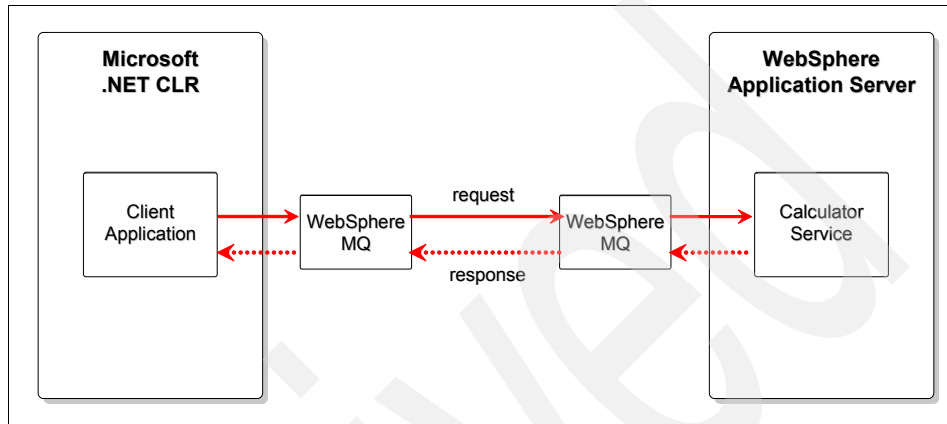


Figure 6-4 .NET client invoking WebSphere service

In the case where WebSphere MQ Servers are installed on each machine, different Queue Managers can be configured on each machine and messages can be sent between the different queue managers by defining remote queues, as shown in Figure 6-5 on page 199.

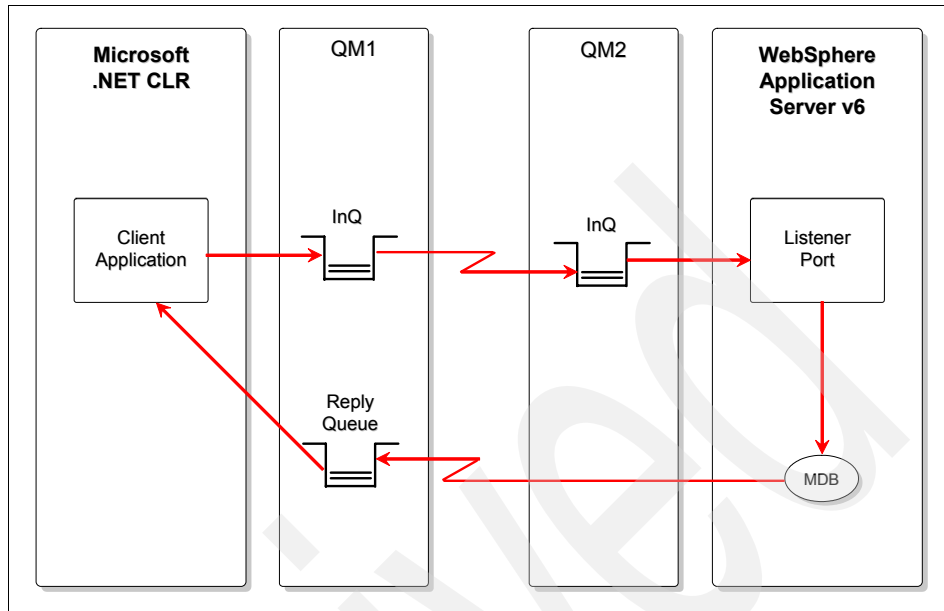


Figure 6-5 Accessing remote queue managers

Alternatively, you can install a WebSphere MQ Server on the server machine and WebSphere MQ client installation on the client. In this case, only one Queue Manager is created on the server machine. All the queues are created inside this Queue Manager. The client machine accesses the queues via WebSphere MQ client using TCP/IP. This is a more typical configuration and this is the configuration we used in this sample interoperability scenario. This configuration is depicted in Figure 6-6.

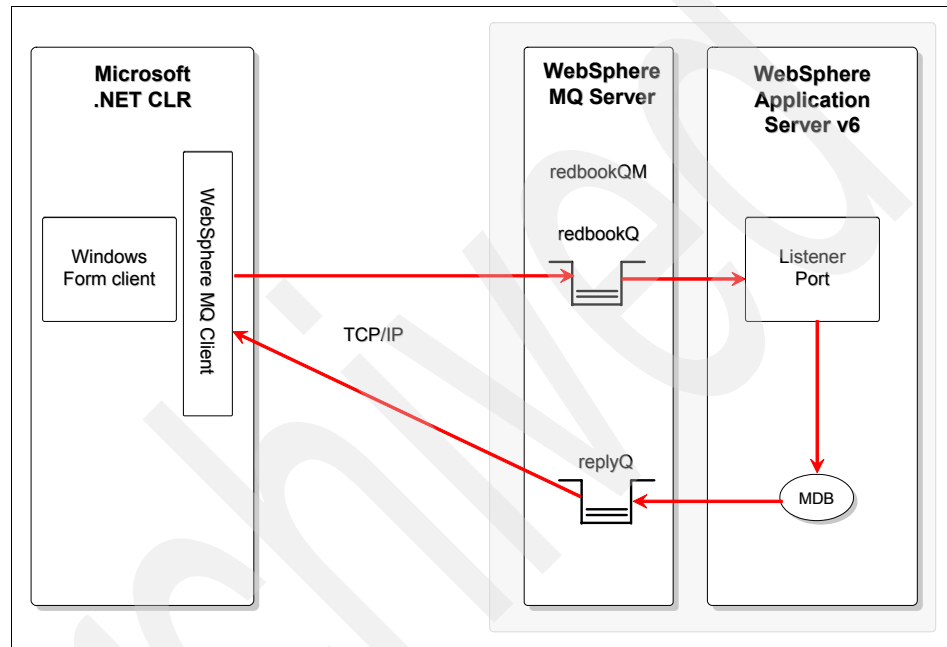


Figure 6-6 WebSphere MQ setup used by the scenario

Using the WebSphere MQ Client's "MQ Classes for .NET", the .NET Windows Form client application communicates with the server by putting messages onto the queue *redbookQ*, which was created by the queue manager *redbookQM* on the server machine. The queue listener on the server receives the messages that are forwarded to the Message-Driven Bean (MDB). The MDB invokes the methods in the Calculator and the result message is placed on the *replyQ* queue. The .NET Windows Form client application retrieves the result message from the *replyQ* message queue using a client connection.

6.1.5 System prerequisites

In order to build and execute the examples provided here, the following software packages are required. You can use the same set of products to configure scenarios where the .NET Windows Form client application component and the WebSphere calculator service component run on the same machine or different machines:

- ▶ .NET client
 - Microsoft .NET Framework V1.1
 - Microsoft Visual Studio.NET 2003
 - IBM WebSphere MQ Client V6.0
- ▶ WebSphere server
 - IBM Rational Application Developer for WebSphere Software
 - IBM WebSphere MQ V6.0

Note: Version information corresponds to the versions we used in our example, and not necessarily the minimum configuration required. For minimum configuration requirements for these components, please refer to the product documentation.

6.2 Solution overview

In this section, we present the overview of our solution for achieving interoperability between a .NET client component and a WebSphere service component using messaging. The solution involves the development of the WebSphere Calculator service component exposed through a J2EE message-driven bean (MDB) and the .NET rich client Windows Form application component.

The solution consists mainly of the following components:

- ▶ The client component
 - .NET Windows Form
 - WebSphere MQ client
- ▶ The service component
 - WebSphere Application Server Version 6
 - WebSphere MQ server
 - JAXB

We used a simple XML message format and JAXB for serialization and deserialization (marshalling/unmarshalling) of Java Object to XML and from XML to Java objects, respectively.

Another design point is the use of Interoperability Adapter. This isolates the client application implemented as presentation logic using the .NET Windows Form from the actual interoperability technology (the interoperability stack). The Interoperability Adapter provides a level of abstraction, making it possible to change the interoperability control or transport without requiring any changes to the .NET Windows Form application. For example, to use component bridging technology for this scenario will require the use of a different Interoperability Adapter without changes to the .NET Windows Form code.

The steps for creating the solution are summarized as follows:

- ▶ Install and configure the queues.
 - Installing and configuring (creation of queue manager, queues, and server connection channels) of WebSphere MQ server on the WebSphere Application Server V6 machine
 - Installing and configuring of WebSphere MQ client on the .NET machine
- ▶ Create messaging resources, schema and classes.
 - Creating JMS resources in a WebSphere Application Server V6 machine
 - Creating XML schema using the XML schema definition tool
 - Generating .NET classes for the XML schema using schema definition tool
 - Generating Java classes for the XML schema using the JAXB compiler
- ▶ Develop the .NET Windows Form client application.
 - Developing the Interoperability Adapter and associated classes
 - Developing the .NET Windows Form
- ▶ Develop the WebSphere service.
 - Developing the WebSphere Message Driven Bean (MDB)
 - Exporting and deploying the WebSphere service and testing the solution

Now we will describe each of the above steps in detail in the following subsections.

The overall architecture of our solution is shown in Figure 6-7 on page 203.

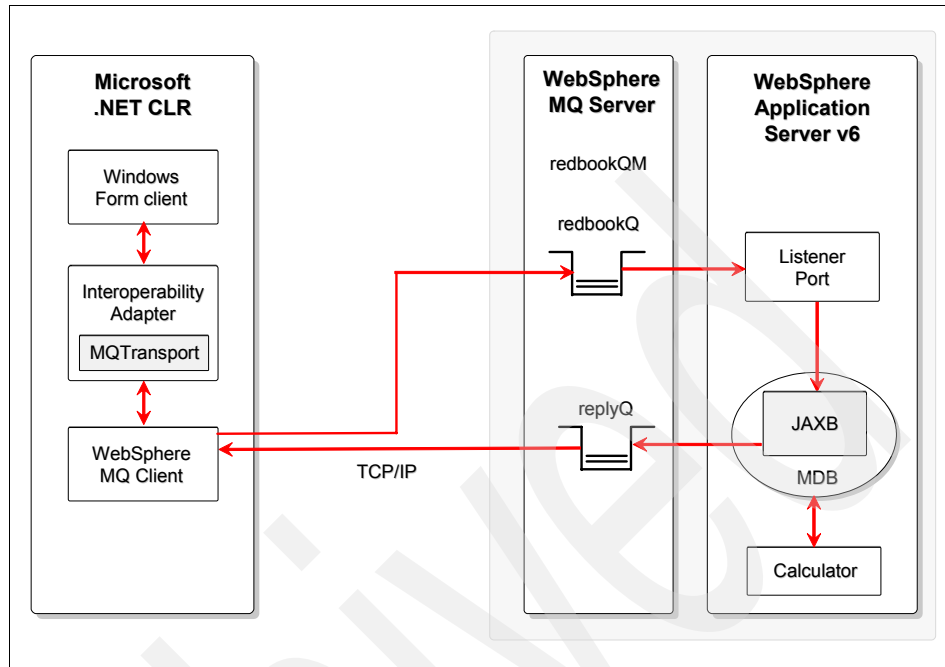


Figure 6-7 Solution architecture overview

6.3 Queues installation and configuration

In this section, we describe how to install and configure the WebSphere MQ server, client, and queues that provide the transport for our solution.

6.3.1 Install and configure the WebSphere MQ Server on the server

The calculator component in our sample interoperability scenario is a set of Java classes that are deployed in a WebSphere Application Server runtime environment running on a server machine. The WebSphere MQ Server is installed in this environment. The following steps assume that WebSphere MQ has been installed in a directory called <WebSphere_MQ_Root>:

1. In order to use WebSphere MQ in a test setting without any authentication mechanism being used, we created a system environment variable called MQSNOAUT and set its value to yes, as shown in Figure 6-8 on page 204.

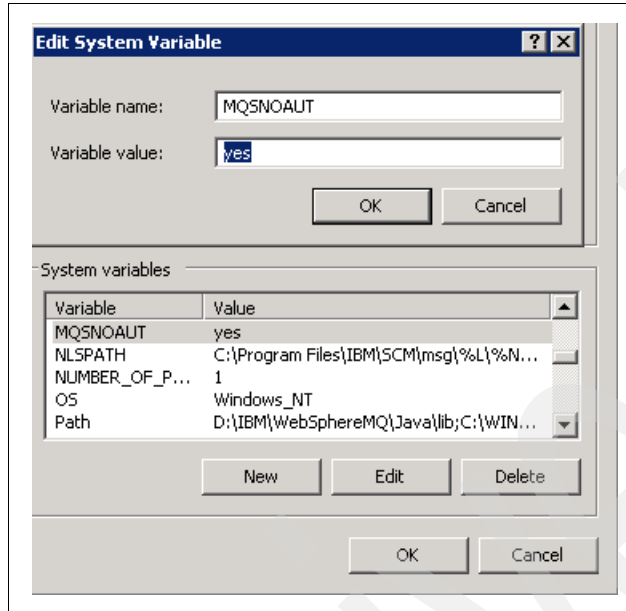


Figure 6-8 Creating the environment variable MQSNOAUT to turn off authentication

Important: If the MQSNOAUT variable is not created and its value is not set to yes before creation of the queue manager, it can lead to authentication failures (with WebSphere MQ reason code 2035) when you run the application. If you want to use proper authentication for your queue manager and queues, you can use the **setmqaut** command. For more information about **setmqaut**, refer to:

<http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.amqzag.doc/zsetaut.htm>

2. Start the WebSphere MQ Explorer by double-clicking **<WebSphere_MQ_Root>\bin\strmqcfg**.
3. A queue manager, redbookQM, is created by right-clicking **QueueManagers** and clicking **New** → **QueueManager**. Enter redbookQM as the name of the queue manager.
4. Create two local queues redbookQ and replyQ by right-clicking **Queues** under redbookQM and selecting **New** → **Local Queue**. Accept all the default settings. The queues are shown in Figure 6-9 on page 205. The redbookQ will be used by the .NET Windows Form client application to send its request messages and it provides replyQ as replyTo queue, where the WebSphere

calculator will place its response message so it can be picked up by the .NET client application (see Figure 6-9).

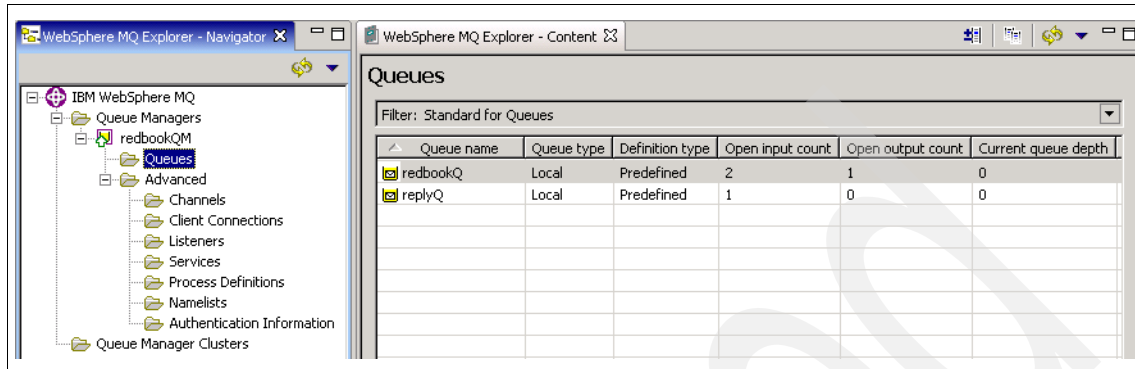


Figure 6-9 Queue Manager and Queues in WebSphere MQ

5. WebSphere MQ provides two types of connections to connect to a Queue Manager: BINDINGS and CLIENT. BINDINGS connections gives you better performance, but it can be used only if the queue manager is on the same physical machine. CLIENT connections use TCP/IP to connect and can be used to connect to a queue manager on the same or a different machine.

Since the WebSphere Application Server and the WebSphere MQ queue manager redbookQM were installed on the same machine, we used BINDINGS transport to connect the WebSphere calculator component to the queue manager for retrieval of messages from the redbookQ and for placing response messages into replyQ. The .NET Windows Form client application was installed on a different machine and therefore it used the WebSphere MQ CLIENT connections to connect to the queue manager redbookQM on the server. To accept these connections, the queue manager needed to be accessed remotely over TCP/IP over a server connection channel. We created a server connection with the name redbookQMChnl, as shown in Figure 6-10 on page 206, using the following steps:

- a. To create a server connection channel, make sure that you start the redbookQM queue manager by right-clicking it and clicking **Start** in WebSphere MQ Explorer.
- b. Click **Advanced** → **Channels** → **New** → **Server-connection channel**.

- c. Specify the name of the server connection channel as redbookQMChnl and accept all the other default properties, as shown in Figure 6-10.

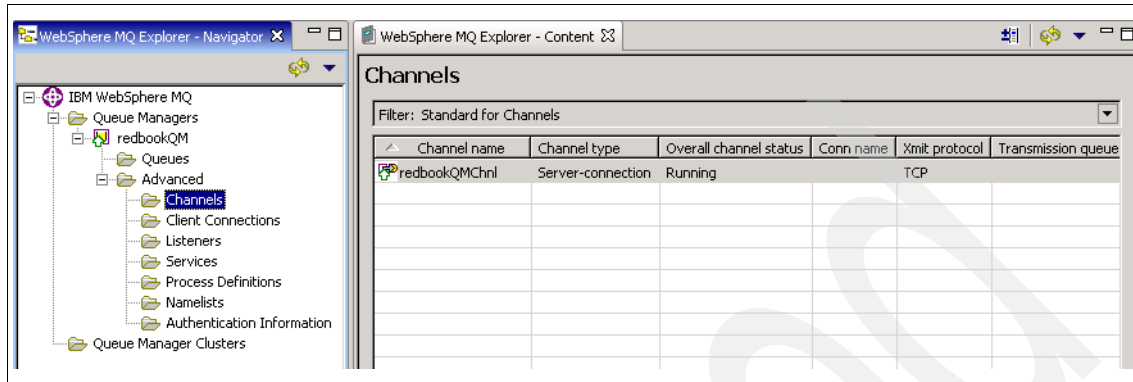


Figure 6-10 Server connection on WebSphere MQ to receive client connections

6. You also need to create a TCP listener (if one has not been already created during queue manager creation) to listen for incoming TCP connection requests. We created a listener with the name redbookLsr on port 1414 using the following steps:
 - a. To create a server connection channel, make sure that you start the redbookQM queue manager by right-clicking it and clicking **Start** in WebSphere MQ Explorer.
 - b. Click **Advanced** → **Listeners** → **New** → **TCP Listener**, as shown in Figure 6-11 on page 207.

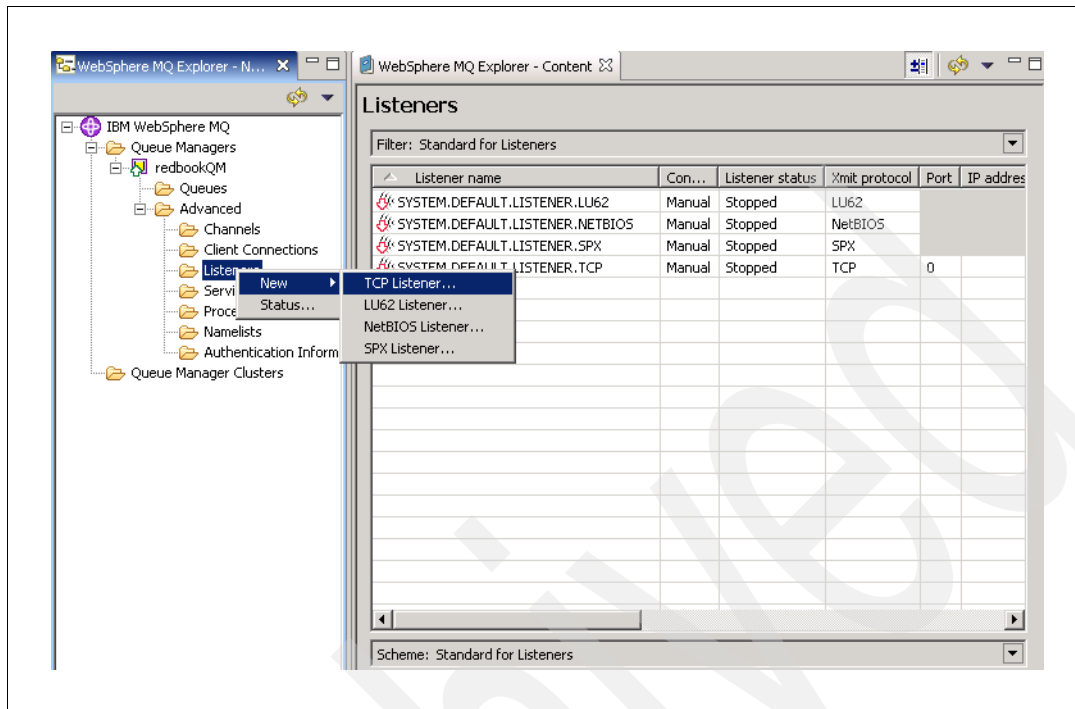


Figure 6-11 Creating a new TCP Listener

- c. Specify **redbookLsr** as the name of the listener, 1414 as the port number, and choose **Queue Manager Start** as the control method so that it starts off automatically whenever the queue manager **redbookQM** is started. The settings for the TCP listener are shown in Figure 6-12.

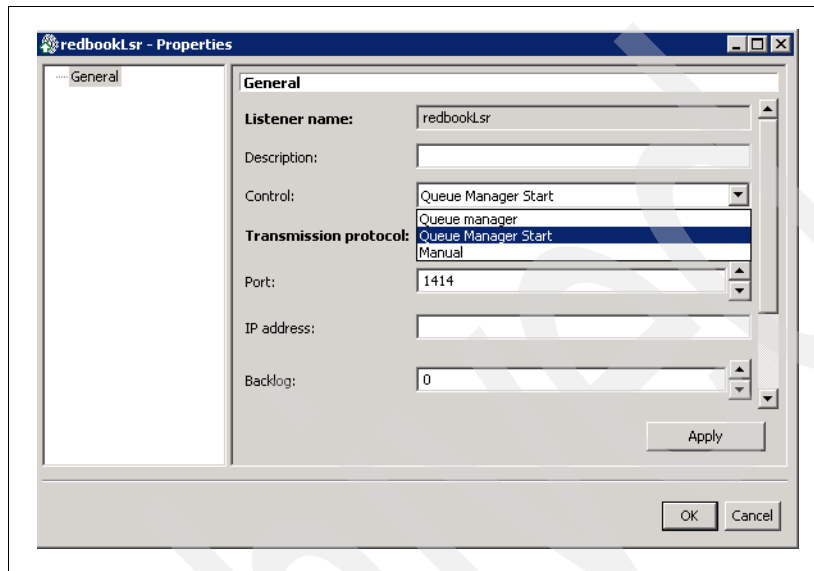


Figure 6-12 Settings for the TCP Listener

- d. Start the **redbookLsr** TCP listener by right-clicking **redbookLsr** and clicking **Start**, as shown in Figure 6-13 on page 209.

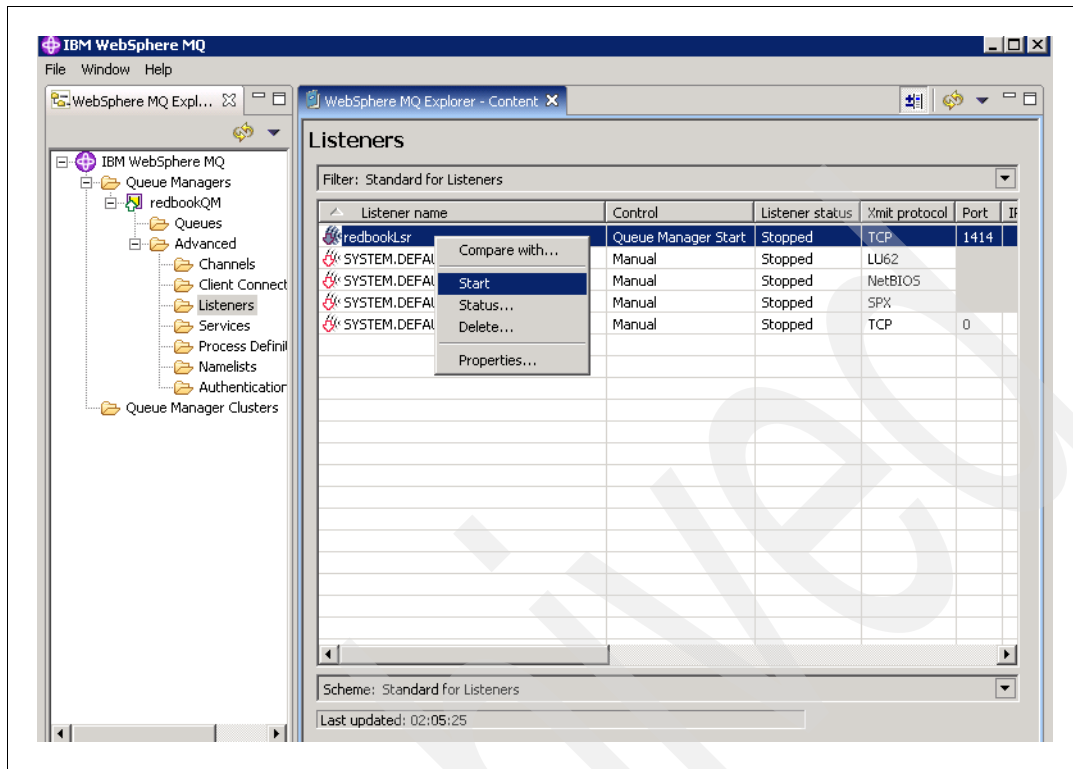


Figure 6-13 Starting the TCP Listener

6.3.2 Install and configure WebSphere MQ Client on the client

The WebSphere MQ client is installed on the client machine where the .NET Windows Form client application is deployed. The WebSphere MQ client configuration is minimal and consists of the following two steps:

1. Setting the MQServer environment variable

Create an environment variable with the name MQServer that points to the machine that hosts the queue manager. The value of this variable is of the form: *<Server connection name>/<type of transport>/<hostname of the machine that hosts the queue manager>*. In our sample application, the value of MQServer environment variable is redbookQMChnl/TCP/washost.itso.ral.ibm.com, as shown in Figure 6-14.

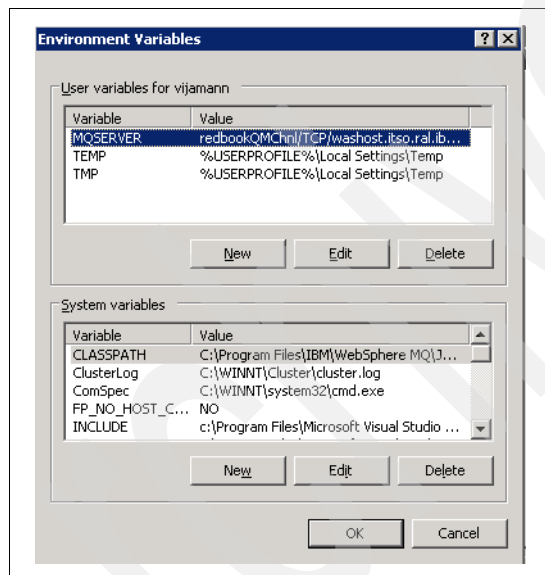


Figure 6-14 Setting up the MQServer Environment Variable on the .NET machine

2. WebSphere MQ classes for .NET

WebSphere MQ classes for .NET are available as DLLs with the names amqmdnet.dll and amqmdxcs.dll, which can be found under *<WebSphere_MQ_Root>\bin*.

For information about how to use WebSphere MQ classes for .NET, refer to the following programming guide:

<ftp://ftp.software.ibm.com/software/integration/support/supportpacs/individual/csqzav01.pdf>

This documentation is also available at:

<http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/topic/com.ibm.mq.csqzav.doc/csq870x.htm>

6.4 Create messaging resources, schemas, and classes

In this section, we describe how to create the message resources, schemas, and classes you need on the client and server sides.

6.4.1 Create JMS resources in WebSphere Application Server

JMS resources for WebSphere MQ needs to be created in the WebSphere Application Server.

Creating WebSphere MQ Queue Connection Factories

In WebSphere Application Server, we created two WebSphere MQ Queue Connection Factories, redbookQCF and redbookClientQCF, as shown in Figure 6-15. To create the WebSphere MQ queue connection factories, click **Resources** → **JMS Providers** → **WebSphere MQ** → **WebSphere MQ queue connection factories** → **New** and specify the settings described in Figure 6-15, and shown in Figure 6-16 on page 213 and in Figure 6-17 on page 214.



Figure 6-15 WebSphere MQ Queue Connection Factories in WAS

The connection factory settings are used by the Messaging Service in WebSphere Application Server to connect to the Queue Manager specified for a particular Queue Connection Factory. JMS applications make use of Connection Factories to connect to Queue Managers and do not have knowledge of Queue Managers that they connect to. The two connection factories that we created, redbookQCF and redbookClientQCF, both connect to the same queue manager redbookQM. In this case, one could use just one queue connection factory. However, we decided to use two, so we use different connection factories (redbookQCF for connecting to the redbookQ and redbookClientQCF for connecting to the replyQ) so that if one needed to move to a different configuration where both queues did not exist on the same machine, then all one needed to do was to change the connection factory settings for the queue to specify a different queue manager (on the remote machine) and no code changes would be needed. Both connection factories use the BINDINGS transport, as the queue manager redbookQM resides on the same machine as WebSphere Application Server. These settings are shown in Figure 6-16 on page 213 and in Figure 6-17 on page 214. JNDI names for these two connection factories are specified as jms/redbookQCF and jms/redbookClientQCF respectively. All the other settings are the default settings.

* Name
redbookQCF

* JNDI name
jms/redbookQCF

Description

Category

Component-managed authentication alias
(none) ▼

Container-managed authentication alias
(none) ▼

Mapping-configuration alias
DefaultPrincipalMapping ▼

Queue manager
redbookQM

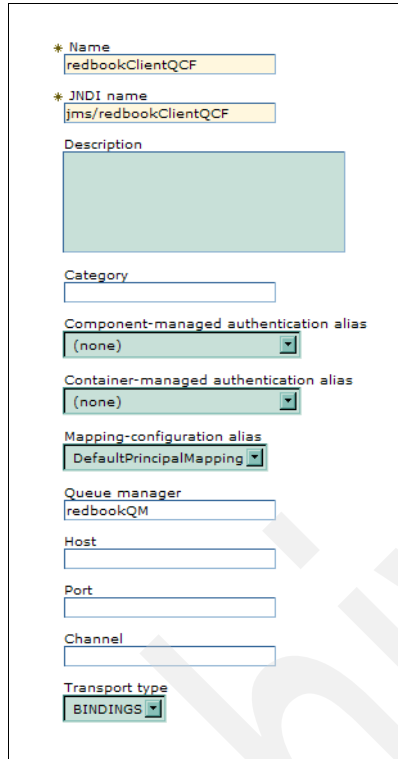
Host

Port

Channel

Transport type
BINDINGS ▼

Figure 6-16 WebSphere MQ Queue Connection Factory Settings for redbookQCF



The image shows a screenshot of the WebSphere MQ Queue Connection Factory Settings for a resource named 'redbookClientQCF'. The settings are as follows:

- Name:** redbookClientQCF
- JNDI name:** jms/redbookClientQCF
- Description:** (Empty text area)
- Category:** (Empty text field)
- Component-managed authentication alias:** (none)
- Container-managed authentication alias:** (none)
- Mapping-configuration alias:** DefaultPrincipalMapping
- Queue manager:** redbookQM
- Host:** (Empty text field)
- Port:** (Empty text field)
- Channel:** (Empty text field)
- Transport type:** BINDINGS

Figure 6-17 WebSphere MQ Queue Connection Factory Settings for redbookClientQCF

Creating WebSphere MQ Queue Destinations

We created two WebSphere MQ Queue destinations in WebSphere Application Server, redbookQ and replyQ, corresponding to the queues that were created in WebSphere MQ. To create the WebSphere MQ queue destinations click

Resources → **JMS Providers** → **WebSphere MQ** → **WebSphere MQ queue destinations** → **New**.

These destinations are shown in Figure 6-18.

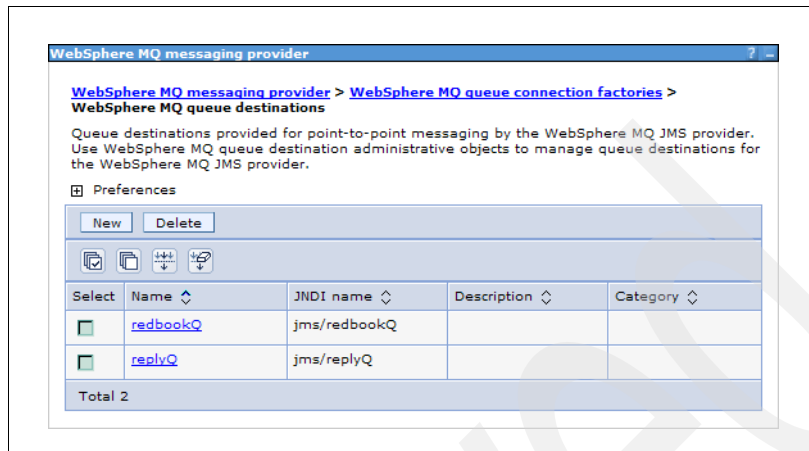
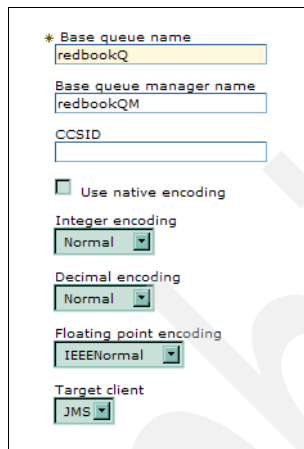


Figure 6-18 WebSphere MQ Queue Destinations in WebSphere Application Server

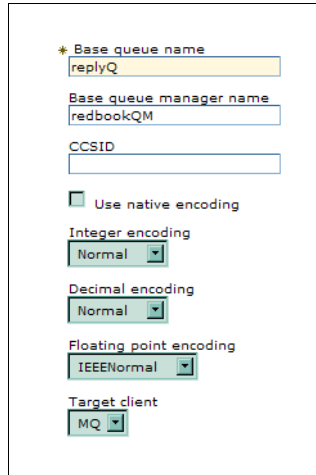
The settings for these two queue destinations are shown in Figure 6-19 and in Figure 6-20 on page 217. Both queue destinations use the same queue manager name redbookQM. All the other settings are the default settings, except the Target client setting for the replyQ. The Target client setting for a queue destination specifies whether the receiving application is JMS-compliant or is a traditional WebSphere MQ application that does not support JMS. This is a new feature in WebSphere Application Server Version 6.0 and is used by the application server to decide whether to put a JMS header in the message or not. In our solution, since the receiving application for replyQ is a .NET application that does not understand JMS, we set the Target Client for replyQ as WebSphere MQ (see Figure 6-19).

A screenshot of the WebSphere MQ Queue Destination Settings for redbookQ. The settings are as follows: Base queue name: redbookQ; Base queue manager name: redbookQM; CCSID: (empty); Use native encoding: (checked); Integer encoding: Normal; Decimal encoding: Normal; Floating point encoding: IEEENormal; Target client: JMS.

* Base queue name
redbookQ
Base queue manager name
redbookQM
CCSID
<input checked="" type="checkbox"/> Use native encoding
Integer encoding
Normal
Decimal encoding
Normal
Floating point encoding
IEEENormal
Target client
JMS

Figure 6-19 WebSphere MQ Queue Destination Settings for redbookQ in WebSphere Application Server

Figure 6-20 on page 217 shows the WebSphere MQ Queue Destination Settings for replyQ in WebSphere Application Server.



* Base queue name
replyQ

Base queue manager name
redbookQM

CCSID

☐ Use native encoding

Integer encoding
Normal

Decimal encoding
Normal

Floating point encoding
IEEENormal

Target client
MQ

Figure 6-20 WebSphere MQ Queue Destination Settings for replyQ in WebSphere Application Server

Attention: The ability to specify whether the target client (consumer) for a queue is JMS compliant or not is an important one. This is a new feature in WebSphere Application Server Version 6 and is very useful. You can specify the target client as MQ whenever the consumer of that message is a non-JMS application (like the .NET client in our solution). When you specify this feature, WebSphere Application Server removes the JMS header from the message that goes into the queue. In the absence of such a feature, WebSphere Application Server will continue to put messages with a JMS header into the queue and the consumer application will have to take care of removing the JMS header while processing the message. The value of the target client can also be set programmatically in Java as follows:

```
javax.jms.Queue queue = (javax.jms.Queue)msg.getJMSReplyTo();
com.ibm.mq.jms.MQQueue q = (com.ibm.mq.jms.MQQueue) queue;
q.setTargetClient(1);
```

A value of 1 specifies MQ as the target client, while a value of 0 specifies JMS as the target client. We set target client as MQ in both ways, via the admin console settings for the replyQ, as well as programmatically in the onMessage() method call of TestMDBBean, as shown in Example 6-14 on page 240.

Creating a listener port

A listener port defines the association between a connection factory, a destination, and a deployed message-driven bean. Listener ports are used to simplify the administration of the associations between these resources. When a deployed message-driven bean (MDB) is installed, it is associated with a listener port and the listener for a destination. When a message arrives at the destination, the listener passes the message to a new instance of a message-driven bean for processing. In our sample application, the .NET client application sends a request message, and it is picked up by an MDB deployed on WebSphere Application Server. Therefore, we have to create a listener port that associates the connection factory and the WebSphere MQ queue destination.

We created a listener port with the name TestLP and associated it with the connection factory jms/redbookQCF and destination queue jms/redbookQ (redbookQ). The destination queue specified is jms/redbookQ and not jms/replyQ (replyQ), as jms/redbookQ is the destination queue used by the .NET client application to send its message, and therefore the listener port should pick the message from this queue. To create a listener port, click **Servers** → **Application servers** → **server1** → **Messaging** → **Message Listener Service** → **Listener Ports** → **New** and specify the settings as described above and also shown in Figure 6-21.



Figure 6-21 Listener Port in WebSphere Application Server

6.4.2 Create the XML schema for messages

This sample scenario uses XML for sending and receiving messages. This means that we had to create an XML schema for all our messages. We designed the XML schema using Microsoft Visual Studio .NET 2003's XML Schema Definition Tool. To do this, use the following steps:

1. Create a new XML schema by clicking **File** → **New** → **File** → **General** → **XML Schema**. Press the **Open** button. The XML Schema Definition Tool will open up.

This step is shown in Figure 6-22.

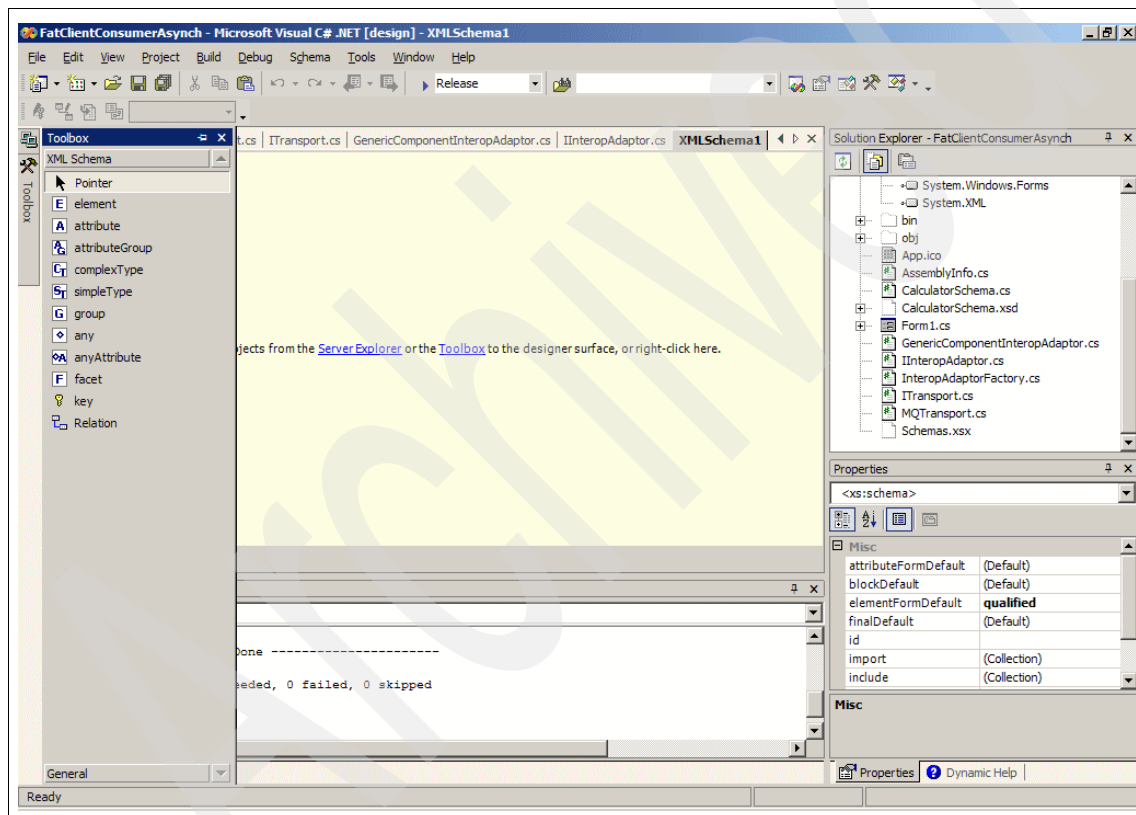


Figure 6-22 Creating the XML schema using the XML schema definition tool in Microsoft Visual Studio .Net 2003

2. Use the toolbox on the left to drag and drop widgets to create a complex type, CalculatorInput with three elements, Input1 of type float, Input2 of type float, and Result of type float. Create two elements: CalculatorInput of type float and Result of type float.

The final XML schema definition tool window should look something like the one shown in Figure 6-23.

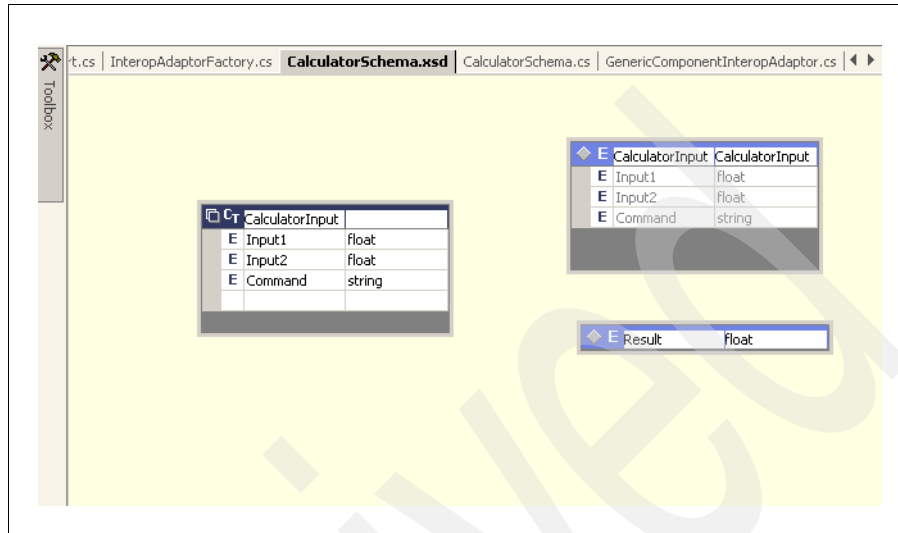


Figure 6-23 CalculatorSchema developed in the XML schema definition tool

3. Save the schema with the name CalculatorSchema.xsd. The schema will look like what is shown in Example 6-3.

Example 6-3 Text form of CalculatorSchema

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://tempuri.org/XMLSchema.xsd"
  elementFormDefault="qualified"
  xmlns="http://tempuri.org/XMLSchema.xsd"
  xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="CalculatorInput">
    <xs:sequence>
      <xs:element name="Input1" type="xs:float" />
      <xs:element name="Input2" type="xs:float" />
      <xs:element name="Command" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="CalculatorInputElement"
    type="CalculatorInput"></xs:element>
  <xs:element name="Result" type="xs:float"></xs:element>
</xs:schema>
```


6.4.3 Generate the .NET classes corresponding to the XML schema

To generate the .NET classes, corresponding to the CalculatorSchema, we used the xsd.exe utility, which comes with Microsoft Visual Studio .NET 2003. The following steps are used:

1. Start the Visual Studio .NET command prompt.
2. Go to the directory where the CalculatorSchema.xsd file resides. At the command prompt, type:

```
xsd CalculatorSchema.xsd /c
```
3. It should generate a class file with the name CalculatorSchema.cs in the current directory. This class should be added to the .NET project.

6.4.4 Generate the Java classes corresponding to the XML schema

We used Java Architecture for XML Binding (JAXB) for generating the Java classes corresponding to the schema and also for automatic serialization and deserialization of Java classes into XML (and vice versa). JAXB now comes as part of the Java Web Services Developer Pack (JWSDP) 2.0 and can be downloaded from:

<http://java.sun.com/webservices/downloads/webservicespack.html>

To generate the Java classes, we used the JAXB compiler (xjc.bat).

Invoke it as follows:

```
<JWSDP_INSTALL_ROOT>\jaxb\bin\xjc.bat -extension  
<FULL_PATH_OF_THE_SCHEMA_FILE>\CalculatorSchema.xsd
```

This should generate Java files in three packages:

- ▶ redbook.coex.async.stateless
- ▶ redbook.coex.async.stateless.impl
- ▶ redbook.coex.async.stateless.impl.runtime

Note: At the time of the writing of this redbook, the latest version of JAXB is v1.0. For more information, refer to:

<https://jaxb.dev.java.net/>

6.5 Developing the .NET Windows Form client application

The .NET Windows Form client application provides the graphical user interface for accepting the user data and for displaying the result of the interoperability by calling the WebSphere calculator service component. The .NET client application is one of the two elements that make up the presentation tier for this sample scenario. The other element is the interoperability adapter, which interfaces with the server side through the WebSphere MQ messaging queues. The interoperability adapter performs a number of functions. It provides a level of abstraction that hides the interoperability control technology, and provides the implementation for the application and data layers in the application interoperability stack. In this section, we describe the steps for developing the interoperability adapter and then the Windows Form client application.

6.5.1 Developing the interoperability adapter

Though we have a very simple example application in the calculator service, we demonstrate how you can implement an interoperability adapter to create an interoperability layer to provide the application interface, data serialization, and deserialization. It also hides the interoperability control and transport from the client application.

The interoperability adapter consists of the elements described in the following sections.

An adapter factory - InteropAdaptorFactory

The adapter factory performs two key functions: first, it enables the client application to obtain the appropriate adapter by merely specifying the settings for the type of adapter. Second, it enables flexibility in making changing to the features and capabilities of adapters.

The presentation layer gets a reference to a valid interoperability adapter by making the following static method call:

```
static IInteropAdaptor createInteropAdaptor(InteropAdaptorType adaptorType,
SerializationType serType)
```

The InteropAdaptor types currently specified in the InteropAdaptorFactory are Messaging and Bridge. The Serialization types currently specified in the InteropAdaptorFactory are XML and Binary. Our sample component interoperability scenario illustrates only the messaging type interoperability adaptor that uses XML serialization. The InteropAdaptorFactory also specifies three Call types: Request, Response, and RequestResponse.

Interface for the adapter - IInteropAdaptor

The IInteropAdaptor interface has the following methods, as shown in Example 6-4.

Example 6-4 IInteropAdaptor interface

```
...
// specifies the transport type
void setTransport(ITransport transport);

//returns the transport object
ITransport getTransport();

// specifies the data type of response object for deserialization
void setResponseAttributes(Object returnType, Object attributes);

// make the actual method call passing the arguments and call type
Object execute(Object[] args, InteropAdaptorFactory.CallType callType);
...
```

Interface for the transport - ITransport

The ITransport interface has the following methods, as shown in Example 6-5.

Example 6-5 ITransport methods

```
...
String connectionSetup();
String connect();
void disconnect();
bool sendOneWayMessage(Object[] args);
Object getMessage(bool blocking);
Object sendRequestResponseMessage(Object[] args);
void printConnectionProperties();
...
```

Implementation of ITransport for MQ - MQTransport

This is the class where the actual implementation for communicating with WebSphere MQ using the .NET classes for MQ resides. In this class, the first thing to note is the using IBM.WMQ statement at the top of the C# code. This is done to make sure that we get access to all the WebSphere MQ for .NET classes in the amqmdnet.dll. At the top of this file, the details about the queue manager name and the queue names was hardcoded for our sample application, as shown in Example 6-6.

Example 6-6 Queue manager and queue names

```
private MQQueueManager mqQMGr;
private MQQueueManager replyTomqQMGr;
private MQQueue mqSendQueue; // MQQueue for sending Messages
private MQQueue mqReceiveQueue; //MQQueue for receiving Messages
private String queueManagerName="redbookQM";
private String queueName="redbookQ"; // Name of queue to use
private String replyToQueueManagerName = "redbookQM";// send replies to
this QM
private String replyToQueueName = "replyQ";
```

The following steps need to be performed:

1. The first step is connect to the queue manager in the connectionSetup call. This is shown in Example 6-7. The creation of this object is slightly different, depending on whether you are using the default Queue Manager, or whether the application is running on an MQSeries® Client or an MQSeries Server. See the WebSphere MQ documentation for more details.

Example 6-7 Connecting to the queue manager

```
public String connectionSetup()
{
    String str;
    try
    {
        // MQQueueManager instance

        mqQMGr = new MQQueueManager( queueManagerName);
        replyTomqQMGr = new MQQueueManager(replyToQueueManagerName);
    }
    catch (MQException mqe)
    {
        // stop if failed
        str ="The following MQ error occurred: " + mqe.Message+"
"+mqe.ToString();
        return str;
    }
}
```

```

        str ="Connection Setup Successful.";
        return str;
    }

```

2. Once a new instance of the MQQueueManager object has been created, we can open the queue of our choice. WebSphere MQ requires you to specify the actions you intend to take upon a queue when opening it and defines the constants to do so. These constants are exposed via the MQ Classes for .NET through the MQ.MQC public interface. For more information, see the WebSphere MQ documentation. The code we used to open the output queue (for writing messages) and the input queue (for receiving/reading messages) is given in Example 6-8.

Example 6-8 Opening the output and input queues

```

public String connect()
{
    String str;
    try
    {
        mqSendQueue = mqMgr.AccessQueue( queueName, // the name of the queue
            MQC.MQOO_OUTPUT // open queue for output
            + MQC.MQOO_FAIL_IF_QUIESCING ); // don't if MQM stopping
        mqReceiveQueue = mqMgr.AccessQueue( replyToQueueName,
            MQC.MQOO_INPUT_AS_Q_DEF // open queue for input
            + MQC.MQOO_FAIL_IF_QUIESCING );
    }
    catch (MQException mqe)
    {
        // stop if failed
        str= "The following MQ error occurred: " + mqe.Message+"
"+mqe.ToString();
        return str;
    }
    str ="Connection established.";
    return str;
}

```

3. Once we have opened the queue, we are ready to send our message. This is done in the `sendOneWayMessage(Object[] args)` method for `MQTransport`. The `Object` is passed to `sendOneWayMessage` is already in the form of a valid XML string. The MQ Classes for .NET abstracts the message into an object as well as the MQ Put Message Options. For more information about the Put Message Options, see the WebSphere MQ documentation.

The steps involved here are to create a string message, declare its length, create an instance of the message object, load the string message into that object, create Put Message Options, and set them as desired. Once the message is created, you simply call the *Put* method on our Queue object, passing the Message object and the Put Options object into it.

The code to send a message is shown in Example 6-9.

Example 6-9 Sending a message using MQ classes for .NET

```

public bool sendOneWayMessage(Object[] args)
{
    MQMessage          mqMsg;           // MQMessage instance
    MQPutMessageOptions mqPutMsgOpts;    // MQPutMessageOptions instance

    int                msgLen;           // Message length
    String              message;         // Message buffer
    // create message
    message = (String)args[0];
    msgLen = message.Length; // set message length
    mqMsg = new MQMessage(); // new message instance
    mqMsg.WriteString( message ); // load message w/payload
    mqMsg.Format = MQC.MQFMT_STRING; // declare format

    mqPutMsgOpts = new MQPutMessageOptions(); // declare options
    mqMsg.ReplyToQueueManagerName = replyToQueueManagerName; // send replies
to this QM
    mqMsg.ReplyToQueueName = replyToQueueName; // send replies to this queue
    // put message on queue
    try
    {
        mqSendQueue.Put( mqMsg, mqPutMsgOpts );
    }
    catch (MQException mqe)
    {
        // stop if failed

        Console.WriteLine("The following MQ error occurred: " + mqe.Message+"
"+mqe.ToString());
        return false;
    }

    return true;
}

```

4. Once a message has been sent to a queue, the next step is to retrieve any messages from the replyTo queue specified in the input message. This is done in the method `getMessage(bool blocking)`, as shown in Example 6-10 on page 227. The Boolean argument specifies whether the method should block

or not. The first step is to create a message object to hold the message we are about to get off of the queue. The message object is of type `MQMessage`. We define an object of this type and then create a new instance of it. We then create a `MQGetMessageOptions` object to define how we are going to get these messages. The `MQGetMessageOptions` object abstracts the MQ Get Message Options. For a full description of the capabilities, see the WebSphere MQ documentation.

Here, we use the Get Message Options to set a `WaitInterval` for our Get. This tells WebSphere MQ how long we would like to wait for a new message assuming one is not immediately available. If *blocking* is set to true, we wait for a really long time to simulate a blocking call; otherwise, we wait for three seconds. Now we are ready to get our message. We do so by invoking the `Get()` method on the queue object we created, passing both the newly created message object and the newly created Get Message Options. Whatever message object is read from the queue is returned to the user (of this class). `MQException` errors are thrown and caught via a try/catch block. With this method, errors can actually be normal. One *normal* error is No Message Available, which simply means that no messages were on the queue to be read. So, we surround the above code in a try { ... } and implement a catch { ... } to evaluate what happened in the event of an error (see Example 6-10).

Example 6-10 *Getting a message from a queue using MQ classes for .NET*

```
// implements a blocking/non-blocking get
public Object getMessage(bool blocking)
{
    MQMessage          mqMsg;          // MQMessage instance
    MQGetMessageOptions mqGetMsgOpts; // MQGetMessageOptions
    String strMessage="";
    mqMsg = new MQMessage();// create new message object
    mqMsg.Format=MQC.MQFMT_STRING;
    // mqMsg.Format=MQC.MQFMT_RF_HEADER_2; //
    // mqMsg.Encoding = 273; //
    // mqMsg.CharacterSet = 819; //

    // Get Message Options & set them
    mqGetMsgOpts = new MQGetMessageOptions();
    if(blocking)
    {
        Console.WriteLine("setting up blocking i/o");
        mqGetMsgOpts.WaitInterval = 3600 *100; // 1 hour limit
    }
    else
    {
        Console.WriteLine("setting up non blocking i/o");
        mqGetMsgOpts.WaitInterval = 3000; // 3 second limit for waiting
    }
}
```

```

// get the message
try
{
    //strMessage = " "+mqReceiveQueue.Name;
    mqReceiveQueue.Get( mqMsg, mqGetMsgOpts );
}
catch (MQException mqe)
{
    // report reason, if any
    if ( mqe.Reason == MQC.MQRC_NO_MSG_AVAILABLE )
    {
        // special report for normal end
        strMessage = "No message to read.";
        return strMessage;
    }
    else
    {
        // general report for other reasons
        strMessage = "MQ returned error: " + mqe.Message;
        return strMessage;
    }
}
strMessage = mqMsg.ReadString(mqMsg.MessageLength);
return (Object) strMessage;
}

```

The MQTransport class also has a `sendRequestResponse()` method that just simulates a synchronous call by making use of a combination of the `sendOneWayMessage` method and `getMessage()` method with blocking set to true.

Note: The MQ Transport class provides support for both asynchronous and a simulated synchronous mode of communication. This illustrates how one can achieve synchronous communication even with a messaging technology like WebSphere MQ.

Interoperability adaptor implementation: GenericComponentInteropAdaptor

This is the actual implementation of the interoperability adaptor. Its main purpose is to use the underlying transport for communication and provide the required XML serialization (conversion from normal string arguments from the Windows Form to valid XML) and deserialization (conversion from XML to valid string arguments that can be displayed in the Windows Form). The following using statements are required at the top of this class:

```
using System.Xml.Serialization; //For the XML Serializer Class
using System.IO;
using System.Text;
using System.Xml;
```

Most of its functionality lies in the execute method shown in Example 6-11.

Note: Two important points can be noted here. First, to deserialize the XML message from WebSphere into a valid .NET Object, the response attributes in terms of the type of the expected Object and the element name and namespace of the expected XML have to be specified. The GenericComponentInteropAdaptor checks this before making any method calls that expect a response. Second, as the default encoding used by JAXB is UTF-8, while the default encoding of strings in .NET is UTF-16, .NET strings have to undergo a transformation from UTF-16 to UTF-8. On the other side, the MDB on WebSphere Application Server, ensures (through a method call) that JAXB uses UTF-16 encoding for marshalling any Java object into XML before sending it across.

Example 6-11 XML serialization and deserialization and use of the transport object

```
public Object execute(Object [] args, InteropAdaptorFactory.CallType callType)
{
    Object response = new Object();
    XmlSerializer deserializer=null;
    StringBuilder sbuilder = new StringBuilder();
    StringWriter swriter = new StringWriter(sbuilder);
    if(callType==InteropAdaptorFactory.CallType.Response ||
    callType==InteropAdaptorFactory.CallType.RequestResponse)
    {
        if(responseType==null)
            return null;
        else
            deserializer = new XmlSerializer(responseType.GetType(),
            (XmlRootAttribute) responseAttributes);
    }
}
```

```

for(int i=0; i<args.Length;i++)
{
    XmlSerializer serializer = new XmlSerializer(args[i].GetType());
    serializer.Serialize(swriter, args[i]);
}
swriter.Close();
/* JAXB's default encoding is UTF-8 whereas strings in .NET use UTF-16
 * encoding. Therefore a transformation is required
 */
sbuilder.Replace("utf-16","utf-8");
Console.WriteLine(sbuilder.ToString());
switch(callType)
{
    case InteropAdaptorFactory.CallType.Request:
        transport.sendOneWayMessage(new Object[]{sbuilder.ToString()});
        return null;
    case InteropAdaptorFactory.CallType.RequestResponse:
        response = transport.sendRequestResponseMessage(new
Object[]{sbuilder.ToString()});
        break;
    case InteropAdaptorFactory.CallType.Response:
        response = transport.getMessage(false);
        break;
}
Console.WriteLine("Received response type is
"+response.GetType().ToString());
StringReader sreader = new StringReader((String)response);
String resp = (String)response;
if(resp.StartsWith("<?xml")) //deserialize only XML strings
    response = deserializer.Deserialize(sreader);
sreader.Close();
return response;
}

```

6.5.2 Developing the Windows Form .NET client

The .NET client component in our sample scenario is a rich client application that is implemented using Windows Form. The .NET assembly is written in C# and uses the MQ Classes for .NET to create properly formatted request messages and put them to the queue. Code also support retrieval of response messages from the queue:

- To create a new Windows Form based .NET application, open Microsoft Visual Studio .NET 2003 and create a New Project by selecting **File** → **New** → **Project** → **Visual C# Projects** → **Windows Application**.

You will need to select a name and location; we chose **FatClientConsumerAsynch**. Once you click **OK**, your Windows.Forms project will be created with a blank dialog.

- ▶ As depicted in Figure 6-24, we added a few text boxes to contain input values (called tbArg1 and tbArg2) and the result (called tbResult), and a few buttons (button1, button2 and GetResult). The buttons button1 and button2 implement the Add and Subtract commands and use largely the same code that creates the XML message and sends it to the redbookQ queue. The GetResult button implements the code that retrieves the message from the replyQ and displays the result in the tbResult textbox.

Figure 6-24 shows the Windows Form for the .NET client.

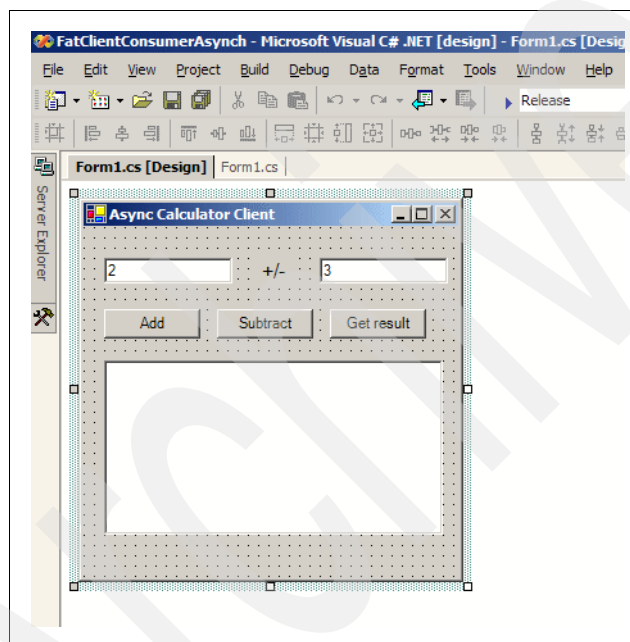


Figure 6-24 Windows Form for the .NET client

Follow these steps:

- a. The first step in implementing the code is to create a reference to the MQ Classes for .NET so that we can use them. There are two steps to this process. First, add a project reference within Solution Explorer by right-clicking **References** under the project we just created and selecting **Add Reference**.
- b. In the resulting dialog under the .NET tab, choose to browse for a DLL and select amqmdnet.dll and amqmdxcs.dll, which can be found under <WebSphere_MQ_Root>\bin.

- c. Once the button is placed on the form, double-clicking it will bring up the code that will be executed when the button is clicked. The name of this file is Form1.cs. We can either keep our example very simple, and implement all our code here, or we can implement an interoperability adapter, as described in the previous section. By implementing the interoperability adapter, we can easily switch between various interoperability control technologies (for example, from WebSphere MQ to a component bridge).

After executing these steps, your project should look something like Figure 6-25. Note the amqmdnet and amqmdxcs references in the Solution Explorer.

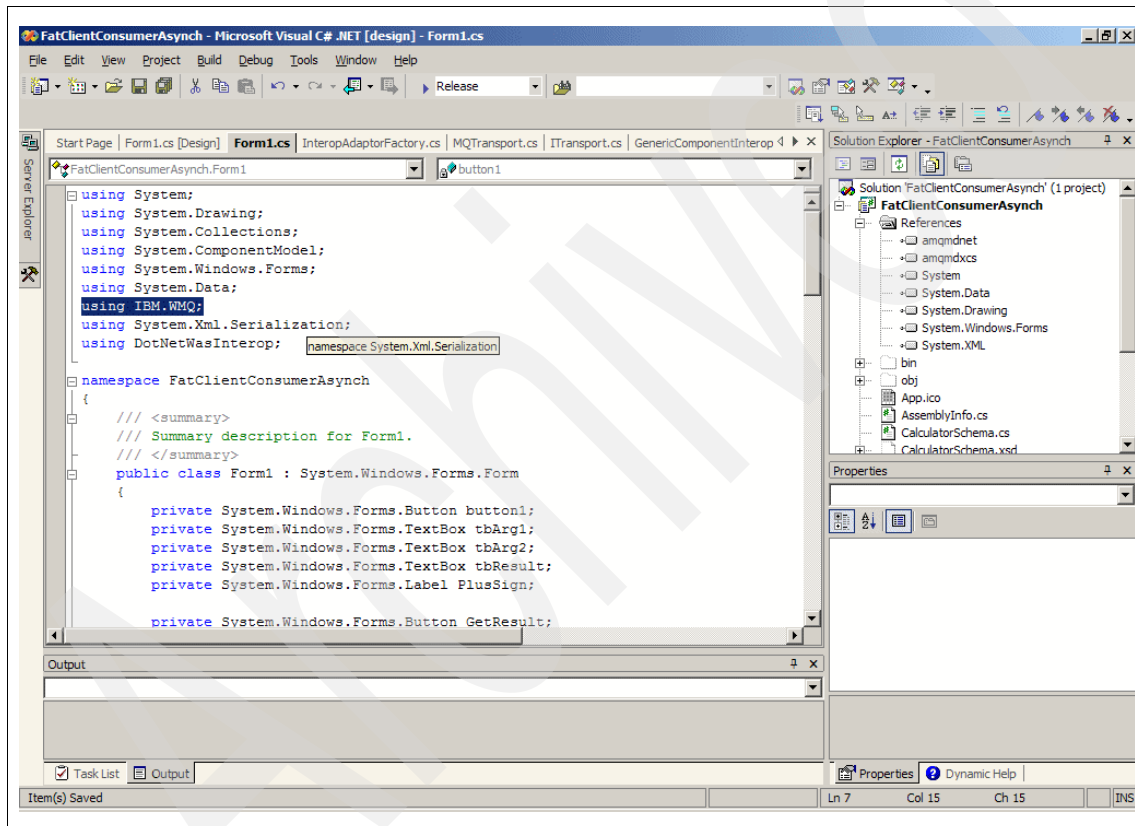


Figure 6-25 References required for the use of MQ classes for .NET

Modify the Windows Form client to use the interoperability adapter

The code that resides in the class Form1.cs is invoked when the user clicks any of the three buttons: button1, button2, and GetResult. As mentioned above at the beginning of this section, the code for button1 and button2 is very similar, except that the command is sent as part of the message (Add and Subtract respectively). This class has a member variable with the name adaptor of type IInteropAdaptor that is initialized with the actual adaptor reference when the Windows Form loads:

```
    adaptor =  
    InteropAdaptorFactory.createInteropAdaptor(InteropAdaptorFactory.InteropAdaptor  
    Type.Messaging, InteropAdaptorFactory.SerializationType.XML);
```

The code for button1 is given in Example 6-12.

Example 6-12 Sending a message by calling the interoperability adaptor

```
private void button1_Click(object sender, System.EventArgs e)  
{  
    CalculatorInput input = new CalculatorInput();  
    input.Input1 = Convert.ToSingle(tbArg1.Text);  
    input.Input2 = Convert.ToSingle(tbArg2.Text);  
    input.Command = "ADD";  
    adaptor.execute(new Object[]{input},  
    InteropAdaptorFactory.CallType.Request);  
}
```

The code for GetResult is given in Example 6-13. Note that to deserialize the XML message from WebSphere into a valid Java Object, response attributes (in the form of element name and type) have to be specified. For more details on this topic, refer to the .NET documentation on XML serialization/deserialization.

Example 6-13 Getting a message by calling the interoperability adaptor

```
private void GetResult_Click(object sender, System.EventArgs e)  
{  
    // Specify the response attributes in XML based on Schema  
    XmlRootAttribute xRoot= new XmlRootAttribute();  
    xRoot.ElementName = "Result";  
    xRoot.Namespace="http://tempuri.org/XMLSchema.xsd";  
    adaptor.setResponseAttributes(new Single(), xRoot);  
    Object response = adaptor.execute(new  
    Object[] {}, InteropAdaptorFactory.CallType.Response);  
    tbResult.Text=""+response+"\r\n";  
}
```

6.6 Developing the WebSphere calculator component

We assume that the business logic for the Calculator application is pre-existing code and is contained in the Calculator.jar file. So, in order to enable the Calculator code as an asynchronous service, we need to create a message driven bean front end to the application.

Develop WebSphere Service Interface using MDB

We need the Calculator component to be activated upon receipt of a request message. This is achieved using Message Driven Beans (MDBs). Message Driven Beans are stateless, server-side, transaction-aware components for processing asynchronous JMS messages.

Our MDB will listen on a queue for receipt of a message. Upon receipt, the `onMessage()` method of the bean is automatically executed. While the MDB is responsible for the actual processing of the message, quality of service items, such as transactions, security, resources, concurrency, and message acknowledgement are all handled automatically by the bean's container, enabling us to focus on implementing the actual business logic.

We shall assume for simplicity of discussion that the only methods of the Calculator class we are interested in will be the `add(float,float)` method and the `subtract(float, float)` method, and the message format being sent by the consumer will conform to the CalculatorInput schema described in 6.4.2, "Create the XML schema for messages" on page 219.

Creating the Message Driven Bean is achieved by using wizards in the IBM Rational Application Developer for WebSphere Software. The following are the steps for creating the MDB:

1. Create a new EJB project (J2EE V1.4) by selecting **File** → **New** → **EJB Project** (Figure 6-26 on page 235). Enter the name ComponentInteropEJB, as shown in Figure 6-27 on page 236.

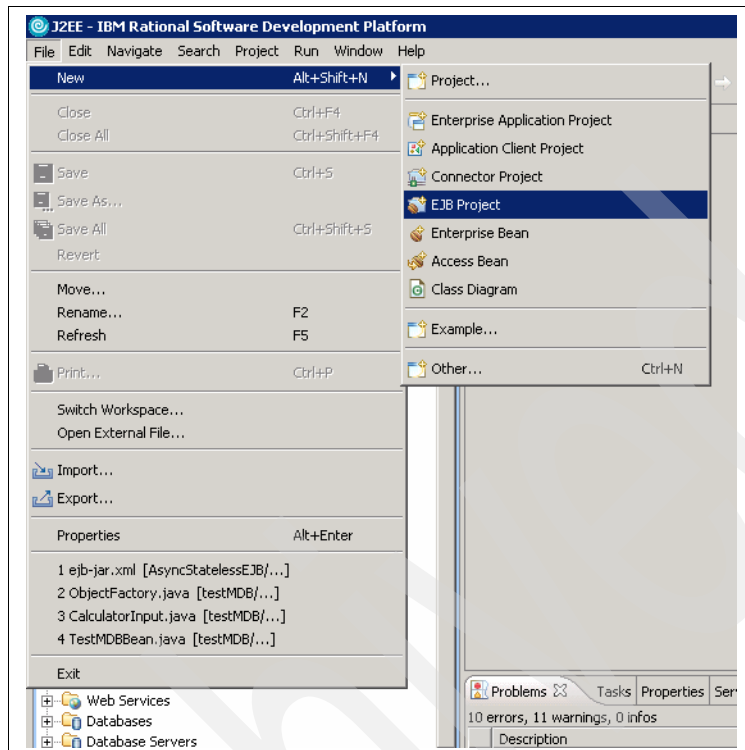


Figure 6-26 Creating a new EJB project in Rational Software Architect

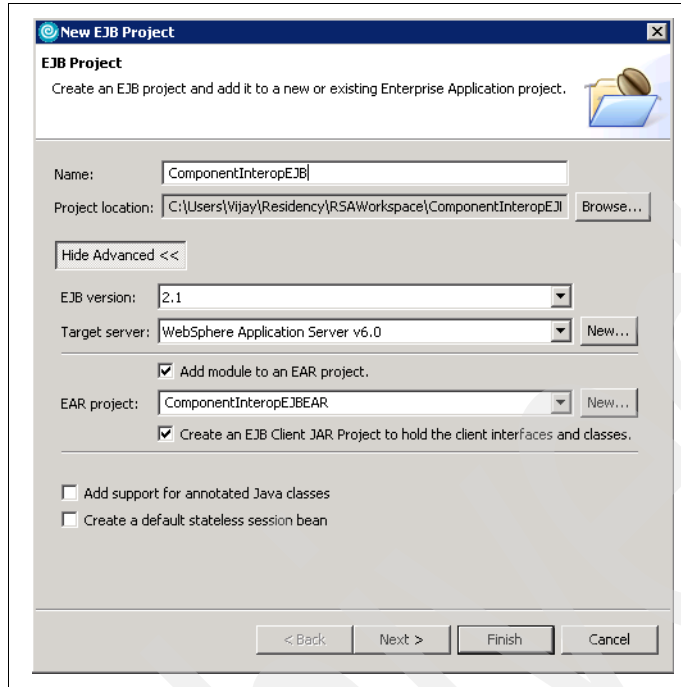


Figure 6-27 Creating a new EJB project - entering the project name

2. Right-click the **EJB project**, and from the context menu, select **New** → **Other** → **EJB** → **Enterprise Bean**. This is shown in Figure 6-28 on page 237. Ensure the correct EJB project is selected, and on the subsequent window, you should select **Message-driven bean**. Enter the appropriate values for the Bean name and Default package, for example, TestMDBBean and redbook.wasdnet.compinterop and click **Next**. This is shown in Figure 6-29 on page 237. Accept all default settings on the next few windows. Click **Finish**. You will see that a new message driven bean has been created, and within the bean there is a method `onMessage(javax.jms.Message msg)`. It is this method that will be executed automatically when a message is detected on the queue.

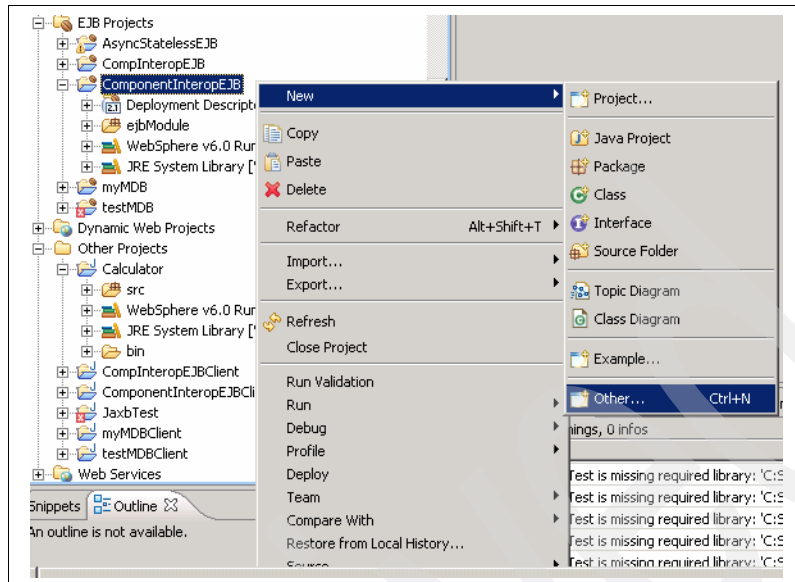


Figure 6-28 Creating a new EJB

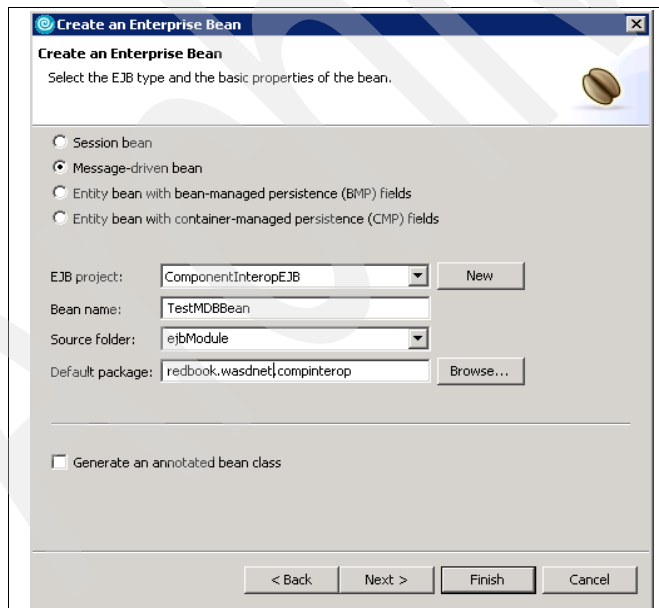


Figure 6-29 Specifying the EJB type, the bean name, and default package name

3. We want the business logic to call the `add(float, float)` and `subtract(float, float)` methods of the `CalculatorService` class in the `Calculator.jar` file. The EJB project needs to reference the `Calculator.jar` file, so to open up the properties page for the project, select **Java build path** → **Add External JARs** and add `Calculator.jar` to the build path.
4. Add the two jars under *<WebSphere Application Server Install>* `Root\lib\WMQ\java\lib` to the build path for setting the target client as MQ for `replyQ`.
5. Also, add the following JAXB libraries (for automated deserialization/serialization of XML into Java objects and vice versa) to the build path:
 - `jaxb-api.jar`
 - `jaxb-impl.jar`
 - `jaxb-libs.jar`
 - `namespace.jar`
 - `relaxngDatatype.jar`
 - `xsdlib.jar`

The Java build path should look like as shown in Figure 6-30 on page 239.

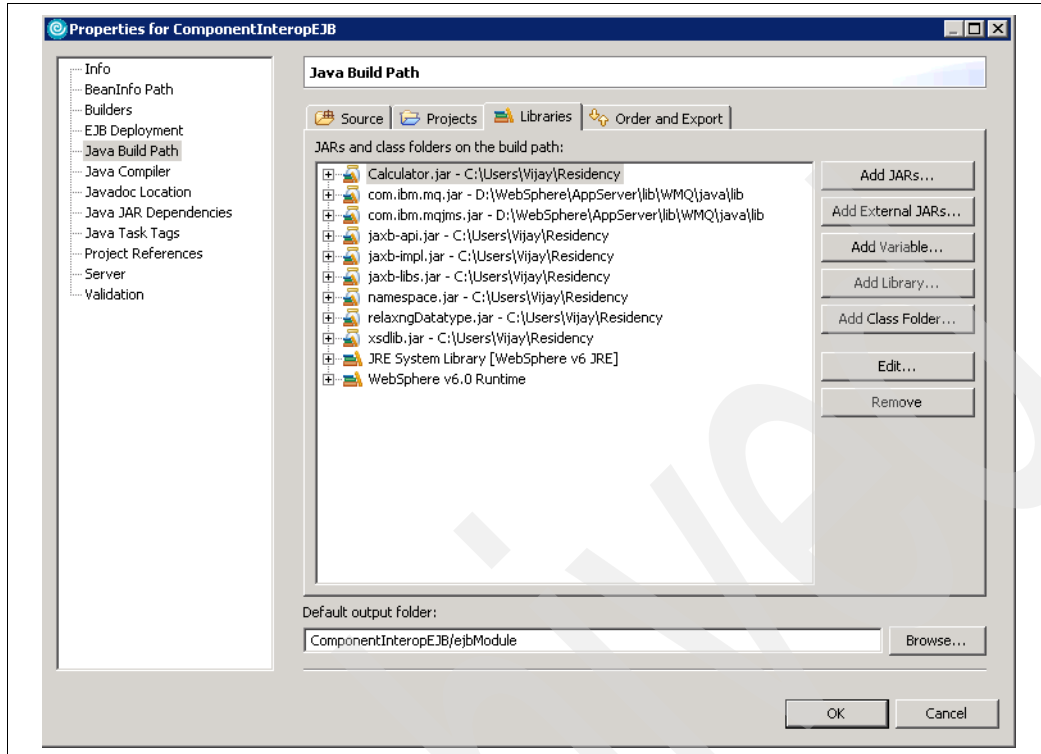


Figure 6-30 Adding Calculator.jar, WMQ jars and JAXB libraries to the build path for the project

6. Import the Java files that were generated from the XML schema (CalculatorSchema.xsd) using the JAXB compiler in 6.4.4, “Generate the Java classes corresponding to the XML schema” on page 221. To do this, make sure that ComponentInteropEJB project is selected and then select **File** → **Import** → **File system** and browse to the correct folder (<JAXB_CLASSES_DESTINATION_DIR> in 6.4.4, “Generate the Java

classes corresponding to the XML schema” on page 221), as shown in Figure 6-31. Click **Finish**.

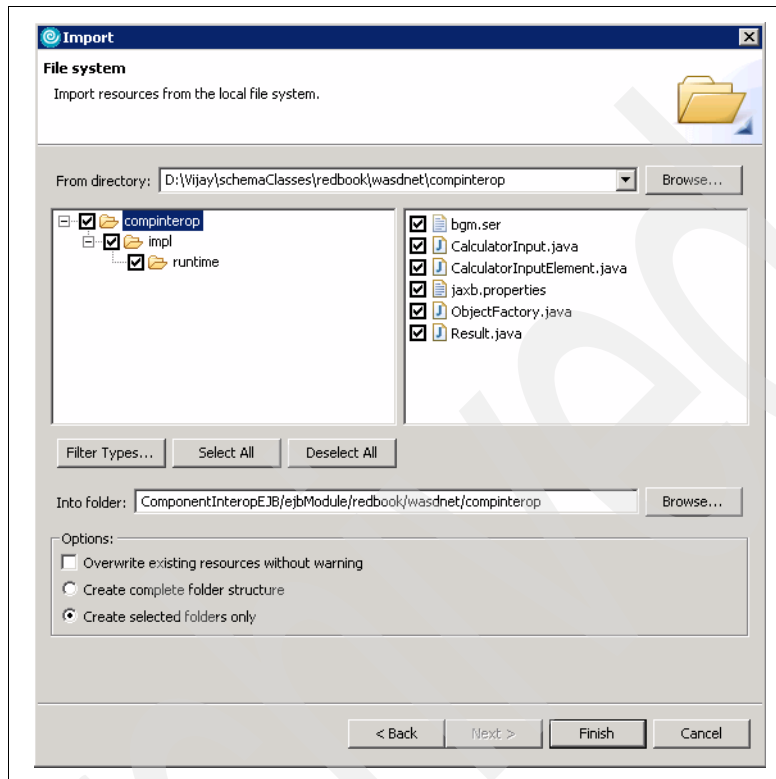


Figure 6-31 Importing the files that were generated from the XML schema using JAXB

7. Back in the `onMessage()` method, enter the code shown in Example 6-14.

Example 6-14 Implementation of the `onMessage` method

```
public void onMessage(javax.jms.Message msg) {
    CalculatorService s = new CalculatorService();
    try {
        JAXBContext jc = JAXBContext.newInstance("redbook.wasdnet.compinterop");
        Unmarshaller unmarshaller = jc.createUnmarshaller();
        TextMessage m = (TextMessage)msg;
        System.out.println("[TestMDBBean::onMessage] Got new message
        :"+m.getText());
        CalculatorInputElement ci =
        (CalculatorInputElement)unmarshaller.unmarshal(new
        StringBufferInputStream(m.getText()));
        float a1 = ci.getInput1();
        float a2 = ci.getInput2();
    }
}
```

```

        float output=0;

        String command = ci.getCommand();
        System.out.println(" [TestMDBBean::onMessage]Args: a1="+a1+" a2="+a2+"
command= "+command);
        Result result = new ResultImpl();

        if(command.equals("ADD"))
        {
            output = s.add(a1,a2);
        }

        if(command.equals("SUBTRACT"))
        {
            output = s.subtract(a1,a2);
        }
        ....
        ....
    }

```

The `javax.jms.Message` received from the queue is cast into a `javax.jms.TextMessage`, as the data format being sent is an XML String consisting of `CalculatorInput` complex type, which in turn comprises two float values and a command. The `getText()` method retrieves the actual XML text of the message, and JAXB library calls are used to unmarshal the expected Java class (`CalculatorInput`). The unmarshaller class in JAXB uses UTF-8 encoding by default. Therefore, the sender should ensure that any data intended for JAXB unmarshalling should use UTF-8 encoding (as shown in Example 6-11 on page 229). For more information about using JAXB, refer to the JAXB documentation:

<http://java.sun.com/webservices/jaxb/>

Another good article that explains the use of JAXB:

<http://java.sun.com/webservices/jaxb/users-guide/jaxb-works.html>

Once the float values have been read in from the message, the `Calculator` back-end code can be invoked by calling the `add(float, float)` or the `subtract(float, float)` method on the `CalculatorService` class.

The code example above simply prints out the result of the addition/subtraction, but for a full asynchronous solution, the result would need to be placed onto a reply queue. The reply queue expected by the consumer will be specified in the original message, and can be determined with the following line of code:

```
Queue replyTo = (Queue) msg.getJMSReplyTo();
```

This replyTo queue can now be used by the service, and the result can be put (as a new TextMessage) onto this queue. The consumer can then retrieve the message from the reply queue at a later time.

The code to put the result on a reply queue is shown below in Example 6-15; insert the code after the two if blocks in the onMessage() method.

Note: Two things to note here are:

- ▶ We programmatically set the target client for replyQ to be MQ (using the value 1 for setTarget call) in addition to specifying it in the administrative settings for replyQ.
- ▶ JAXB marshaller is set to use UTF-16 encoding so that any consumer that uses UTF-16 encoding (like the .NET client in our solution) will be able to process the data directly.

Example 6-15 How to put the result onto the reply queue

```
//put the reply onto the reply queue

    InitialContext ctx = new InitialContext();

    QueueConnectionFactory qcf =
    (QueueConnectionFactory)ctx.lookup("jms/redbookClientQCF");

    QueueConnection qConn = qcf.createQueueConnection();
    System.out.println("[TestMDBBean::onMessage] created QueueConnection");
    QueueSession session = qConn.createQueueSession(false,
    Session.AUTO_ACKNOWLEDGE);
    System.out.println("[TestMDBBean::onMessage] created QueueSession");
    Queue queue = (Queue)msg.getJMSReplyTo();
    com.ibm.mq.jms.MQQueue q = (com.ibm.mq.jms.MQQueue) queue;
    System.out.println("[TestMDBBean::onMessage]
    replyToQueue="+queue.getQueueName()+" targetClient="+q.getTargetClient());
    q.setTargetClient(1);

    QueueSender sender = session.createSender(queue);
    System.out.println("[TestMDBBean::onMessage] created QueueSender");
    //send the result
    qConn.start();
    TextMessage mReply = session.createTextMessage();
    mReply.setJMSMessageID(m.getJMSMessageID());
    //call the add method of the calculator service
    Marshaller marshaller = jc.createMarshaller();
    /* JAXB's default encoding is UTF-8 whereas strings in .NET use UTF-16
    * encoding. Therefore a transformation is required.
    * /
    marshaller.setProperty(Marshaller.JAXB_ENCODING,"UTF-16");
```

```

        StringWriter sw = new StringWriter();

        result.setValue(output);
        System.out.println(output);

        marshaller.marshal(result,sw);
        mReply.setText(sw.toString());
        sender.send(mReply);
        System.out.println("[TestMDBBean::onMessage] sent the result
"+sw.toString());
        qConn.close();
        sender.close();
        session.close();

    } catch (JMSEException e) {
        e.printStackTrace();
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (JAXBException e){
        System.err.println("Got an exception while marshalling/unmarshalling
XML: "+e);
    } catch (JMSEException e) {
        e.printStackTrace();
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (JAXBException e){
        System.err.println("Got an exception while marshalling/unmarshalling
XML: "+e);
    }
}

```

The reply message, mReply, will sit in the reply message queue until it is retrieved by the consumer.

The consumer in this case is running in a .NET environment using the WebSphere MQ Classes for .NET.

Make sure you add the following import statements shown in Example 6-16 to the class.

Example 6-16 Import statements to be added

```
import java.io.StringBufferInputStream;
import java.io.StringWriter;
import javax.jms.JMSException;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;
import redbook.wasdnet.compinterop.impl.*;
import redbook.coex.sall.business.CalculatorService;
```

Export the application EAR by right-clicking the EAR project **ComponentInteropEJBear** and selecting **Export** → **EAR**. Enter the name and location of the desired EAR file and click **Finish**.

6.7 Testing the sample application

To test the sample application, do the following:

1. Copy the JAXB libraries (jaxb-api.jar, jaxb-impl.jar, jaxb-libs.jar, namespace.jar, relaxngDatatype.jar, xsdlib.jar) and Calculator.jar to *<WebSphere Application Server Install Root>\classes*.
2. Start the WebSphere Application Server and deploy the enterprise application archive (EAR) on WebSphere Application Server by selecting **Applications** → **Install New Application** and selecting the EAR file for deployment. Accept the default settings for the next couple of screens and keep clicking **Next**. On the “Provide listener bindings for message-driven beans” screen, specify the name of the Listener port as TestLP (as shown in Figure 6-32 on page 245). Click **Next** and finally click **Finish**. Save your changes and then start the application.

Attention: WebSphere Application Server Version 6 comes with WebSphere MQ libraries (com.ibm.mq.jar and com.ibm.mqjms.jar) that can be found in *<WebSphere Application Server Install Root>\lib\WMQ\java\lib*. Depending on the WebSphere Application Server version you have, the version of WebSphere MQ libraries that come with WebSphere Application Server might change. If you face problems starting your application, it could be due to a version mismatch, and you might have to copy the WebSphere MQ libraries from *<WebSphere MQ Install Root>\Java\lib* into *<WebSphere Application Server Install Root>\lib\WMQ\java\lib*.

Enterprise Applications Close page

Install New Application

Specify options for installing enterprise applications and modules.

Step 3: Provide listener bindings For message-driven beans

Each message-driven enterprise bean in your application or module must be bound to a listener port name or to an activation specification JNDI name. When a message-driven enterprise bean is bound to an activation specification JNDI name you may also specify destination JNDI name and authentication alias.

☒ Apply Multiple Mappings

Select	EJB module	EJB	URI	Messaging Type	Bindings
<input type="checkbox"/>	ComponentInteropEJB	TestMDBBean	ComponentInteropEJB.jar,META-INF/ejb-jar.xml	javax.jms.MessageListener	<input checked="" type="radio"/> Listener port Name: <input type="text" value="TestLP"/> <input type="radio"/> Activation Specification JNDI name: <input type="text"/> Destination JNDI Name: <input type="text"/> ActivationSpec Authentication Alias: <input type="text"/>

Previous Next Cancel

Figure 6-32 Specifying the name of the listener port while deploying the EAR

3. Start the queue manager on WebSphere MQ using the following command:
`<WebSphereMQ_INSTALL_ROOT>\bin\strmqm redbookQM`
4. If the MQ listener has not already started, start the MQ listener by using the following command:
`<WebSphereMQ_INSTALL_ROOT>\bin\runmqlsr.exe -m redbookQM -t TCP`

5. Run the .NET Windows application you have just developed. Enter a value into the two arguments textbox, then click **Add**, as shown in Figure 6-33. Then click **Get result** and you should see the results in the textbox.

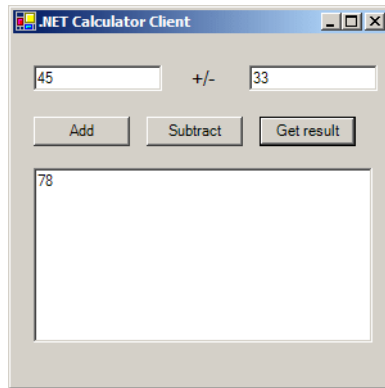


Figure 6-33 Running the .NET application

6.7.1 Troubleshooting

The following are some points concerning troubleshooting:

- ▶ If you observe that clicking **Get result** shows RFH in the response text box, it means that the message received from the WebSphere service includes the JMS header. To overcome this, ensure that the TargetClient setting for replyQ is set to MQ.
- ▶ If you see an error message on the .NET side that has the reason code 2059, it means that the MQ client on .NET was not able to contact the MQ server, as there is no listener process running there. It can happen either because the MQ listener on the WebSphere Application Server node is not running or it is not running on the default port (1414) or the MQServer environment variable on .NET node does not have the correct machine name and channel name. Ensure that none of these is the case.
- ▶ If you see an error message on the .NET side that has the reason code 2035, it means that the .NET user's user ID is not authorized to access the queue on the remote (or local) queue manager. Ensure that the environment variable *MQSNOAUT=yes* has been defined before the creation of queue manager on the WebSphere Application Server node. You can also consider using the **setmqaut** command to grant access to a user ID to a queue.



Part 3

Web Services interoperability

ARCHIVED

Introduction to Web Services Interoperability

In this chapter we provide an overview of Web Services and Services Oriented Architecture (SOA). We also introduce Web Services Interoperability, the organization that defines profiles that ensures the best interoperability for Web Services. And finally, we introduce architecture for Web Services Interoperability solutions.

This chapter contains the following sections:

- ▶ Introduction to Web Services
- ▶ Overview of Web Services Interoperability

7.1 Introduction to Web Services

Web Services is emerging as the next evolution of the use of Web technology to drive operational efficiencies and interoperability within the enterprise and across the extended enterprise. The World Wide Web ushered in a new era in conducting electronic business. Initially, it provided the means to establish a Web presence where an enterprise used the World Wide Web to market its products and services and to reach its customers. With the emergence of e-commerce, the use of Web technologies created a whole new sales channel, enabling business-to-consumer and business-to-business buying and selling. E-business extended the use of Web technologies to drive business processes, enabling collaboration with business partners and suppliers. With e-business, companies attained new operational efficiencies and integrated across operational boundaries. E-business operations were driven mainly through program-to-user interaction. Web Services is driving further operational efficiencies and integration through program-to-program interaction.

7.1.1 Web Services background

The W3C Web Services Architecture Working Group defined a Web Service as:

“A software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via internet-based protocols.”¹

This precise definition provides the foundation for a language-neutral and environment-neutral programming model.

The programming model consists of three key components:

- ▶ A service model for integrating loosely coupled network accessible resources
- ▶ A set of specifications that define protocols and standardized application programming interfaces (APIs)
- ▶ Distributed architecture model for creating distributed, scalable, and integrated applications using loosely coupled components and messaging

¹ <http://www.w3.org/TR/2002/WD-wsa-reqs-20020429>

7.1.2 Web Services model

The Web Services model provides one of the three key components of the language-neutral and environment-neutral Web Services programming model. From a systems perspective, the Web Services model is based on three key roles and a set of operations executed by and between the three roles.

The roles in a Web Services model are as follows:

- ▶ Service provider

The service provider hosts (usually is also the creator of) network-accessible software components that implement the Web Services. The service provider publishes the description (using Web Services Description Language) of the Web Services to service registries.

- ▶ Service requestor

Service requestors consume Web Services. Requestors must first find and retrieve the service description for the desired Web Service from the service registry (can also be passed locally), and then bind to the Web Service before invoking operations on the Web Service. Service requestors include users running Web browsers, program without a user interface, and other Web Services. The commonly used protocol for the exchange of information including lookup, retrieval, and binding is the Simple Object Access Protocol (SOAP).

- ▶ Service registry

The service registry is the repository of service descriptions published by service providers. Service requestors search the repository for services and binding information. At development time, service requestors have the option to statically bind to services; however, at runtime, the binding is dynamic. Universal Description, Discovery, and Integration (UDDI) is the Internet based service registry developed by industry wide initiative.

Figure 7-1 shows an overview of the Web Services model.

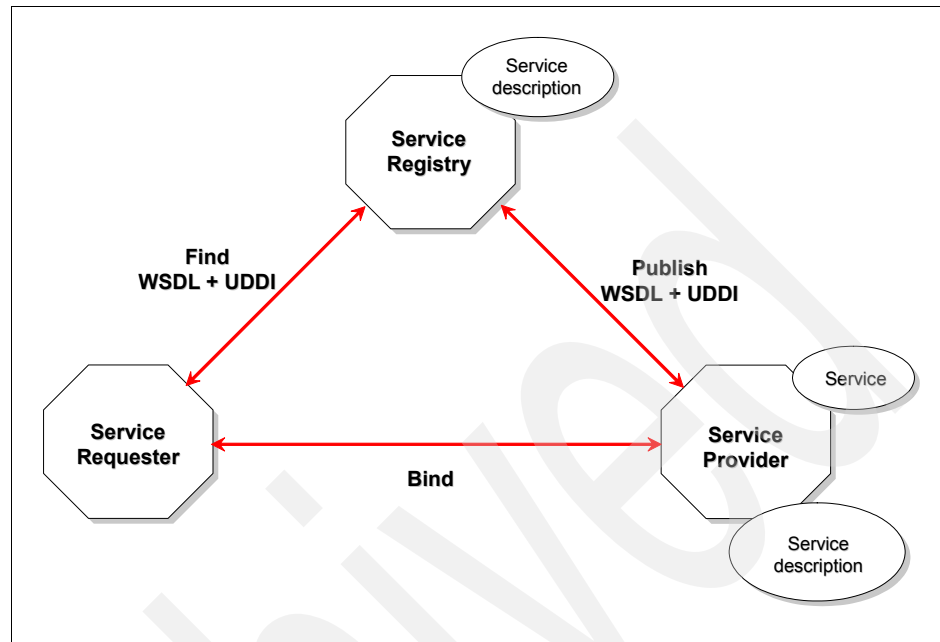


Figure 7-1 Web Services model

Three operations drive the flow and behavior in the Web Services model. The operations are:

- Publish

The provider of a Web Service creates the Web Services Description Language (WSDL), which describes the interface to the service. The WSDL definition include data type definitions, input and output messages, the operations, and communication endpoints. The publish operation is the process of making the WSDL available so service requestors can find and bind to the Web Service through the WSDL. The publication could be to a central location, such as UDDI, or distributed to practically any existing Web infrastructure using Web Services Inspection Language (WSIL) documents.

- Find

The service requestor must perform a find operation in order to discover and retrieve the WSDL for published services. The find operation can be performed either directly or through queries to the service registry, such as UDDI. The discovery can be performed at design time where the service description is statically incorporated to the program development or it can be dynamically discovered at runtime.

► Bind

Before invoking or initiating a request with a Web Service, the service requestor makes use of the binding information from the retrieved WSDL to bind to the Web Service.

7.1.3 Web Services specifications

The second component of the language and environment-neutral Web Services programming model is the collection of Extensible Markup Language (XML) based specifications that define protocols for describing, delivering, and interacting with Web Services. Figure 7-2 is just one of the possible visualizations showing different categories that fit in a Web Services programming stack.

Note: At any point in time, versions of Web Services specifications are in various stages of development and approval.

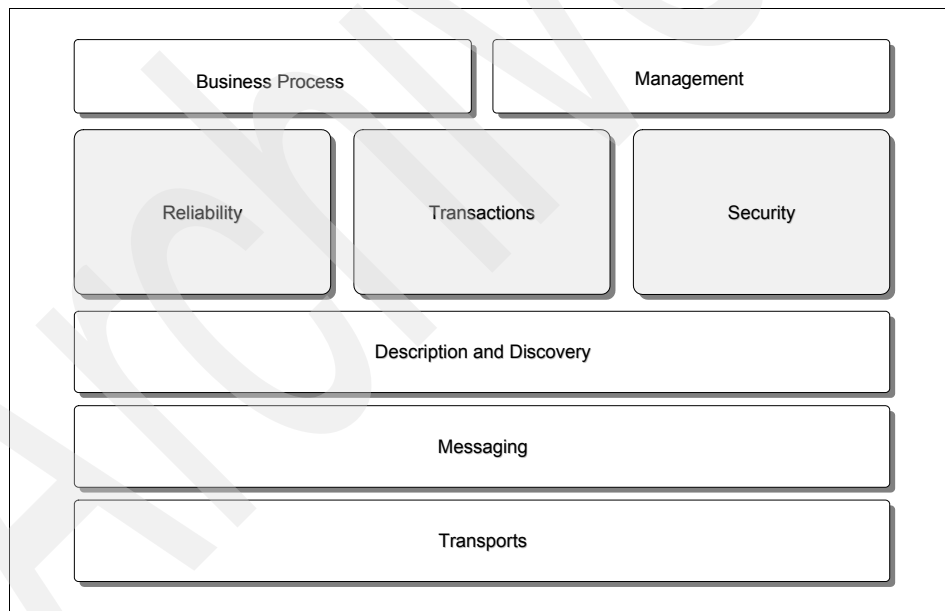


Figure 7-2 Web Services categories

Transports

The transports is the foundation of the Web Services programming stack. Included in this category are protocols that support the network accessibility of Web Services. Transport protocols include HTTP, FTP, e-mail, IIOP, messaging middleware, and so on.

Messaging

The messaging specifications define the standards for exchanging information in the Web Services distributed environment.

SOAP

Simple Object Access Protocol (SOAP) provides the definition of the structured and typed XML-based information that is exchanged between parties in a distributed environment. SOAP messages are self-contained and describe the structure and type of the information they carry within the message.

Each SOAP message consists of an envelope that contains an arbitrary number of headers and one body that carries the payload. SOAP messages might contain exceptions to report failures or unexpected conditions.

Even though SOAP implements a stateless, one-way paradigm, it can be used to implement more complex paradigms, such as request/response and solicit/response. Example 7-1 shows the structure of SOAP messages.

Example 7-1 Structure of SOAP messages

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    ...
  </env:Header>
  <env:Body>
    ...
  </env:Body>
</env:Envelope>
```

Figure 7-3 on page 255 shows the SOAP message structure.

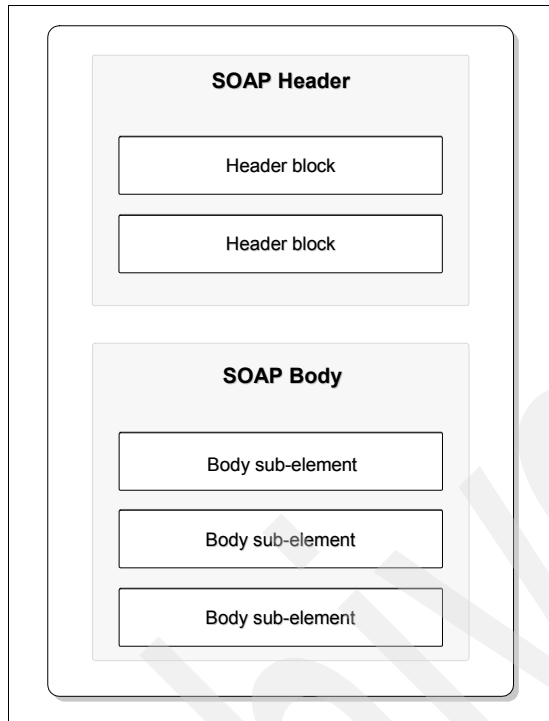


Figure 7-3 SOAP message structure

Web Services Addressing

Web Services Addressing (WS-Addressing) provides transport-neutral mechanisms to address Web Services and messages. Specifically, this specification defines XML elements to identify Web Services endpoints and to secure end-to-end endpoint identification in messages.

Web Services Notification

The Web Services Notification (WS-Notification) is a family of specifications that include "Publish-Subscribe Notification for Web Services" and the following normative specifications:

- **WS-BaseNotification**

Defines the Web Services interfaces for NotificationProducers and NotificationConsumers. It includes standard message exchanges to be implemented by service providers that wish to act in these roles, along with the operational requirements expected of them. This is the base specification on which the other WS-Notification specification documents depend.

- ▶ **WS-BrokeredNotification**

Defines the Web Services interface for the NotificationBroker. A NotificationBroker is an intermediary which, among other things, allows publication of messages from entities that are not themselves service providers.

- ▶ **WS-Topics**

Defines a mechanism to organize and categorize items of interest for subscription known as "topics." These are used in conjunction with the notification mechanisms defined in WS-BaseNotification.

Web Services attachments profile

These are a set of specifications, with respective clarifications and amplifications, that promote interoperable SOAP Messages with Attachments.

Description and discovery

These specifications standardize the description of Web Services, the creators and providers, and the technical interfaces that enable the Web Services to be discovered:

- ▶ **UDDI**

The Universal Description, Discovery, and Integration (UDDI) standard defines the means to publish and to discover Web Services. Of the three Web Services standards, this is the least important one, because one can also implement and deploy Web Services without UDDI. However, in certain situations, UDDI is a must.

- ▶ **WSDL**

The Web Service Definition Language (WSDL) describe Web Services as abstract service endpoints that operate on messages. Both the operations and the messages are defined in an abstract manner, while the actual protocol used to carry the message and the endpoint's address are concrete.

WSDL is not bound to any particular protocol or network service. It can be extended to support many different message formats and network protocols. However, because Web Services are mainly implemented using SOAP and HTTP, the corresponding bindings are part of this standard.

- ▶ **Web Services Semantics (WSDL-S)**

A specification submission that defines how to add semantic information to WSDL documents. Semantic annotations define the meaning of the inputs, outputs, preconditions, and effects of the operations described in a service interface.

► Web Services Metadata Exchange

This specification enables efficient, incremental retrieval of a Web service's metadata. It defines three request-response message pairs to bootstrap communication with a Web Service:

- Retrieve the WS-Policy associated with the receiving endpoint or with a given target namespace
- Retrieve either the WSDL associated with the receiving endpoint or with a given target namespace
- Retrieve the XML Schema with a given target namespace

► Web Services Policy Assertions Language

WS-PolicyAssertions is a building block that is used in conjunction with other Web Services and application-specific protocols to accommodate a wide variety of policy exchange models.

► Web Services Policy Attachment

This specification defines general-purpose mechanisms for associating policies with the subjects to which they apply. Specifically, it defines the following:

- How to reference policies from WSDL definitions
- How to associate policies with deployed Web service endpoints
- How to associate policies with UDDI entities

► Web Services Policy Framework

Provides a general purpose model and syntax to describe and communicate the policies of a Web Service that enable Web Services providers and consumers to discover the capabilities and constraints to facilitate interoperability of these services.

► Web Services Resource Framework

The Web Services Resource Framework defines a family of specifications for accessing stateful resources using Web Services. It includes the WS-ResourceProperties, WS-ResourceLifetime, WS-BaseFaults, and WS-ServiceGroup specifications.

Reliability

Reliability specifications ensure that messages are delivered reliably in the presence of software component, system, or network failures.

Web Services Reliable Messaging

Web Services Reliable Messaging (WS-ReliableMessaging) defines a protocol and a set of mechanisms that ensure messages are delivered reliably between two endpoints and support a broad spectrum of delivery assurance and robust services. The protocol, as defined, is independent of the network transport technologies; however, it specifies SOAP binding for Web Services interoperability.

Transactions

The Web Services Transactions specifications define mechanisms for transactional interoperability. It describes an extensible coordination framework (WS-Coordination) and specific coordination types for short duration transactions (WS-AtomicTransaction) and long running business transactions (WS-BusinessActivity):

- ▶ **WS-Coordination**
Describes an extensible framework for providing protocols that coordinate distributed activities.
- ▶ **WS-AtomicTransaction**
Defines three specific agreement coordination protocols for the atomic transaction coordination: completion, volatile two-phase commit, and durable two-phase commit.
- ▶ **WS-BusinessActivity**
Defines two specific agreement coordination protocols for the business activity coordinations `BusinessAgreementWithParticipantCompletion` and `BusinessAgreementWithCoordinatorCompletion`.

Security

Using these security specifications, applications can engage in secure communication designed to work with the general Web Services framework. The specifications define mechanisms for a wide variety of security models and encryption technologies:

- ▶ **XML Signature**
The XML Signature specification defines the XML syntax and processing rules to sign and verify digital signatures for digital content.

► XML Encryption

This standard specifies a process for encrypting data and representing the result in XML. The data can be arbitrary data (including an XML document), an XML element, or an XML element content. The result of encrypting data is an XML encryption element that contains or references the cipher data.

► XML Key Management (XKMS)

The XML Key Management Specification (XKMS) specifies protocols for distributing and registering public keys. The specification is comprised of two parts:

– XML Key Information Service Specification (X-KISS)

X-KISS defines the protocol for delegating resolution of Key Information associated with an XML signature.

– XML Key Registration Service Specification (X-KRSS)

XML Key Registration Service Specification (X-KRSS) defines the protocol for delegating registration of public key information. X-KRSS supports the public key life cycle, registration, renewal, revocation, recovery, and roaming.

► WS-Security

The WS-Security specification describes extensions to SOAP that allow for quality of protection of SOAP messages. This includes, but is not limited to, message authentication, message integrity, and message confidentiality. The specified mechanisms can be used to accommodate a wide variety of security models and encryption technologies. It also provides a general-purpose mechanism for associating security tokens with message content.

► WS-Trust

The WS-Trust specification defines primitives and extensions for the issuance, exchange, and validation of security tokens. It also enables the issuance and dissemination of credentials within different trust domains. WS-Trust relies on the secure messaging mechanisms of WS-Security.

► WS-SecureConversation

The WS-SecureConversation specification defines extensions for the establishment and sharing of security contexts, and session key derivation to enable a secure conversation. The security context is established when multiple communicating parties wish to exchange multiple messages. It exists for the duration of the communications session. WS-SecureConversation relies on the mechanisms defined by WS-Security and WS-Trust.

► WS-Federation

The WS-Federation specification defines mechanisms that enable different security realms to federate by allowing and brokering trust of identities, attributes, and authentication between participating Web Services. The mechanisms support both passive and active requestors.

– Active Requester Profile

The WS-Federation Active Requester Profile specification defines how the cross trust realm identity, authentication, and authorization federation mechanisms defined in WS-Federation, are used by active requestors, such as SOAP-enabled applications.

– Passive Requester Profile

The WS-Federation Passive Requester Profile specification defines how the WS-Federation model of cross trust realm identity, authentication, and authorization federation mechanisms is applied to passive requestors, such as Web browsers that support the HTTP protocol.

► WS-Security Kerberos Binding

WS-SecurityKerberos describes basic usage patterns that leverages existing Kerberos security infrastructure and integrates Kerberos security environments with the Web service security architecture.

► WS-Policy

WS-Policy defines the framework for associating constraints and requirements expressed as policy assertions with Web Services. It also defines a base set of assertions that describe how messages are to be secured.

Business processes

Business process specifications define how operations from a collection of Web Services from multiple services and organizations participate in the execution of business processes. The specification include the data shared between the Web Services, how participants are involved, and the joint exception handling.

► Business Process Execution Language for Web Services (BPEL4WS)

BPEL4WS provides a language for the formal specification of business processes and business interaction protocols. It extends the Web Services interaction model to enable support for business transactions.

► Web Services Choreography Description Language (WS-CDL)

The WS-CDL specification is not an "executable business process description language" or an implementation language like BPEL4WS; it describes collaboration between participants, regardless of the platform or programming model.

Management

Web Services manageability specifies the set of capabilities for discovering the existence, availability, health, performance, and usage, as well as the control and configuration of Web Services.

WS-Manageability

WS-Manageability consists of concept, normative specification, and representations.

- ▶ WS-Manageability concept
Provides an overview of Web Services architecture and implications for manageability and the role of the manager in the Web Services architecture, manageability implementation patterns, and discovery considerations.
- ▶ WS-Manageability specification
Defines manageability model in terms of manageability topics, (identification, configuration, state, metrics, and relationships) and the aspects (properties, operations, and events).
- ▶ WS-Manageability representations
Defines WSDL interfaces for accessing the manageability model for Web Services.

7.1.4 Web Services architecture model

The Web Services model and the Web Services specifications provide the functional building blocks and define the characteristics that form the basis for the Web Services architecture model:

- ▶ Loose coupling
Web Services are modular applications that can be published, located, and invoked across the Web.
- ▶ Self-contained and self-describing
It encapsulates functionality that is exposed through interfaces accessible via standard Internet communication protocols.
- ▶ Language-independent and interoperable
The XML-based interfaces, service description, and protocols enable platform and language independence and interoperability between services realized on different underlying infrastructures.

- Dynamic discovery and binding

Dynamic service discovery and invocation (publish, find, and bind) provide the just-in-time service discovery framework for distributed scalable system architecture.

The Web Services architecture model is a realization of Service-Oriented Architecture (SOA) software design principles. SOA is comprised of interface definitions loosely coupled to software modules, which represent complete functions accessible by name via an interface, and the relationship between the services and the service consumers.

7.2 Overview of Web Services Interoperability

The language and environment-neutral features of Web Services coupled with the Web Services specifications were designed to enable improved application interoperability. In practice, however, simply adhering to Web Services specifications does not ensure seamless interoperability.

The Web Services Interoperability (WS-I) organization, an open industry group composed of global leading solution providers, was established with the mission of promoting Web Services interoperability. It delivers resources that enable consistent Web Services implementation, promotion of Web Services adoption, and acceleration of Web Services deployment. WS-I deliverable includes profiles, sample applications, and testing tools that improved interoperability between Web Services. Figure 7-4 on page 263 gives an overview of the WS-I deliverables.

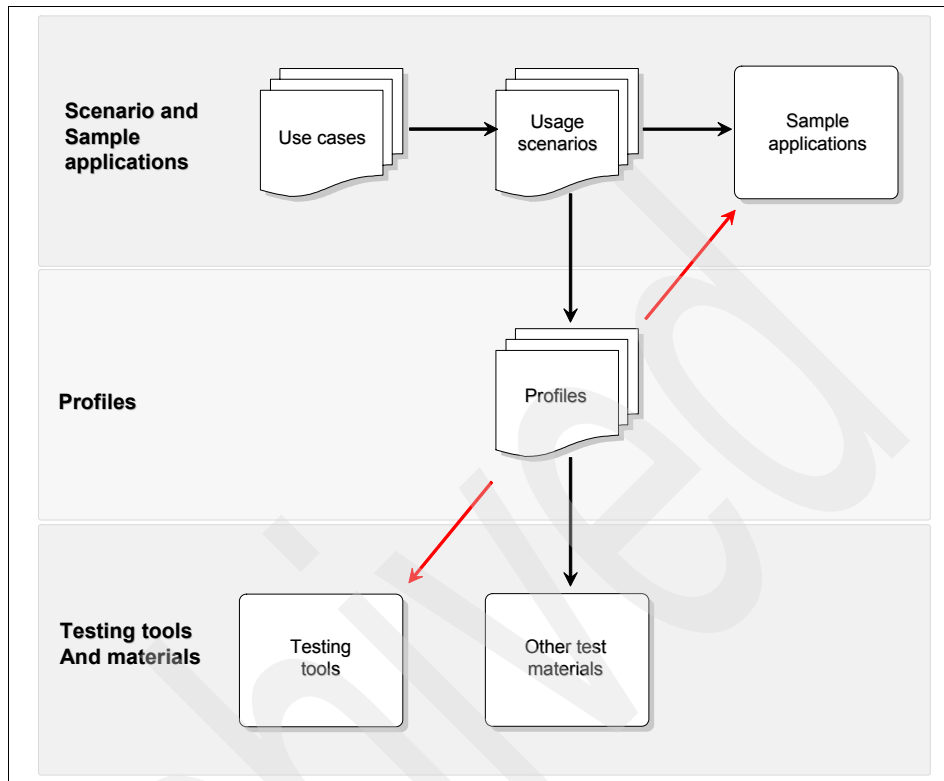


Figure 7-4 WS-I deliverables

7.2.1 Profiles

WS-I profiles are a set named Web Services specifications, that, together with interoperability implementation guidelines, recommend how the specifications can be used to attain interoperability across the independently developed Web Service standards.

Basic Profile

WS-I Basic Profile provides constraints and clarifications to a set of core Web Services specifications and implementation guidelines recommending how they should be used together to develop interoperable Web Services.

The set of core Web Service specifications include the following:

- ▶ SOAP 1.1
- ▶ WSDL 1.1
- ▶ UDDI 2.0
- ▶ XML 1.0 (Second Edition)
- ▶ XML Schema Part 1: Structures
- ▶ XML Schema Part 2: Data types
- ▶ RFC2246: The Transport Layer Security Protocol Version 1.0
- ▶ RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile
- ▶ RFC2616: HyperText Transfer Protocol 1.1
- ▶ RFC2818: HTTP over TLS Transport Layer Security
- ▶ RFC2965: HTTP State Management Mechanism
- ▶ The Secure Sockets Layer Protocol Version 3.0

The following are used in WS-I Basic Profile:

- ▶ **Messaging:** The exchange of protocol elements, usually over a network, to affect a Web service.
- ▶ **Description:** The enumeration of the messages associated with a Web service, along with implementation details.
- ▶ **Discovery:** Metadata that enables the advertisement of a Web service's capabilities.
- ▶ **Security:** Mechanisms that provide integrity and privacy.

Attachments Profile

The WS-I Attachments Profile complements the WS-I Basic Profile by adding support for interoperable SOAP messages with attachments. Attachments are typically used to send binary data. SOAP Messages with Attachments (SwA) define a MIME multipart/related structure for packaging attachments with SOAP messages.

The WS-I Attachments Profile incorporates the following specifications:

- ▶ SOAP Messages with Attachments
- ▶ Extensible Markup Language (XML) 1.0
- ▶ Namespaces in XML 1.0
- ▶ RFC2557: MIME Encapsulation of Aggregate Documents, such as HTML

- ▶ RFC2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies
- ▶ RFC2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types
- ▶ RFC2392: Content-ID and Message-ID Uniform Resource Locators

The following are used in WS-I Attachments Profile:

- ▶ Message: Protocol elements that transport the envelope.
- ▶ Envelope: The serialization of the soap:Envelope element and its content.
- ▶ Description: Descriptions of types, messages, interfaces, and their concrete protocol and data format bindings, and the network access points associated with Web Services.
- ▶ Instance: Software that implements a wsdl:port or a uddi:bindingTemplate.
- ▶ Consumer: Software that invokes an instance.
- ▶ Sender: Software that generates a message according to the protocol(s) associated with it.
- ▶ Receiver: Software that consumes a message according to the protocol(s) associated with it.

Simple SOAP Binding Profile

The WS-I Simple SOAP Binding Profile provides clarifications to and amplifications of Web Services specifications that promote interoperability.

This Profile incorporates the following specifications:

- ▶ Simple Object Access Protocol (SOAP) 1.1
- ▶ Extensible Markup Language (XML) 1.0 (Second Edition)
- ▶ Namespaces in XML 1.0
- ▶ RFC2616: Hypertext Transfer Protocol -- HTTP/1.1

The following are used in a WS-I Simple SOAP Binding Profile:

- ▶ Envelope: The serialization of the soap:Envelope element and its content.
- ▶ Message: Protocol elements that transport the envelope.
- ▶ Description: Descriptions of types, messages, interfaces, and their concrete protocol and data format bindings, and the network access points associated with Web Services.
- ▶ Instance: Software that implements a wsdl:port or a uddi:bindingTemplate.
- ▶ Receiver: Software that consumes a message according to the protocol(s) associated with it.

Note: A number of the WS-I profiles have yet to attain a finalized version. The following are currently released as draft versions:

- ▶ Basic Security Profile
- ▶ Kerberos Token Profile
- ▶ REL Token Profile
- ▶ SAML Token Profile

7.2.2 Sample applications

WS-I delivers Web Services sample applications that implement WS-I profiles compliant applications. The sample applications that are developed using multiple platforms, languages, and programming tools demonstrate interoperability and are usable resources for developers looking for reusable working examples that follow the WS-I guidelines.

A high level definition of the applications are modeled using WS-I Use Cases. The use cases in turn employ WS-I Scenarios to define Web Services structured interactions. In designing the sample applications, WS-I defined scenarios with three basic Web Services interaction patterns that conform to the Basic Profiles:

- ▶ One-way

The Web Service Consumer sends a request message to the Provider. The Provider processes the message, but does not return a response message and the Consumer does not expect a response.

- ▶ Synchronous request/response

The Consumer sends a request message to the Provider. The Provider receives the message, processes it, and sends back a response message back to the Consumer.

- ▶ Basic Callback

The Basic Callback scenario is a form of asynchronous message exchange for Web Services. It consists of a pair of synchronous request/response interactions. The description of the callback service is defined and published by the Provider. The Consumer implements the callback service and provides the endpoint information to the Provider.

The callback scenario is initiated by the Consumer at runtime through a request in the first of two request/response sequences. The Provider receives the request and sends back an acknowledgement immediately. Subsequently, the Provider initiates the callback request/response to the Consumer with the response data for the initial request sent by the Consumer.

7.2.3 Testing tools

The testing tools are resources made available to developers to ensure that the messages exchanged by their Web Services implementations conform to WS-I guidelines. Messages flowing between Web Services are monitored and analyzed with the goal of uncovering unconventional usage or errors in specification implementations. The output is a log identifying known interoperability issues.

Note: WS-I has delivered tests to verify conformance with the Basic Profile 1.0; tests for the other WS-I profiles are expected to follow.

Designing Web Services interoperability

In this chapter, we introduce the key elements for realizing Web Services interoperability. We identify the activities for defining and invoking Web Services, and the constraints that are designed to make Web Services more interoperable.

This chapter contains the following:

- ▶ Elements of Web Services interoperability
- ▶ Web Services description
- ▶ Web Services invocation
- ▶ Web Services constraints
- ▶ WS-Security support

8.1 Elements of Web Services interoperability

We introduced the key elements for realizing application interoperability in 1.4, “Elements of interoperability” on page 27. Designing Web Services interoperability has associated activities and constraints that correspond to the key elements. In this section, we identify the Web Services activities and introduce WS-I Profiles as constraints that ensure interoperability.

8.1.1 Web Services activities

There are two basic types of Web Services activities: description activities, which are activities you undertake to define Web Services, and invocation activities, which are activities associated with the flow in the invocation of a Web Service from the calling endpoint to the called endpoint.

Description activities

Web Services definitions make use of Web Services Definition Language (WSDL) to define Web Service endpoints, which includes abstract definitions (operations and messages) and concrete definition (network protocol and message formats). The description stack consist of the following:

- ▶ Types
XML Schema language definition of the base data types.
- ▶ Message
Definition of the data being transmitted or received. Each message is a named collection of message parts, where each part consists of a name and a type.
- ▶ Port type
Port types are a named collection of related operations, where operations are defined as the signature, inputs, outputs, and fault messages.
- ▶ Binding
This is the association of the protocol for message communication with the port type, its operations, and its messages.
- ▶ Port
Port definitions associate network address to port types.
- ▶ Service
Service is the process by which related ports are given names that relate to the functions that they provide.

Figure 8-1 on page 271 gives an overview of the Web Services description stack.

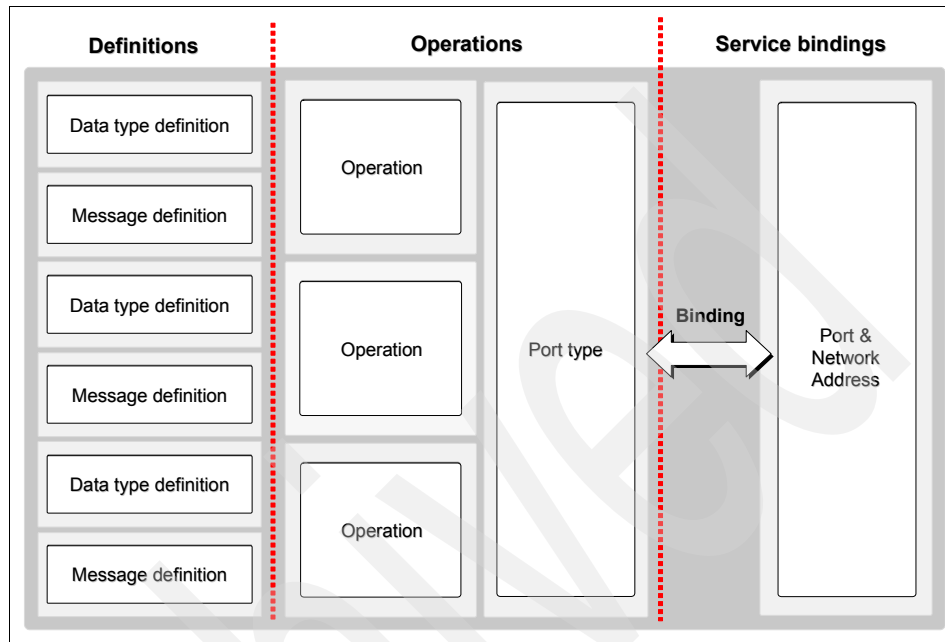


Figure 8-1 Web Services description stack

You do not have to worry about creating these descriptions by hand, both WebSphere and .NET IDEs eliminate a large amount of coding and make use of wizards to automate these tasks, including the generation of Java and C# stubs.

Invocation activities

Associated with the four layers of the application interoperability stack are Web Services invocation activities. The activities identify specific Web Services interoperability functions in the flow from the calling endpoint to the called endpoint:

- ▶ **Application layer**

The application layer is where your application interfaces with the Web Service. Your Java or C# code invokes the Web Service passing data, as defined in operations signature.

- ▶ **Data layer**

The writing and processing of XML data messages that are exchanged are defined in the data layer.

- ▶ Control layer

The binding to SOAP protocol and the definition of the syntactic and semantic processing of outgoing and incoming SOAP messages.
- ▶ Transport layer

Definition of the underlying protocol for sending and receiving messages.

Figure 8-2 gives an overview of Web Services invocation activities.

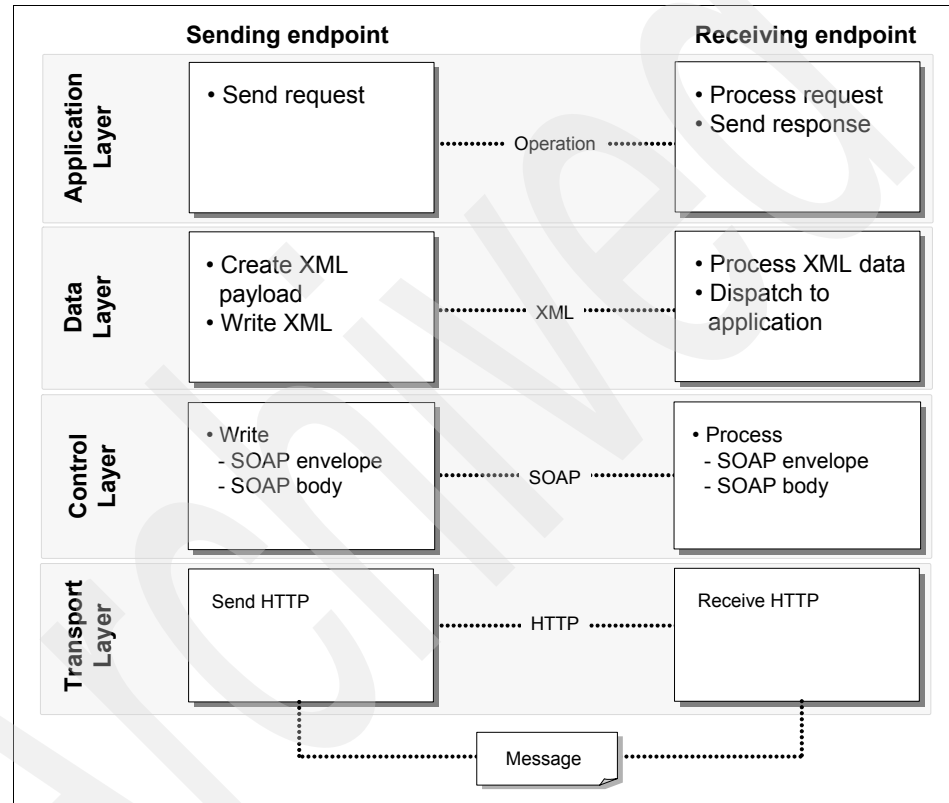


Figure 8-2 Web Services invocation activities

8.1.2 Web Services constraints

The WS-I Basic Profile provides constraints for each of the description and invocation activities. Compliance to these constraints goes a long way in ensuring that your Web Services solutions attains the best interoperability.

Note: The WS-I Security Profiles provides the security constraints; however, it is still in the draft stage.

8.2 Web Services description

A Web Services description is basically how you define the interoperable interface of any new application you are developing (or have developed, for that matter). You have two approaches for creating the description. One, you start with an existing application, an EJB or a COM+ object, for example, and using Rational Application Developer for WebSphere Software or Visual Studio .NET, you generate the WSDL file that matches the interfaces in your Java or C# application. The second approach is where you start with the WSDL definition and have the development tool (Rational Application Developer for WebSphere Software or Visual Studio .NET) generate the Java or C# application that corresponds to the definitions in the WSDL file.

From a programming perspective, the application to WSDL generation approach is obviously easier, since you do not have to deal with WSDL details. However, the generated WSDL file may include platform specific constructs and behaviors that are counter to interoperability. With the WSDL to application approach, by defining the data and behavior interface first, you stand to achieve better interoperability, since the generated applications conform to a common interface.

Tip: Design the XSD and WSDL for your Web Service first and then create the interoperable applications against the schema and interface.

As the starting point for all interactions with the Web Services, the WSDL interface has a number of responsibilities that call for careful considerations. They include the following:

- ▶ **State management**
With the expectation of many clients concurrently accessing the Web Services, you have to consider how to manage state information and support multithreaded access and transaction processing.
- ▶ **Granularity of Service**
You have to consider the functionality of the methods and strive to achieve a balance between flexibility from fine-grained method interfaces and the efficiency of coarse-grained interfaces. Fine-grained interfaces are quite flexible, but can be chatty, resulting in network overhead and reduced performance. Coarse-grained interfaces, while more efficient, require careful consideration to ensure that logical operations are grouped together.

► Parameter types

Data types are a major source of interoperability issues. Careful consideration should therefore be given to data types for the parameters of the methods exposed in the Web Services.

► Interactions

You should consider the type of processing associated with each exposed method. If a request can be processed in a short period of time, you should consider making the interface synchronous where the requester blocks until the request is serviced. If, on the other hand, the processing will take a considerable amount of time, it may be best not to block the requester, but rather to process the request asynchronously.

Type definition

Types are one the key elements of Web Services. They contain data type definitions that describe the messages exchanged in Web Services. Types are specified using the XML Schema definition (XSD), which is the WSDL preferred canonical type system for maximum interoperability and platform neutrality.

A lot of Web Services interoperability issues are centered around what data types are exposed and passed as parameters. The sample type definition in Example 8-1 shows definitions for two types: “registerClaimResponse”, which defined a data type “registerClaimReturn”, and “findCustomer”, with two data types, “customerID” and “policyID”. Types like the ones in Example 8-1 do not pose problems when it comes to interoperability. They map easily to XSD types and to the native data types in WebSphere and .NET.

Example 8-1 Sample type definition

```
...
<element name="registerClaimResponse">
  <complexType>
    <sequence>
      <element name="registerClaimReturn" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>

<element name="findCustomer">
  <complexType>
    <sequence>
      <element name="customerID" nillable="true" type="xsd:string"/>
      <element name="policyID" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
```

...

Other type related issues, such as the native interpretation of certain data types, do impact interoperability. One of the key problems with data types is a result of underlying implementation differences in WebSphere and .NET. These differences manifest in mappings between XSD and native types, hence data sent from one platform is not accurately interpreted on the receiving side. For example, in Java, all arrays, classes, and interfaces are reference type, hence, `java.util.Date` and `java.util.Calendar` classes are reference types. However, in .NET, `System.DateTime` is considered a value type.

There are a number of known aspects of type definition that do impact interoperability. In this section, we identify them and present practical guidelines and best practices that improve interoperability.

Type mismatch

There are some type mismatches in Java and .NET. These mismatches are often sources of interoperability issues when exposed across Web Services. Here are a few that you should pay close attention to:

► Collections

Both Java and .NET support a rich library of collection types. In Java, for example, there are:

- `java.util.Hashtable`
- Vectors
- Hashmap
- Set
- ArrayList

And in .NET there are:

- `System.Collections.Hashtable`
- SortedLists
- Queue
- Stack
- ArrayList

Although these collections appear to be similar between Java and .NET, when exposed across Web Services, however, they are sources of interoperability issues. The problem lies in the interpretation of the XML schema at the receiving end. Weak typing of collections objects leads to inaccuracies in the interpretation and mapping of the types to native data types. You can make

collections more interoperable by wrapping any weakly types collection objects with simple arrays of concrete types.

Note: For more information on designing for type mismatch interoperability, see Web Services programming tips and tricks: Improve the interoperability between J2EE and .NET, Part 2 at:

<http://www-128.ibm.com/developerworks/webservices/library/ws-tip-j2eenet2.html>

► Unavailable types

XSD offers a wide range of types for mapping to and from native types as you construct WSDL definitions. One-to-one mapping is not always possible between native types and XSD types, and between native types. .NET, for example, has unsigned integral (byte, ushort, uint, and ulong) and decimal types that do not exist in Java.

Although XSD provides support for unsigned types, such as `xsd:unsignedInt`, `xsd:unsignedLong`, `xsd:unsignedShort`, and `xsd:unsignedByte`, for the state of interoperability, it is advisable not to expose these numerical data types in the Web Service methods.

You should consider wrapping any methods that expose these numerical types and use `xsd:string` (using `System.Convert.ToString` in C#) to transmit the values.

► Precision

Higher precision beyond the result type is possible on each platform, so loss of precision might occur between platforms. You should test `xsd:decimal`, `xsd:double`, and `xsd:float` types in your interoperable solution for loss of precision.

► Value/reference type

The Java language has two kinds of types: primitive types and reference types. The .NET type system includes value, reference, and pointer types. The corresponding value types can be assigned to variables, passed as parameters, and returned by methods. Interoperability failure can occur if a value type for one language is mapped to a reference type in the other language. For example, the `xsd:dateTime` is mapped to `System.DateTime`, which is a value type in .NET, while `java.util.Date` and `java.util.Calendar` are reference types. You can assign a null value to reference types when it is not referencing any object. .NET Web Services will throw a `System.FormatException` if it receives a null value for a value type. The way to avoid this problem is to define a complex type that wraps the value type. You can then set the complex type to be null to indicate a null reference.

Note: WSDL files that are created or imported into Rational Application Developer for WebSphere Software, can be validated against the W3C Web Services Description Language (WSDL) V1.1 specifications and against the WS-I profiles. WSDLs generated by Rational Application Developer for WebSphere Software are valid but are not necessarily WS-I compliant. Compliance to WS-I profiles depend on your selections, for example, selecting RPC/encoded or SOAP over JMS bindings will generate a non-compliant WSDL.

Attention: When you have a WebSphere client calling Web Services in .NET, there are namespace related issues that you should be aware of.

► ASMX namespace

IBM Rational Application Developer for WebSphere Software maps the domain portion of the ASMX Web Services namespace into its package name. To avoid duplication, make sure that the domain portion of the namespace is unique. For example, with two Web Services with the same name but created in the namespaces

`http://itso.ral.ibm.com/services/claims` and

`http://itso.ral.ibm.com/services/registration`, proxy files in the package `com.ibm.ral.itso` will be generated. The duplication will result in the loss of the second file. Using `http://claims.itso.ral.ibm.com/services` and `http://registration.itso.ral.ibm.com/services` instead will create packages named `com.ibm.ral.itso.claims` and `com.ibm.ral.itso.registration`.

Alternatively, giving unique names to all ASMX files will create uniquely named service proxy files, but a single shared package named `com.ibm.ral.itso`.

► Array types

If two .NET Web Services, such as Registration and Claims above, share Customer schema, and if an array of Customers is passed as a parameter, IBM Rational Application Developer for WebSphere Software puts a generated client proxy into the shared package, but binds the class to the first namespace of the Web Service as a result, so the proxy class generated for the second Web Service is incorrect.

For more information, refer to:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/WSInteropRecsIBM-final.asp>

8.3 Web Services invocation

WSDL supports SOAP, HTTP, and MIME transport bindings. Other bindings, such as EJB, JMS, and plain Java, are available as well. However, the use of SOAP/HTTP binding is a WS-I Basic Profile key constraint.

WSDL binding for SOAP 1.1 endpoints supports the specification of the following protocol-specific information:

- ▶ An indication that a binding is bound to the SOAP 1.1 protocol
- ▶ A way of specifying an address for a SOAP endpoint
- ▶ The URI for the SOAPAction HTTP header for the HTTP binding of SOAP
- ▶ A list of definitions for headers that are transmitted as part of the SOAP envelope

Table 8-1 lists the corresponding elements.

Table 8-1 SOAP elements in WSDL

Extension and attributes		Explanation
<soap:binding ...>		Binding level; specifies defaults for all operations.
	transport="uri" (0 or 1)	Binding level; transport is the runtime transport protocol used by SOAP (HTTP, SMTP, and so on).
	style="rpc document" (0 or 1)	The style is one of the two SOAP communication styles, rpc or document.
<soap:operation ... >		Extends operation definition.
	soapAction="uri" (0 or 1)	URN.
	style="rpc document" (0 or 1)	See binding level.
<soap:body ... >		Extends operation definition; specifies how message parts appear inside the SOAP body.

Extension and attributes		Explanation
	parts="nmtokens"	Optional; allows externalizing message parts.
	use="encoded literal"	encoded: Messages reference abstract WSDL type elements; encodingStyle extension used. literal: Messages reference concrete XSD (no WSDL type); usage of encodingStyle is optional.
	encodingStyle="uri-list" (0 or 1)	List of supported message encoding styles.
	namespace="uri" (0 or 1)	URN of the service.
<soap:fault ... >		Extends operation definition; contents of fault details element.
	name="nmtoken"	Relates soap:fault to wsdl:fault for operation.
	use, encodingStyle, namespace	See soap:body.
<soap:address ... >		Extends port definition.
	location="uri"	Network address of RPC router.
<soap:header ... >		Operation level; shaped after <soap:body ...>.
<soap:headerfault ... >		Operation level; shaped after <soap:body ...>.

SOAP encoding

Encodings define how data values can be translated to and from a protocol format. We refer to these translation steps as *serialization* and *deserialization*, or, synonymously, *marshalling* and *unmarshalling*.

SOAP encodings tell you how to translate from data structures constructed in a specific programming language into SOAP XML and vice versa.

The following encodings are defined:

► SOAP encoding

SOAP encoding enables marshalling/unmarshalling of values of data types from the SOAP data model. This encoding is defined in the SOAP 1.1 standard.

► Literal

The *literal* encoding is a simple XML message that does not carry encoding information. Usually, an XML Schema describes the format and data types of the XML message.

- ▶ Literal XML

The *literal XML* encoding enables direct conversion of existing XML DOM tree elements into SOAP message content and vice versa. This encoding style is not defined by the SOAP standard, but is in the Apache SOAP 2.3 implementation. This encoding is not used in newer SOAP engines.

- ▶ XMI

XML metadata interchange (XMI) is defined by the Apache SOAP implementation. We do not use this encoding in this redbook.

Messaging modes

The two styles (RPC, document) and two most common encodings (encoded, literal) can be freely intermixed for the so called SOAP messaging mode. Although SOAP supports four modes, only three of the four modes are generally used, and further, only two are recommended by the WS-I Basic Profile.

- ▶ Document/literal

Provides the best interoperability between Java and non-Java implementations, and is also recommended for Java-to-Java applications.

- ▶ RPC/literal

Possible choice between Java implementations. Although RPC/literal is WS-I compliant, it is not frequently used in practice. There are a number of usability issues associated with RPC/literal, including, but not limited to, the mapping of Java arrays to RPC/literal WSDL.

- ▶ RPC/encoded

Early Java implementations (WebSphere Application Server Versions 4 and 5.0) supported this combination, but it does not provide interoperability with non-Java implementations.

- ▶ Document/encoded

Not used in practice.

Note: Because the WS-I Basic Profile recommends use of literal binding, only document/literal or RPC/literal should be used for WS-I conformance. The SOAP encoding is not recommended by the WS-I Basic Profile, mainly because of complexity and interoperability problems.

Although the WS-I Basic Profile V1.0 requirement allows for both document/literal and RPC/literal bindings, .NET does not provide explicit support for consuming a Web service that uses an RPC/literal binding.

You can convert WSDLs from RPC/literal to document/literal. For more information, see the Microsoft article "RPC/Literal and Freedom of Choice" at the following URL:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/rpc_literal.asp

Tools are also available for converting RPC/literal WSDL files to wrapped document/literal at:

http://www.gotdotnet.com/team/tools/web_svc/default.aspx

Important: There are restrictions when converting RPC/literal WSDL to wrapped document/literal—it is not possible to support *operation overloading*. Wrapped document/literal requires the element name to be the same as the operation name. Because the type information of the arguments is not available in a SOAP document/literal message, the Web service provider must use a unique element name to map the message to an implementation.

8.4 Web Services constraints

WS-I created Profiles to solve a number of interoperability problems starting with the numerous interrelated Web Services specifications. It identifies the list of applicable Web Services standards (including the version numbers) that are used to specify rules for interoperability implementation guidelines. The rules define both strong (such as MUST, and MUST NOT) and conditional (such as SHOULD, and SHOULD NOT) interoperability requirements.

8.4.1 WS-I Basic Profile V1.1

The WS-I Basic Profile has been split into two separate profiles as of V1.1: conformance to WS-I Basic Profile V1.1 plus conformance to the Simple SOAP Binding Profile V1.0. The two profiles are roughly equivalent to a combined conformance claim of WS-I Basic Profile V1.0 plus the published errata.

Some of the key constraints include:

- ▶ Precludes the use of SOAP encoding (document/literal or RPC/literal *must* be used)
- ▶ Requires the use of SOAP/HTTP binding
- ▶ Requires the use of HTTP 500 status response for SOAP fault messages
- ▶ Requires the use of HTTP POST method
- ▶ Requires the use of WSDL V1.1 to describe the interface
- ▶ Precludes the use of solicit-response and notification-style operations

- ▶ Requires the use of WSDL V1.1 descriptions for UDDI tModel elements representing a Web service

Note: The WS-I Basic Profile V1.1 incorporates the following specifications:

- ▶ Simple Object Access Protocol (SOAP) 1.1
- ▶ RFC2616: Hypertext Transfer Protocol -- HTTP/1.1
- ▶ RFC2965: HTTP State Management Mechanism
- ▶ Extensible Markup Language (XML) 1.0 (Second Edition)
- ▶ Namespaces in XML 1.0
- ▶ XML Schema Part 1: Structures
- ▶ XML Schema Part 2: Datatypes
- ▶ Web Services Description Language (WSDL) 1.1
- ▶ UDDI Version 2.04 API Specification, Dated 19 July 2002
- ▶ UDDI Version 2.03 Data Structure Reference, Dated 19 July 2002
- ▶ UDDI Version 2 XML Schema
- ▶ RFC2818: HTTP Over TLS
- ▶ RFC2246: The TLS Protocol Version 1.0
- ▶ The SSL Protocol Version 3.0
- ▶ RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile

8.4.2 WS-I Attachments Profile V1.0

The WS-I Attachments Profile V1.0 specifies guidelines for interoperability of Web service messages that contain attachments. Specifically, the Attachments Profile guides the use of SOAP messages that contain attachments using the SOAP Messages with Attachments (SwA) specification. The SOAP Messages with Attachments specification is a W3C Note.

In general, the Java industry has adopted the SOAP Messages with Attachments (SwA) specification. The SOAP with Attachments API for Java (SAAJ) API models the MIME message format for SOAP, as specified by SwA.

SwA is not without its problems, however. The SwA solution breaks the Web Services model to a certain extent. Among other issues, SwA does not work with WS-Security at this time. Because of this fundamental problem with SwA, W3C is moving in the direction of Message Transmission Optimization Mechanism (MTOM).

Another fundamental problem for interoperability with .NET is that Microsoft icurrently has not announced a plan to support SwA (and therefore the WS-I Attachments Profile V1.0). For a detailed discussion of possible implementations for passing attachments to or from Microsoft platforms, refer to the article *Web Services, Opaque Data, and the Attachments Problem*, available at:

<http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnwebsrv/html/opaquedata.asp>

8.4.3 WS-I Support

Table 8-2 provides a quick reference to the Web service runtimes and features relevant to Web Services for each of the major WebSphere Application Server releases.

Table 8-2 WebSphere Application Server Web service support

WebSphere Application Server	Web Services runtime	Web Services features
Version 4.0 Version 5.0	IBM SOAP (based on Apache SOAP)	Not WS-I compliant
Version 5.0.2 Version 5.1	IBM WebSphere Apache Axis V1.0 IBM SOAP	WS-I Basic Profile V1.0 JAX-RPC V1.0 JSR109 V1.0 SAAJ V1.1 UDDI V2.0 WS-Security (OASIS Draft 13) SOAP/JMS support Web Services caching Web Services performance monitoring
Version 6.0	IBM WebSphere Apache Axis V1.0 IBM SOAP (deprecated)	WS-I Basic Profile V1.1 JAX-RPC V1.1 JSR109 V1.1 SAAJ V1.2 UDDI V3.0 WS-Security V1.0 WS-AtomicTransactions WS-Coordination JAXR support Multiple protocol/encodings (SOAP/JMS, EJB) Web Services caching Web Services performance monitoring

In general, Web Services implemented in WebSphere Application Server are interoperable with each other, with the following exceptions: Web Services implemented using the IBM SOAP engine are not WS-I compliant and, therefore, are unlikely to interoperate.

Web Services that implement WS-Security will not interoperate between WebSphere Application Server Versions 5.0.2/5.1 and Version 6, because the former implements a draft version of the specification that is not message-level compatible with WS-Security V1.0. Refer to 8.5, “WS-Security support” on page 291 for more information.

IBM Rational Application Developer for WebSphere Software can be configured to require, suggest (or warn about non-compliance), or ignore compliance to the WS-I Basic Profile V1.1, Simple Soap Binding Profile (SSBP) V1.0, and WS-I Attachments Profile (AP) V1.0. Figure 8-3 shows the preference at the workspace level.

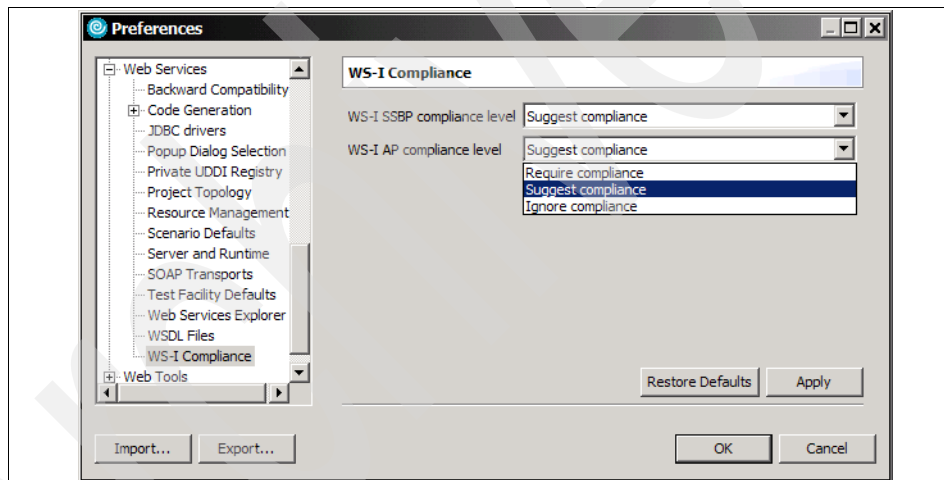


Figure 8-3 WS-I compliance levels preferences

These preferences determine the behavior of the WS-I validation tools built into Rational Application Developer for WebSphere Software:

- ▶ **Require:** The tool does not allow generating a non-compliant Web service.
- ▶ **Suggest:** The tool warns when generating a non-compliant Web service, but allows the user to progress.
- ▶ **Ignore:** The tool does not warn when generating a non-compliant Web service.

These settings can also be specified at the individual *project* level by right-clicking a project and selecting **Properties**. The project settings override the workspace settings. The default for project settings is Follow Preference (that is, follow the workspace settings).

When you use the Add New Project wizard of the Visual Studio.NET to generate a new ASP.NET Web service project, the ASMX page that is created is the starting point for creating a Web service that complies with the Basic Profile. Details of .NET compliance to WS-I Basic Profile is available from *Building Interoperable Web Services: WS-I Basic Profile 1.0*, found at:

<http://www.microsoft.com/products/info/product.aspx?view=22&pcid=e95ee5d8-fb4a-4cee-8d2e-c71746c939f1&crumb=catpage&catid=7dd90809-6d4d-47f4-9d13-e3df27ac09ba>

8.4.4 Web Services description constraints

In this section, we identify the Basic Profile rules that apply to Web Services description activities. The full text for each of the rules can be found at the following URL:

<http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html#references>

► General WSDL constraints (Table 8-3)

These constraints are general in nature and apply to all WSDLs.

Table 8-3 General WSDL constraints

General WSDL constraints	
Description of an Instance	R0001
Importing documents into WSDL	R2001, R2002, R2003, R2004, R2005, R2007, R2008, R2009, R2010, and R2011
Constraints on the overall structure of WSDL	R2020, R2021, R2022, R2023, R2024, R2025, R2026, R2027, R2028, R4002, R4003, and R4004
WSDL Extensions	R2747 and R2748

► Constraints on WSDL types (Table 8-4)

The WSDL type constraints apply to XML schema usage and type definitions.

Table 8-4 Constraints on WSDL types

Constraints on WSDL types	
Constraint on use of QNames	R2101 and R2102
Constraint on declaration of array types	R2110, R2111, R2112, and R2113
Usage of XML Schema	R2105, R2114, R2800, and R2801

► Constraints on WSDL messages (Table 8-5)

Provide the constraints on parts and messages.

Table 8-5 Constraints on WSDL messages

Constraints on WSDL messages	
Constraints relating to bindings and parts	R2201, R2202, R2203, R2204, R2206, R2207, R2208, and R2210
Constraints on portType	R2209

► Constraints on WSDL portTypes (Table 8-6)

These constraints apply to operations and on the wire messages.

Table 8-6 Constraints on WSDL portTypes

Constraints on WSDL portTypes	
Wire representation of the message	R2301, R2302, R2305, R2306, R2710, and R2712
Constraints on operations	R2303 and R2304

► Constraints on WSDL Bindings (Table 8-7)

The bindings constraints range from the structure and transport constraints to SOAP use, and namespace constraints.

Table 8-7 Constraints on WSDL Bindings

Constraints on WSDL Bindings	
Structure	R2029
Allowed bindings	R2401 and R2700
Transport constraints	R2701 and R2702
Constraints on soap:style	R2705
Constraints on soap:use	R2706 and R2707
Relationship to portTypes	R2709, and R2718
Using SOAPAction	R2713
Using soap:namespace attribute	R2716, R2717, and R2726
Faults and header constraints	R2719, R2720, R2721, R2722, R2723, R2740, R2741, and R2749

► Constraints on WSDL Port (Table 8-8)

Provide the constraints on allowed bindings.

Table 8-8 Constraints on WSDL Port

Constraints on WSDL Port	
Allowed bindings	R2711

Web Services invocation constraints

This section provides the mapping of the Basic Profile rules as constraints for the designated Web Services invocation activities.

► Write XML (Table 8-9)

These constraints apply to the XML representation of SOAP messages that are exchanged in Web Services interaction.

Table 8-9 Write XML

Write XML	
XML Representation of SOAP Messages	R4001, R1008, R1009, R1010, R1012, and R1013

► Process XML (Table 8-10)

These constraints apply to the processing of the types and formats in application level messages that are exchanged in Web Services interactions.

Table 8-10 Process XML

Process XML	
XML Representation of SOAP Messages	R4001, R1008, R1009, R1010, R1012, R1013, R1015, and R1017

► Write SOAP Envelope (Table 8-11)

The following constraints apply to the structure of the SOAP envelope.

Table 8-11 Write SOAP Envelope

Write SOAP Envelope	
Envelope structure	R1011 and R2714

► Process SOAP Envelope (Table 8-12)

The process SOAP envelope constraints cover requirements for action and the result of that action.

Table 8-12 Process SOAP Envelope

Process SOAP Envelope	
Envelope requirements	R1011, R1015, R1028, and R2714

► Write SOAP Body (Table 8-13)

The following constraints apply to the creation of SOAP messages.

Table 8-13 Write SOAP Body

Write SOAP Body	
XML Representation of SOAP Messages	R1005, R1006, R1007, R1011, R1014, R2735, and R2737
The SOAP Processing Model	R1025, R1029, and R1030
RPC messages	R2729

► Process SOAP Body (Table 8-14)

These constraints cover requirements for action and the result of that action.

Table 8-14 Process SOAP Body

Process SOAP Body	
XML Representation of SOAP Messages	R1005, R1006, R1007, R1014, R1017, R1028, R1029, and R1030

► Write SOAP Header (Table 8-15)

The constraints on writing SOAP header include the structure of the header blocks, the XML representation of SOAP messages, and the processing model and use of HTTP.

Table 8-15 Write SOAP Header

Write SOAP Header	
XML Representation of SOAP Messages	R4001, R1005, R1008, R1009, R1010, R1012, and R1013
The SOAP Processing Model	R1027
Using SOAP in HTTP	R1109
Header blocks	R2738, R2739, R2751, R2752, and R2753

► Process SOAP Header (Table 8-16)

Table 8-16 Process SOAP Header

Process SOAP Header	
XML Representation of SOAP Messages	R1012, R1005, R1008, R1009, R1010, R1012, R1013, R1015, and R1017
The SOAP Processing Model	R1025, R1026, R1027, R1028, R1029, and R1030

► Send HTTP (Table 8-17)

Table 8-17 Send HTTP

Send HTTP	
General	R1108, R1140, R1141, and R1132
Status code	R1106, R1107, R1111, R1112, R1113, R1114, R1115, R1116, R1124, R1125, R1126, and R1130
SOAPAction Header	R1109, R2713, R2744, and R2745
Cookies	R1120, R1121, R1122, and R1123

► Receive HTTP (Table 8-18)

Table 8-18 Receive HTTP

Receive HTTP	
General	R1110, R1140, R2746
Status code	R1107, R1111, R1112, R1113, R1114, R1115, R1116, R1124, R1125, R1126, R1130, R1131
SOAPAction Header	R1119
Cookies	R1120, R1121, R1122, R1123

8.5 WS-Security support

Both IBM WebSphere Application Server V6.0 and Microsoft Web Service Extensions (WSE) V2.0 implement the finalized OASIS WS-Security V1.0 standard and will interoperate.

The WS-Security V1.0 wire format has changed over time and is not compatible with previous WS-Security drafts. Also, interoperability between implementations based on previous drafts and V1.0 is not possible.

WebSphere Application Server V5.0.2 and V5.1 are based on the April 2002 draft specification of WS-Security. Microsoft WSE V1.0 is also based on a draft specification. Patches are available for WebSphere Application Server V5.x to interoperate between the two.

WebSphere Application Server V6.0 and Microsoft WSE V2.0 both support the approved WS-Security V1.0 standard and can interoperate. Due to the wire format difference, WS-Security V1.0 implementations will not interoperate with draft WS-Security implementations.

Information about the IBM WebSphere Application Server V6.0 support of WS-Security V1.0 is available in the redbook *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461, found at:

<http://www.redbooks.ibm.com/abstracts/sg246461.html?Open>

Web Services interoperability scenario

In this chapter, we introduce a Web Services interoperability scenario. It demonstrates a typical usage requirement for interoperability between IBM WebSphere and Microsoft .NET platforms.

The scenario implements Web Services interoperability between WebSphere and .NET using a common client application. It also demonstrates techniques and best practices for securing Web Services interoperability.

This chapter contains the following sections:

- ▶ Introduction to the claims processing scenario
- ▶ Building the WebSphere Claims Web Service
- ▶ Building the .NET Claims Web Service
- ▶ The client application

9.1 Introduction to the claims processing scenario

Consider the scenario where Lord General Insurance (LGI), a full service insurance company, acquired DirectCarInsure.com (DCI), a modern digital enterprise, which offers automobile insurance service exclusively through Web based e-commerce transactions. To provide a single system view to its customers, LGI is implementing a front-end application that makes use of existing business logic and processes from both companies to provide a seamless user experience.

LGI markets insurance products that it supports through its insurance agents and a call center. Its information system runs on the IBM WebSphere platform. DCI developed a Microsoft .NET solution to provide Web-based, direct automobile insurance services to its customers. The merger of the two companies will increase revenues and provide multiple channels and product offerings to the combined customer base.

This scenario is loosely based on materials originally published on developerWorks® as “Merging disparate IT systems. Build a single integrated view for users quickly and with minimal disruption”, which is available at:

<http://www-106.ibm.com/developerworks/ibm/library/i-merge.html>

Some of the materials used are derived from materials included with *WebSphere and .Net Interoperability Using Web Services*, SG24-6395. This redbook is available at:

<http://www.redbooks.ibm.com/abstracts/sg246395.html?open>

A major consideration in architecting the solution is the goal of showing rapid return on the investment by integrating the two systems as quickly as possible and keeping changes to existing systems and development environments to a minimum. The insurance applications developed and hosted in each company need to be integrated to support the Web based and the full service agent channels using a common service bus.

The integrated application relies on two Web Services: CustomerLookup, and ClaimRegistration. It makes use of a Servlet based user interface, a .NET implementation of ClaimRegistration, and a WebSphere implementation of CustomerLookup.

In addition to integrating the two systems, LGI's requirements include exposing a stable development and runtime service interface to business partners for integrating with their automated claims assessor process.

Use of Web Services

In this sample scenario, we show how to use Web Services technologies to meet the interoperability requirements by building Web Service interfaces using Rational Application Developer and Microsoft Studio .NET.

Note: Web Services is not the only method for achieving interoperability. As discussed in Part 2, “Component interoperability” on page 147, component level interoperability can also be used. Our objective with this scenario is to show in detail how you can use Web Services to implement an interoperability solution.

9.2 Building the WebSphere Claims Web Service

In order to create the CustomerLookup Web Service that wraps LGI’s existing business logic and make it available to the front-end application, we have to perform the following:

- ▶ Create the Web Services for LGI’s Register Claim application from existing Enterprise JavaBeans, which implement the business logic.
- ▶ Test the Web Services using the IBM Rational Application Developer Web Services Explorer.
- ▶ Deploy the Web Services in the WebSphere Application Server.

9.2.1 Configure the Development Environment

We make use of Rational Application Developer for the code development. In order to see the menus described in this section, you must enable the Web Services developer capability in the workbench:

1. Launch **Rational Application Developer**. If this is the first use of a new workspace, close the Welcome window to show the workbench.
2. Select **Window** → **Preferences**.
3. Expand **Workbench** and select **Capabilities**.
4. On the right, expand **Web Service Developer** and check the box by **Web Service Development**. Click **OK**.

Figure 9-1 gives an overview of the Enable Web Services development capabilities.

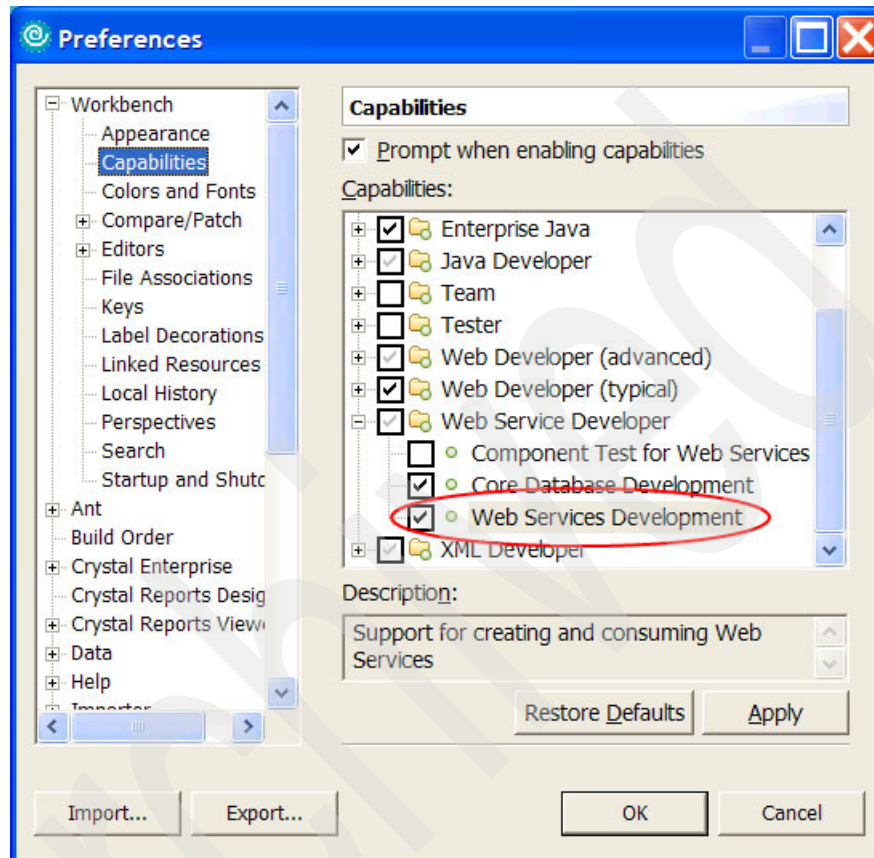


Figure 9-1 Enable Web Services development capabilities

The sample application is provided as a project interchange file. This must be imported into the workspace.

Note: For information about how to obtain the sample application, see Appendix A, “Additional material” on page 377

Perform the following steps:

1. Select **File** → **Import**.
2. Select **Project Interchange** and click **Next**.
3. For the From zip file, browse to the LGI server application ItsoClaim.zip.

- Click the **Select All** Button to select the three projects and click **Finish**.
Figure 9-2 gives an overview of the process.

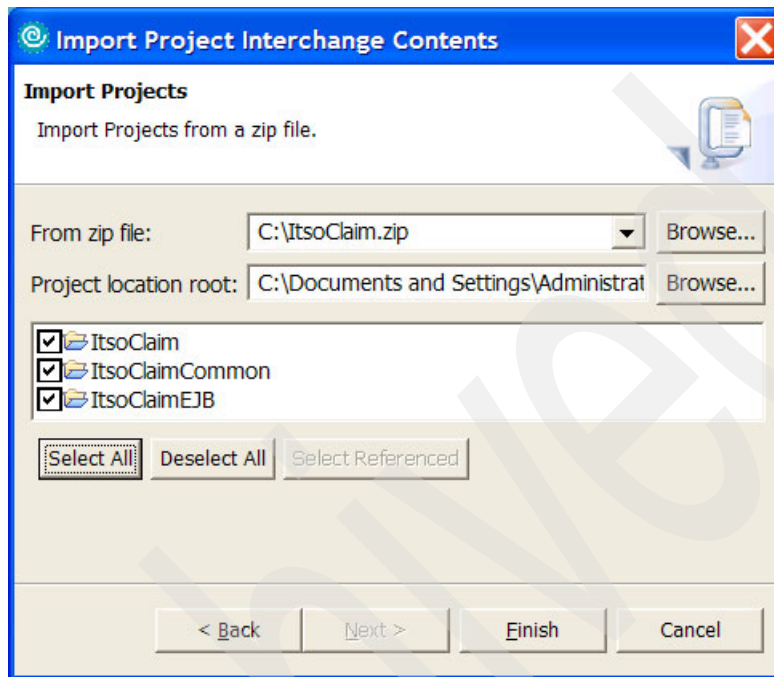


Figure 9-2 Importing the LGI service projects

- Watch the Building Workspace message in the lower right corner of the workbench. When the build completes, there should be no items in the Problems view.

9.2.2 Create Web Service from Session EJB

In this section, we are going to create a Web Service using the bottom-up development method starting with the ItsoClaimEJB. We will use SOAP over HTTP as the transport mechanism.

We will add a Web router project with a Servlet to route the client requests to the Session EJB. However, before we can commence development, we have to configure our environment, start the appropriate servers that we need, and then add the ItsoClaim application to the environment.

Here are the tasks we need to perform before we can create the Web Service:

- Configure a WebSphere Application Server in the Servers view.

Note: The steps to configure a server will vary depending on how your Rational Application Developer is installed. See the Rational Application Developer help for detailed instructions.

2. Start the WebSphere server .
3. Add ItsoClaim application to the server.
4. Generate a Web Service from the EJB:
 - a. In the J2EE Perspective, expand **EJB Projects** → **ItsoClaimEJB** → **Deployment Descriptor** → **Session Beans** in the Project Explorer view.
 - b. Right-click **LGIClaimRegistration** and select **Web Services** → **Create Web Service**.

Note: If you cannot find Web Services on the context menu, it may be because you have not enabled the Web Services capability. 9.2.1, “Configure the Development Environment” on page 295 shows how to configure the Web Services capability.

Figure 9-3 on page 299 gives an overview of the process.

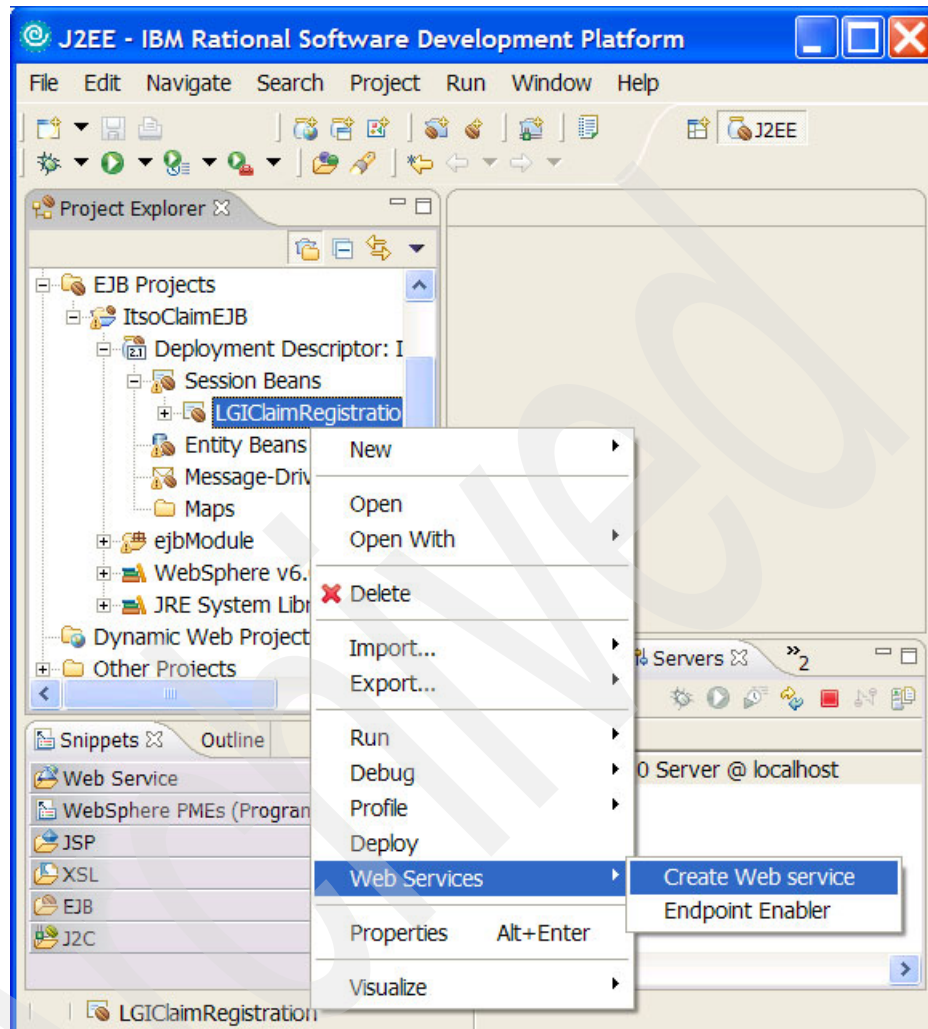


Figure 9-3 Web Service wizard: Launching the Wizard

- c. On the Web Services page, leave all default settings and click **Next** (see Figure 9-4).

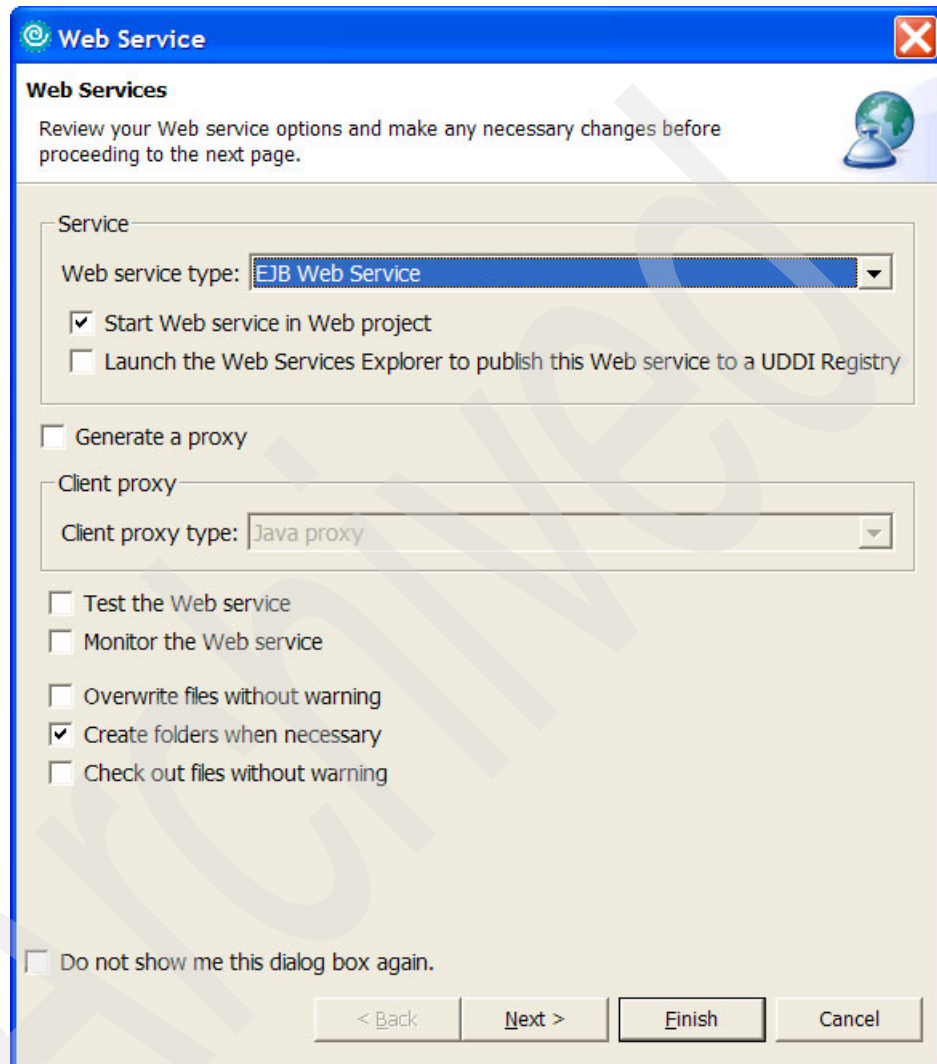


Figure 9-4 Web Service wizard: Web Services

- d. On the Object Selection Page, select the **LGIClaimRegistration EJB** and click **Next** (see Figure 9-5 on page 301).

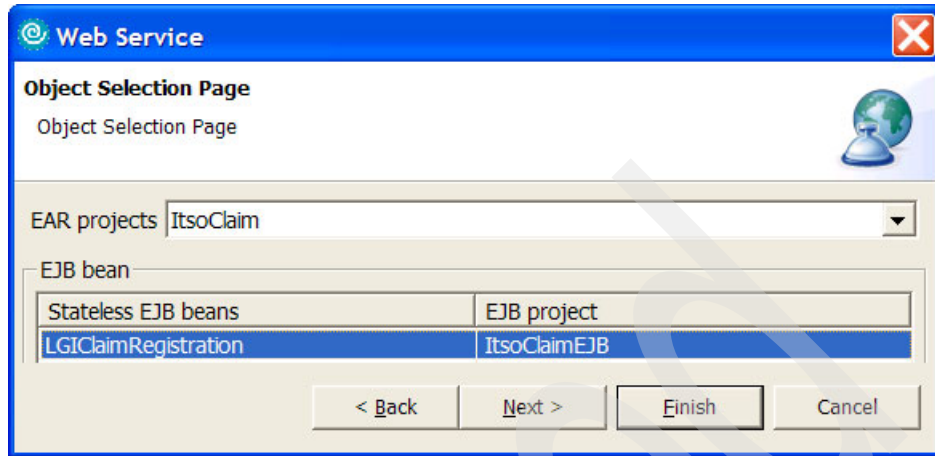


Figure 9-5 Web Service wizard: Object Selection Page

- e. On the Service Deployment Configuration page, leave all default settings and click **Next** (see Figure 9-6).

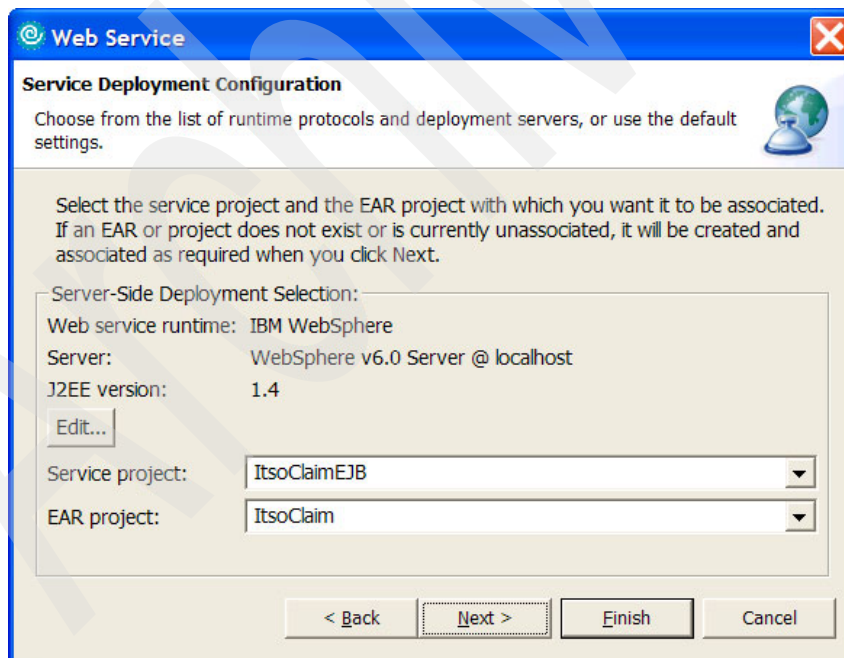
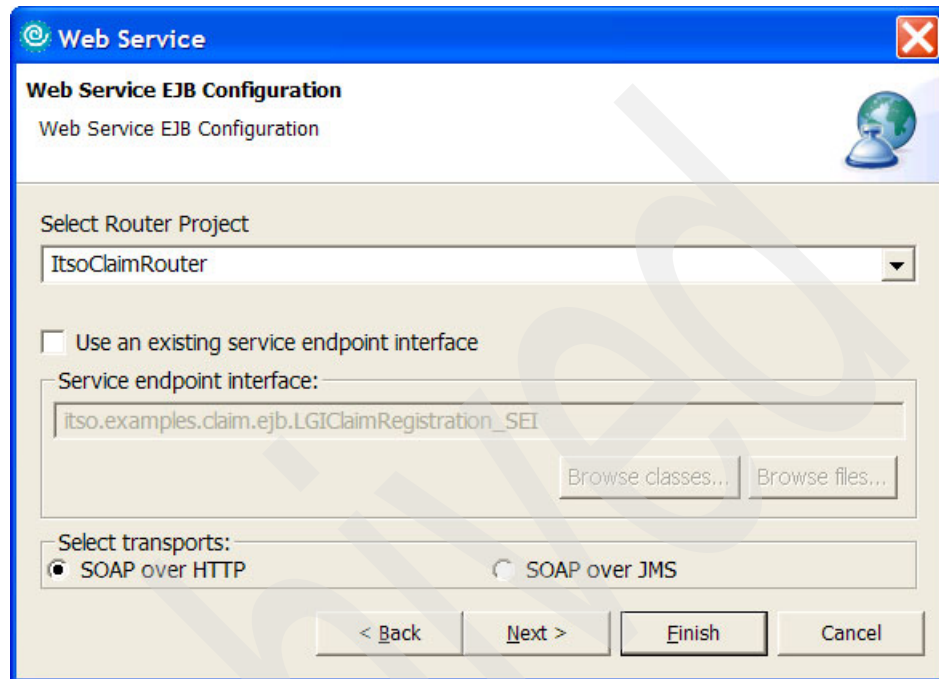


Figure 9-6 Web Service wizard: Service Deployment Configuration

- f. On the Web Service EJB Configuration page, type ItsoClaimRouter for the Router Project name and click **Next** (see Figure 9-7).



The image shows a 'Web Service EJB Configuration' dialog box. The title bar says 'Web Service'. The main title is 'Web Service EJB Configuration'. Below the title is a subtitle 'Web Service EJB Configuration'. There is a globe icon in the top right corner. The dialog has several sections: 'Select Router Project' with a text box containing 'ItsoClaimRouter' and a dropdown arrow; 'Use an existing service endpoint interface' with an unchecked checkbox; 'Service endpoint interface:' with a text box containing 'itso.examples.claim.ejb.LGIClaimRegistration_SEI' and two buttons 'Browse classes...' and 'Browse files...'; 'Select transports:' with two radio buttons, 'SOAP over HTTP' (selected) and 'SOAP over JMS'. At the bottom are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

Figure 9-7 Web Service EJB configuration

- g. On the Web Service Java Bean Identity page, make sure both methods are checked and click **Next** (see Figure 9-8 on page 303).

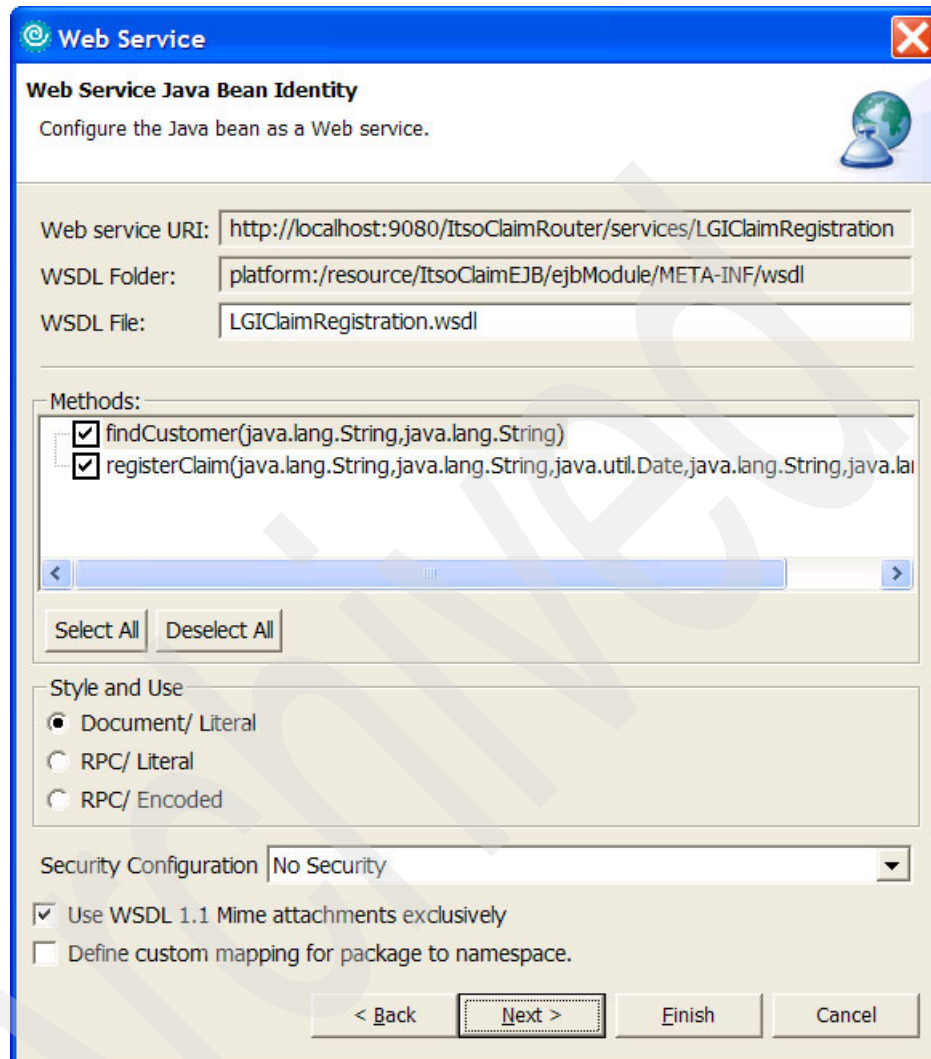


Figure 9-8 Web Service wizard: Web Service Java Bean Identity

- h. On the Web Service Publication page, leave all the default settings and click **Finish** (see Figure 9-9).

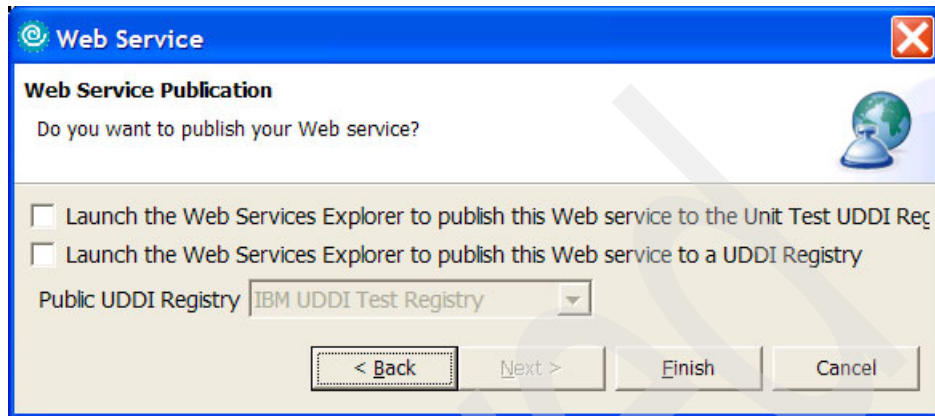


Figure 9-9 Web Service wizard: Web Service Publication

Generated files

The execution of the wizard results in the creation of the following files in the ItsoClaimEJB project.

Serialization helpers for ClaimException:

- ▶ itso.examples.claim.ejb.ClaimException_Deser.java
- ▶ itso.examples.claim.ejb.ClaimException_DeserProxy.java
- ▶ itso.examples.claim.ejb.ClaimException_Helper.java
- ▶ itso.examples.claim.ejb.ClaimException_Ser.java

Remote Interface (RI) and Service endpoint interface (SEI):

- ▶ itso.examples.claim.ejb.LGIClaimRegistration_RI.java
- ▶ itso.examples.claim.ejb.LGIClaimRegistration_SEI.java

A WSDL file that describes the Web service:

/META-INF/wsdl/LGIClaimRegistration.wsdl

There are deployment descriptor files that describe the Web service according to the Web Services for J2EE style (JSR 109):

- ▶ /META-INF/webservices.xml
- ▶ /META-INF/ibm-webservices-ext.xml
- ▶ /META-INF/ibm-Webservices-bnd.xml

The JAX-RPC mapping file is the LGIClaimRegistration_mapping.xml file.

Figure 9-10 shows the generated files.

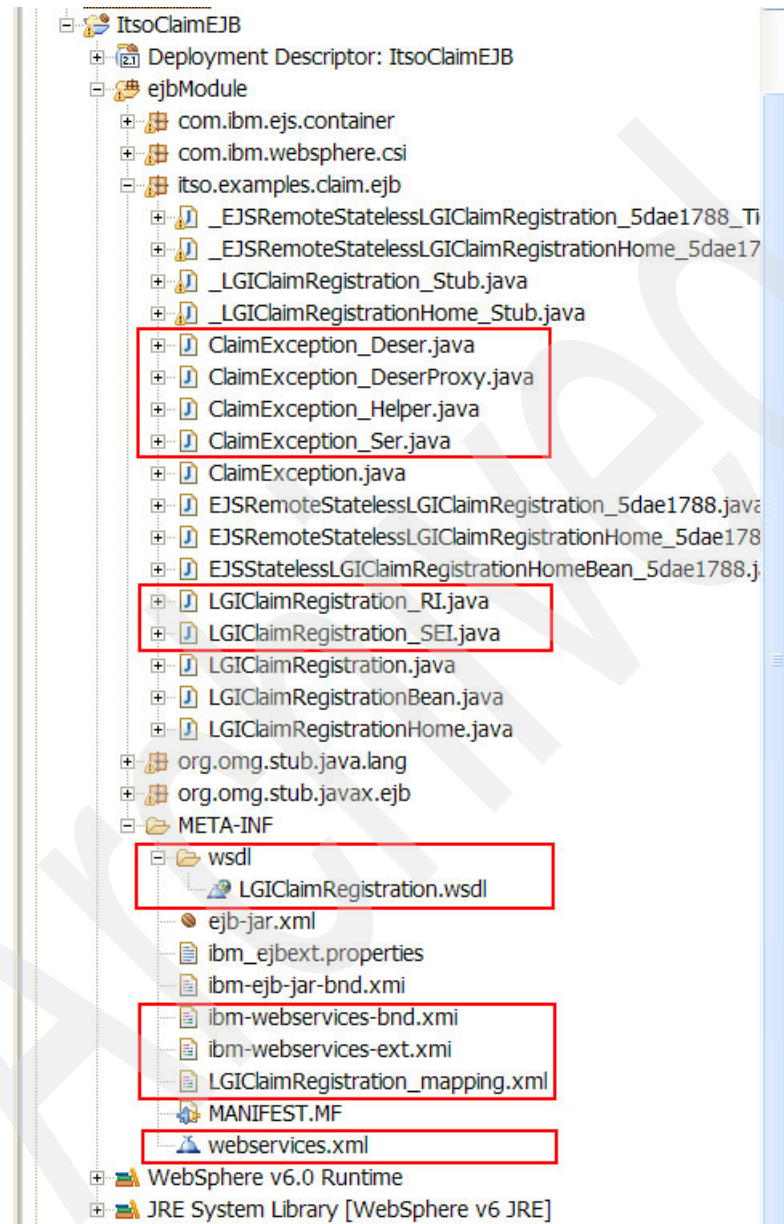


Figure 9-10 Files generated from Web Service Wizard

Note: In the Project Explorer, you also find a separate Web Services section with Services and Clients.

9.2.3 Testing with the Web Services Explorer

Once the Web service is installed and running in a server, it can be tested using the Web Services Explorer.

To start the Web Services Explorer:

1. Select the **LGIClaimRegistration.wsdl** file in ITSOClaimeJB/ejbModule/META-INF/wsdl.
2. Right click and select **Web Services** → **Test with Web Services Explorer**.

A Web Browser view opens with the WSDL file selected. It shows the operations (methods) that can be invoked and the endpoint (see Figure 9-11).

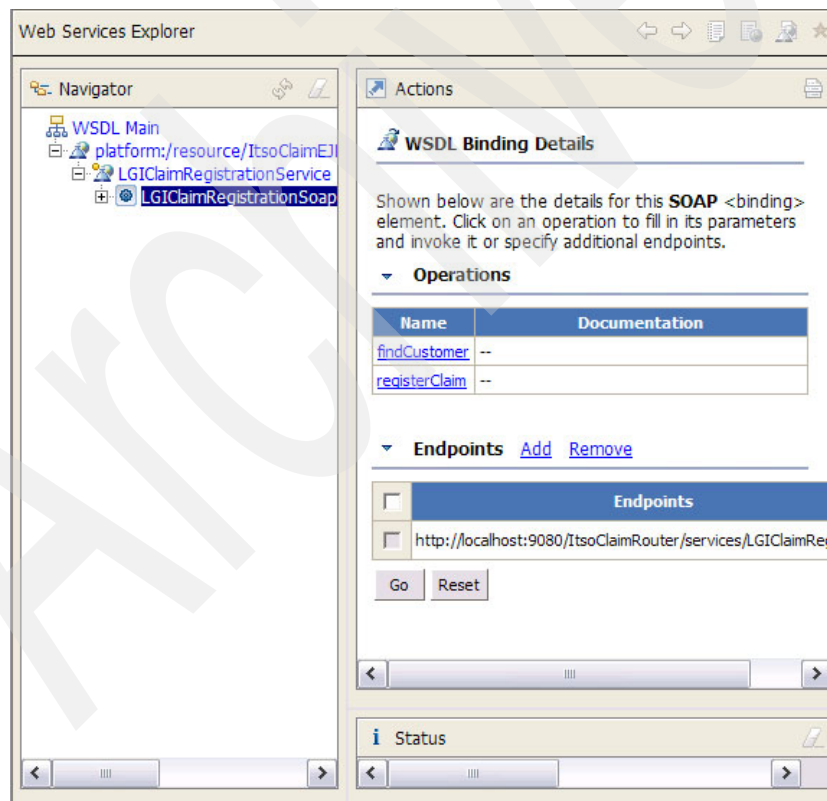


Figure 9-11 Web Services Explorer: Operations

3. Click **findCustomer** in the right panel.
4. Type the customerID ABC123 and policyID P00245, and click **Go**.
The Status pane shown in Figure 9-12 appears.

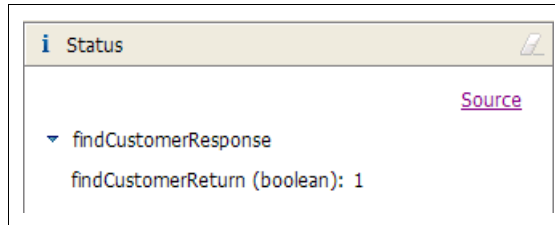


Figure 9-12 Web Services Explorer: Result status populated with Web Service Result

5. in the left panel, click **RegisterClaim** operation.
6. For customerID, type ABC123.
7. For policyID, type P00245.
8. For accidentDate, click **Browse** and select any date in the calendar. A textual representation of the date will be added to the form.
9. For accidentDescription, type in a short description.
10. For involvedCars, click **Add** twice. Type in short descriptions of the cars.
11. Click **Go**.

Figure 9-13 shows the testing of registerClaim.

The screenshot shows a web browser window titled 'Actions' with a sub-header 'Invoke a WSDL Operation'. Below this, there is a text instruction: 'Enter the parameters of this WSDL operation and click **Go** to invoke.' A section labeled 'Endpoints' contains a dropdown menu with the value 'http://localhost:9080/ItsClaimRouter/services/LGIClaimRegistration'. Below this, the 'registerClaim' operation is expanded, showing several input fields: 'customerID' (string) with value 'ABC123', 'policyID' (string) with value 'P00245', 'accidentDate' (dateTime) with value '1963-03-06T19:32:31.939Z' and a 'Browse...' button, and 'accidentDescription' (string) with value 'A Man A Miss A Car A Curve He kissed the Miss and missed the Curve'. An 'involvedCars' section is also expanded, showing a table with two rows: 'Red' and 'Green'. At the bottom of the form are 'Go' and 'Reset' buttons.

string	string	Add	Remove
<input type="checkbox"/>	Red		
<input type="checkbox"/>	Green		

Figure 9-13 Testing registerClaim on WebSphere

The status box should display a response with a claim number, as shown in Figure 9-14 on page 309.

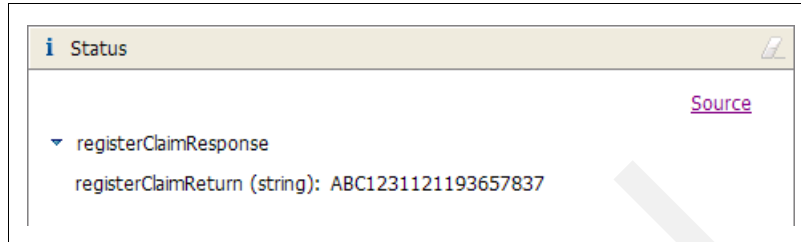


Figure 9-14 Response with claim number

9.2.4 Deploying the Web Service

In order to deploy the Web Service on the application server, we will first export the application to an EAR file and then install the EAR file on WebSphere Application Server.

Exporting the application to an EAR file

Follow these steps:

1. Select the **ITSOClaim Enterprise Application** project.
2. Select **File** → **Export**.
3. Select **EAR file** as the destination, and click **Next**.
4. Click **Browse** to locate the Destination. By default, the output file name that is populated is *ItsoClaim*. Change the file name to *ITSOClaimWS* and click **Save**.
5. Click **Finish** to perform the export. *ITSOClaimWS.ear* will be generated in the target directory.

Install the EAR file on WebSphere Application Server

This section assumes that WebSphere Application Server V6.0 is installed and a stand-alone profile is running on the default ports. Then:

1. Open the WebSphere administrative console using a Web browser with the URL, where the <server-hostname> is that of your WebSphere Application Server V6 machine:
<http://<washostname>:9060/ibm/console/>
2. Log in with a user ID.
3. Expand **Applications** and click **Install New Application**.
4. Specify the full path name of the enterprise application file (EAR file) on the local file system. Click **Next**.

Figure 9-15 gives an overview of these steps.

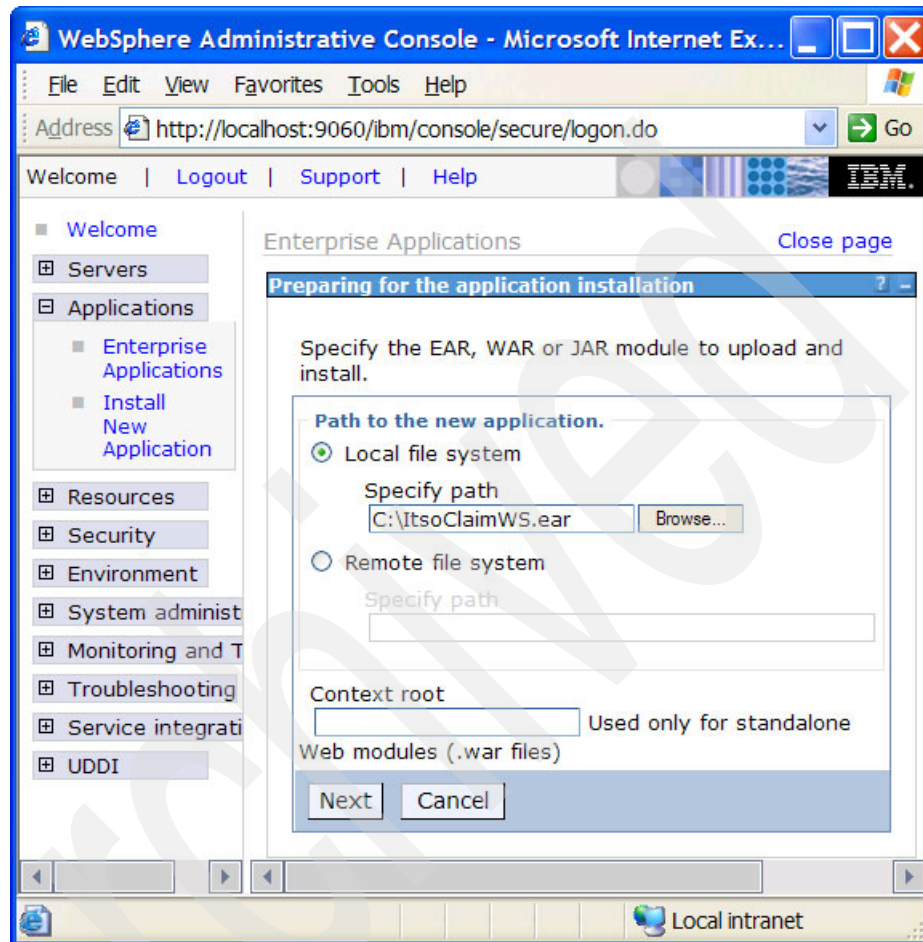
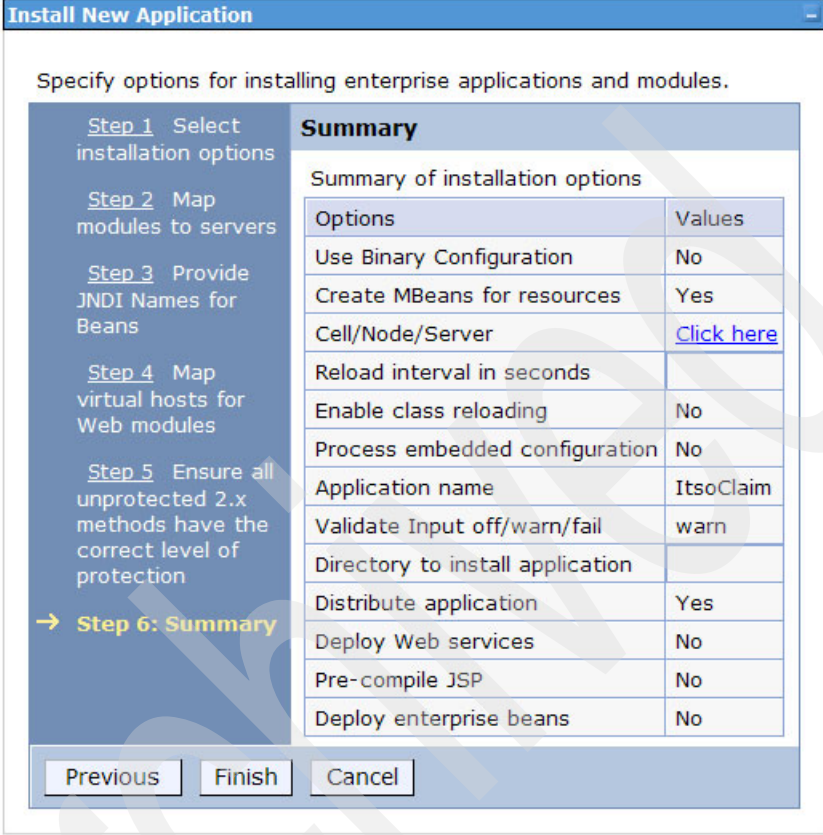


Figure 9-15 Preparing for the application installation

5. On the Preparing for the application installation page, leave all default values and click **Next**.
6. All remaining default values should be correct. Either step through the wizard to review all values, or click **Step 6** (Summary) to go directly to installation.

Figure 9-16 shows the installation summary.



Install New Application

Specify options for installing enterprise applications and modules.

[Step 1](#) Select installation options

[Step 2](#) Map modules to servers

[Step 3](#) Provide JNDI Names for Beans

[Step 4](#) Map virtual hosts for Web modules

[Step 5](#) Ensure all unprotected 2.x methods have the correct level of protection

→ **Step 6: Summary**

Summary

Summary of installation options

Options	Values
Use Binary Configuration	No
Create MBeans for resources	Yes
Cell/Node/Server	Click here
Reload interval in seconds	
Enable class reloading	No
Process embedded configuration	No
Application name	ItsoClaim
Validate Input off/warn/fail	warn
Directory to install application	
Distribute application	Yes
Deploy Web services	No
Pre-compile JSP	No
Deploy enterprise beans	No

[Previous](#) [Finish](#) [Cancel](#)

Figure 9-16 Installation summary

7. On the Summary page, click **Finish**.
8. When the installation completes, the message Application ItsoClaim installed successfully will be displayed. Click **Save to Master Configuration**.
9. Click the **Save** button to commit the changes.


Start the application

To start the application:

1. Expand **Applications** and click **Enterprise Applications**.
2. Check **ItsoClaim**.
3. Click the **Start** button.

Test the deployed Web Service using Web Service Explorer

The deployed Web Service can be tested using the Web Service Explorer in the development environment:

1. In Rational Application Developer, select **Run** → **Launch the Web Services Explorer**.
2. If not already on the WSDL page, click the  icon to switch to it. The window in Figure 9-17 should appear.
3. Click **WSDL Main** and enter the WSDL URL:
`http://<washostname>:9080/ItsoClaimRouter/services/LGIClaimRegistration?wsdl`
4. Click **Go**.

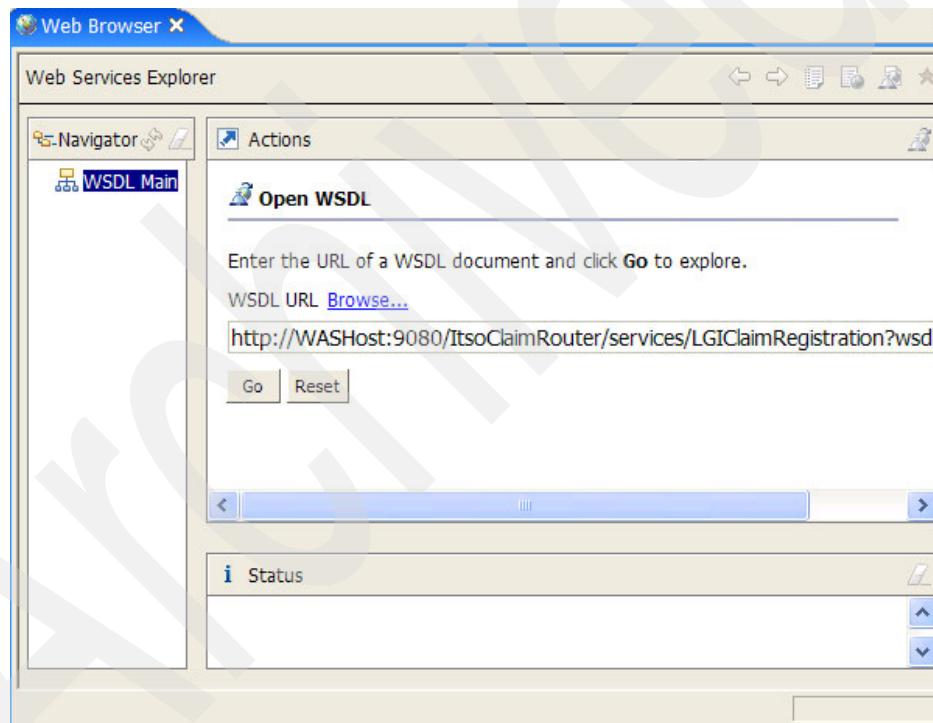


Figure 9-17 Using Web Services Explorer with WebSphere Application Server

5. Test, as in 9.2.3, “Testing with the Web Services Explorer” on page 306.

9.3 Building the .NET Claims Web Service

The next stage in the scenario is to create the Microsoft .NET Web Service for the DirectCarInsure (DCI) register claim application.

As with the LGI scenario, the register claim application is already implemented and in use with DCI's own Web site. The task is to wrap the existing application classes in a Web Service and deploy it onto the Windows 2003 server.

9.3.1 Create Web service project

Before we can create the Web service, we need to create an ASP.NET Web service project In Microsoft Visual Studio .NET 2003:

1. Start **Microsoft Visual Studio .NET 2003**.
2. Select **File** → **New** → **Project**.
3. Select **Visual C# Project** on the left-hand side. The window shown in Figure 9-18 on page 314 should appear.
4. Select **ASP.NET Web Service** on the right-hand side.
5. Rename the Web Service project name in Location from:
 http://localhost/WebService1
 to
 http://localhost/ItsoClaim
6. Click **OK**.

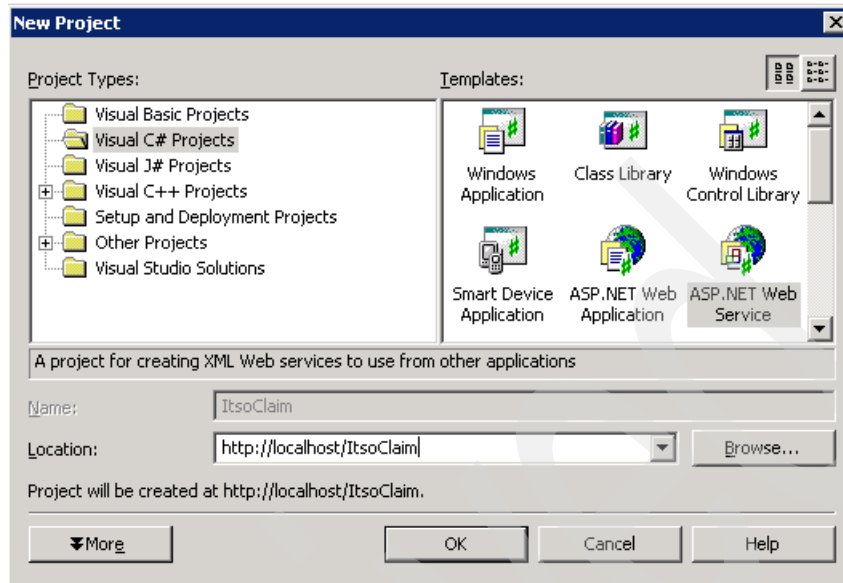


Figure 9-18 Create New Project for ItsoClaim ASP.NET Web service using C#

7. The Web service page opens up and has the extension .asmx.cs[Design]. The screen does not show any code, so we need to click the line **Click here to switch to code view**. See Figure 9-19.

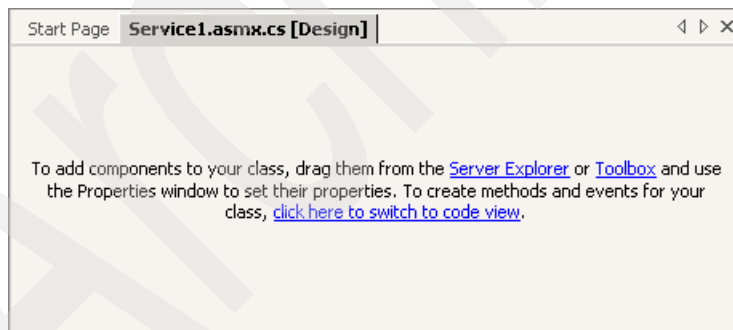


Figure 9-19 ItsoClaim.aspx.cs [Design] with default Service1 class name

8. Rename the file from Service1.aspx to ItsoClaim.aspx, as shown in Figure 9-20 on page 315.

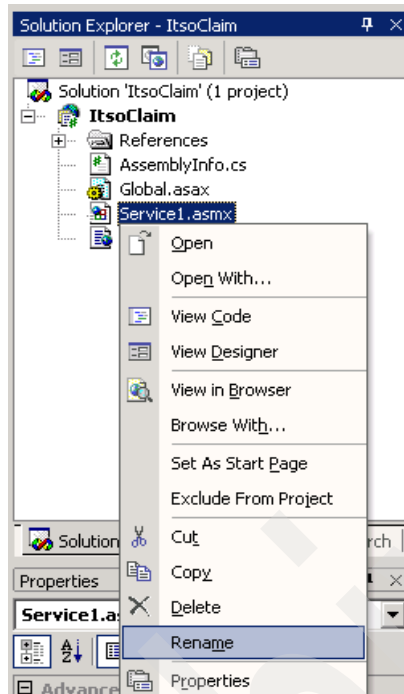


Figure 9-20 Rename Service1.asmx

9.3.2 Import existing classes

The existing DCI application consists of four classes:

- ▶ CustomerDataAccess: Encapsulates the access to customer data repository
- ▶ ClaimDataAccess: Encapsulates the access to claims data repository

Note: Customer and Claim data access classes simply return hardcoded values. These classes could be changed to use a database without the need to modify anything else in this example

- ▶ DataException: Simulates data access errors
- ▶ ClaimException: Simulates exception generated by the Web Service

Note: The Exception classes were extended and constructor calls to the parent classes were simplified.

To import existing classes into the project, do the following:

1. Select **File** → **Add Existing Item**.
2. Navigate to the directory containing the DCI application.
3. Hold the Ctrl key and select **ClaimDataAccess.cs**, **ClaimException.cs**, **CustomerDataAccess.cs**, and **DataException.cs**, as shown in Figure 9-21.
4. Click **Open**.

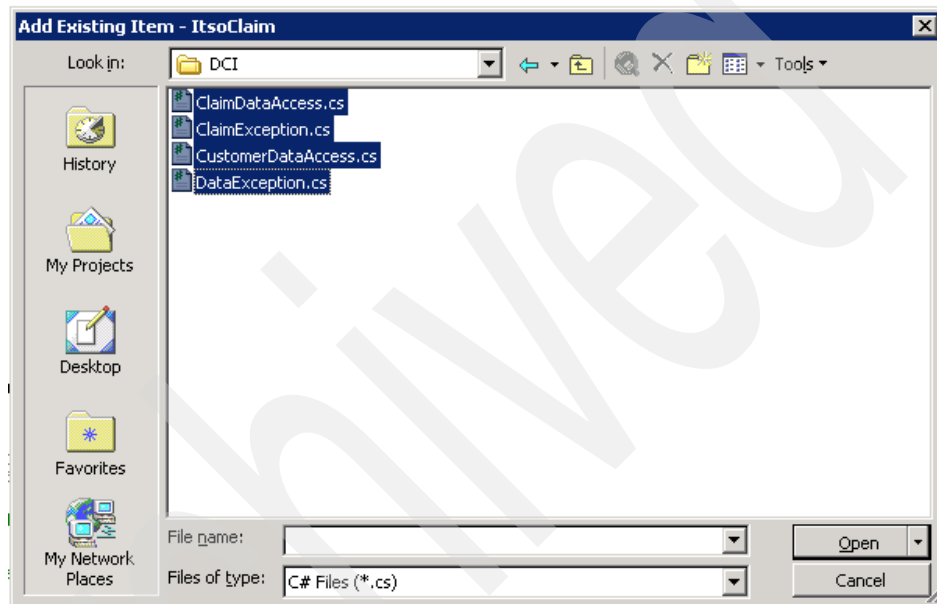


Figure 9-21 Import existing classes to the project

The application implementation files will now appear in the Solution Explorer as a part of the ItsoClaim project, as shown in Figure 9-22 on page 317.

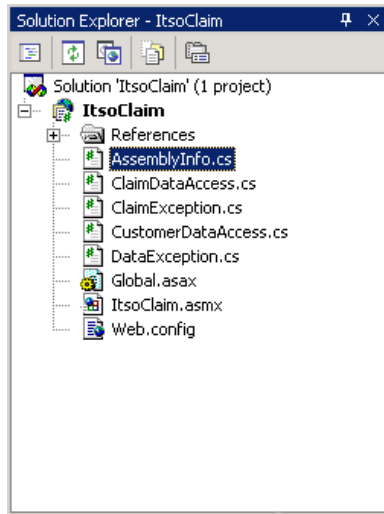


Figure 9-22 The ItsoClaim Project with files imported

9.3.3 Update the Web Service code

The Web Service implementation must be updated to call the existing implementation classes:

1. Select all the generated code in ItsoClaim.asmx (actually ItsoClaim.asmx.cs; see Example 9-1) and replace it with the code in Itsoclaimpaste.txt. This defines a Web service, which includes two Web Methods to call the existing application functionality.

Example 9-1 ItsoClaim.asmx.cs

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace ItsoClaim {
    WebService(Namespace="http://dotnet.claim.examples.itso",
        Name="ItsoClaim")]
    public class ItsoClaim: System.Web.Services.WebService {
        public ItsoClaim() {
            InitializeComponent();
        }
        #region Component Designer generated code
```

```

private IContainer components = null;
private void InitializeComponent() {}
protected override void Dispose( bool disposing ) {
    if(disposing && components != null) {
        components.Dispose();
    }
    base.Dispose(disposing);
}
#endregion
[WebMethod]
public Boolean findCustomer(String customerID,String policyID ) {
    CustomerDataAccess customerObj = new CustomerDataAccess();
    try {
        return customerObj.getCustomer(customerID, policyID);
    } catch (DataException de) {
        throw new ClaimException(de.Message);
    }
}
[WebMethod]
public string registerClaim(String customerID,String policyID,
    DateTime accidentDate, String accidentDescription,
    String [] involvedCars) {
    ClaimDataAccess claimObj = new ClaimDataAccess(customerID, policyID,
        accidentDate,accidentDescription, involvedCars);
    try {
        return claimObj.getClaimCode();
    } catch (DataException de1) {
        throw new ClaimException(de1.Message);
    }
}
}
}
}

```

9.3.4 Building and deploying the Web Service on IIS6

The build process generates proxy classes and the WSDL file that describes the Web service.

To build the project:

1. Select **Build** → **Build Solution** from the top menu.
2. The Output window should display:

Build: 1 succeeded, 0 failed, 0 skipped

Microsoft .NET only uses SOAP Document/Literal binding in the WSDL description, though WS-I profile 1.1 allows RPC/Literal binding. Rational Application Developer, by default, generates Document/Literal binding in the WSDL description.

When Microsoft Visual Studio .NET 2003 rebuilds the Web service, it also publishes the Web service to the Internet Information Server, which resides in the c:\inetpub\wwwroot directory. There are default Web sites and virtual directories that are created when Internet Information Services are installed. You can also use the Internet Information Services Manager to create a new Web site or virtual directory.

Note: Internet Information Services V6.0 with Windows 2003 Server were used for this scenario. You can install Internet Information Services separately by adding the Application Server Windows components in Windows 2003.

To start Internet Information Services Manager:

1. Click the **Start** menu in the bottom left corner of the desktop.
2. Select **All Programs**.
3. Select **Administrative Tools**.
4. Select **Internet Service (IIS) Manager**.

Alternatively, to start the Internet Information Services Manager:

1. Click the **Start** menu.
2. Select **Run**.
3. Type INETMGR.
4. Click **OK**

The tree in Figure 9-23 should now appear.

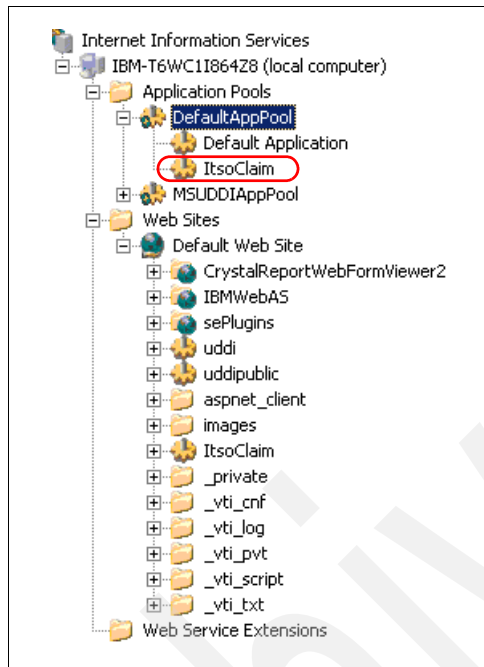


Figure 9-23 IIS Manager shows ItsoClaim on the DefaultAppPool and DefaultWebSite

Note: If the IIS Manager does not show ItsoClaim on the DefaultAppPool and Default Web Site, right-click **DefaultAppPoll** to select **Refresh**. Similarly, right-click **Default Web Site** to select **Refresh**.

9.3.5 Test the Microsoft .NET Web Service

Follow these steps:

1. Once the ItsoClaim Web Service is built and IIS is running, the application can be tested by running a browser on the .NET development system and then opening:

`http://localhost/ItsoClaim/ItsoClaim.asmx?op=findCustomer`

The window in Figure 9-24 on page 321 should appear.

2. Type ABC1234 for the customerID.
3. Type 1234567890 for the policyID.
4. Click **Invoke**.

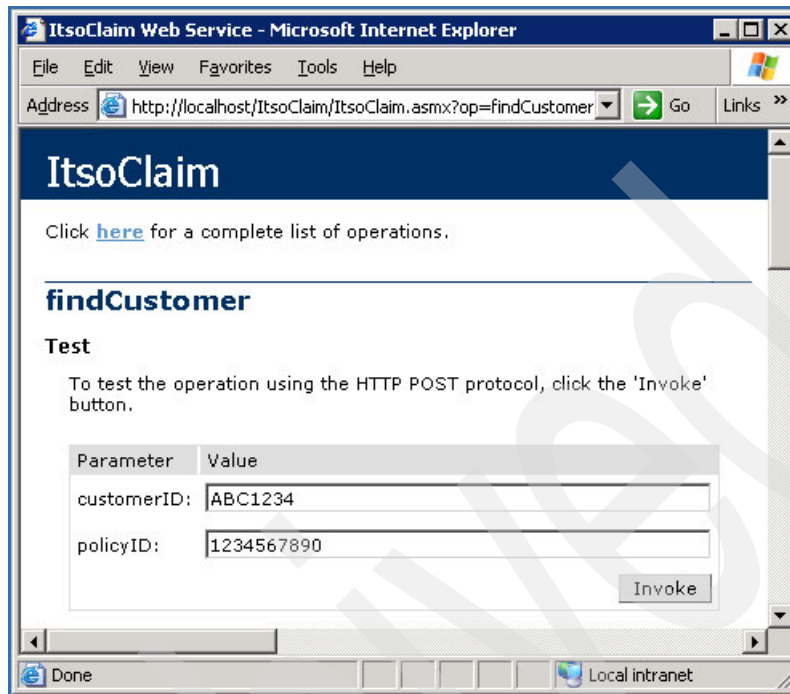


Figure 9-24 Testing findCustomer


This will result in a return value of true, as shown in Figure 9-25.

```
<?xml version="1.0" encoding="utf-8" ?>
<boolean
  xmlns="http://dotnet.claim.examples.itso">true</boolean>
```

Figure 9-25 Result of findCustomer test

Testing the registerClaim method is a bit more complex. The default Microsoft Visual Studio .NET 2003 Test Client does not allow testing of operations with DateTime fields. One approach to testing this operation would be to write a .NET client to test it. Another possibility is to use the Rational Application Developer Web Services Explorer to test the operation:

1. Start **Rational Application Developer**. This can be on the system where the .NET Web Service is running, or on a different system.
1. In Rational Application Developer, select **Run** → **Launch the Web Services Explorer**.

2. If not already on the WSDL page, click the  icon to switch to it. The window in Figure 9-26 should appear.
3. Click **WSDL Main** and enter the WSDL URL:
<http://<dotnethost>/ItsoClaim/ItsoClaim.asmx?WSDL>
4. Click **Go**.

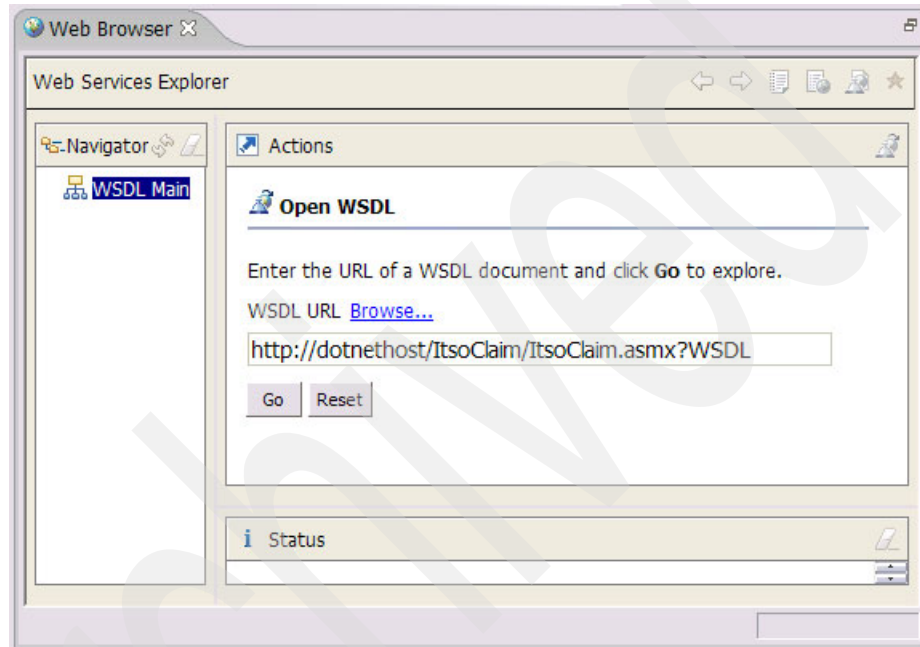


Figure 9-26 Using Web Services Explorer to test .Net Web Service

5. Click the **RegisterClaim** operation.
6. For customerID, click **Add** and type ABC1234.
7. For policyID, click **Add** and type 1234567890.
8. For accidentDate, click **Browse** and select any date in the calendar. A textual representation of the date will be added to the form.
9. For accidentDescription, click **Add** and type in a short description.
10. For involvedCars, click **Add** (to add the content) and then click the nested **Add** twice for the values. Type in short descriptions of the cars.
11. Click **Go**.

Figure 9-27 on page 323 gives an overview of the process.

Actions

Invoke a WSDL Operation

[Source](#)

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

Endpoints

http://dotnethost/ItsoClaim/ItsoClaim.asmx

registerClaim

customerID string [Add](#) [Remove](#)

Values

ABC12345

policyID string [Add](#) [Remove](#)

Values

1234567890

accidentDate dateTime

2005-03-06T18:18:04.586Z

[Browse...](#)

accidentDescription string [Add](#) [Remove](#)

Values

Minor fender bender

involvedCars [Add](#) [Remove](#)

Content

string string [Add](#) [Remove](#)

Values

1999 Corbin Sparrow

2006 Hummer H3

Go

Reset

Figure 9-27 Testing registerClaim on .Net

Chapter 9. Web Services interoperability scenario 323

The status box should display a response with a claim number, as shown in Figure 9-28.

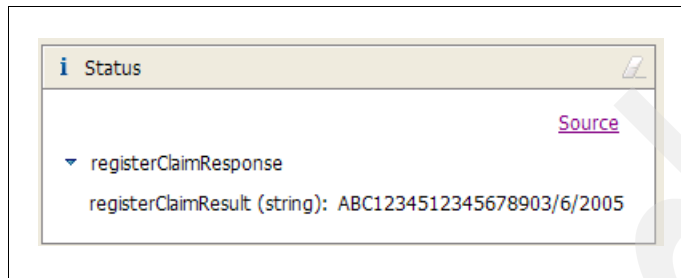


Figure 9-28 Response and claim number

9.4 The client application

Now that we have built wrappers for the insurance claims application in IBM WebSphere and Microsoft .NET, we can update the existing Web application to make use of the Web Services.

A Java 2 Enterprise Edition Web Service client, also known as a service consumer or service requestor, is an application component that acts as the client to Web Services. As in all classical interoperability implementations, the Web Service client component is implemented to wrap the remote invocation to the business logic.

Both WebSphere and Microsoft .NET platforms provide automatic tools for generating the proxy class for wrapping Web Services. To create a proxy class and all other related classes, the same steps are required for both IBM WebSphere and Microsoft .NET Web Services. After building the proxy, each client application that accesses the Web Service need only instantiate the proxy class and invoke the required operations.

As already described, the sample scenario Web Service client communicates with Web Services using SOAP messages. Both the SOAP request and SOAP response messages must follow the format specified in the WSDL file associated with each Web Service. The WSDL file is the specification about the information to be sent in SOAP request and response messages, including all information regarding operations exposed by the Web Service, input and output variables, variable types, document encoding type, and, last but not least, the location of the Web Service. Therefore, the WSDL file is both necessary and sufficient information to develop a client for the corresponding Web Service.

When you build wrappers for an existing Web application to create Web Services, as is the case in this scenario, the WSDL files are different. If the WebSphere and .NET services were being developed from scratch, the preferred approach would be a top-down development method starting with the WSDL definition and creating a single proxy that could be used to access both implementations.

Note: More information about how to use Rational Application Developer to build and test Web Services client or proxy can be found in the redbook *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461.

9.4.1 Import the existing client

We need to import the existing EJB based client application into Rational Application Developer:

1. Launch Rational Application Developer and select **File** → **Import**.
2. Select **Project Interchange** and click **Next**.
3. Browse to **ItsoClaimClient.zip** for the From zip file.
4. Click **Select All** to import both projects.
5. Click **Finish**.

Figure 9-29 gives an overview of the process.

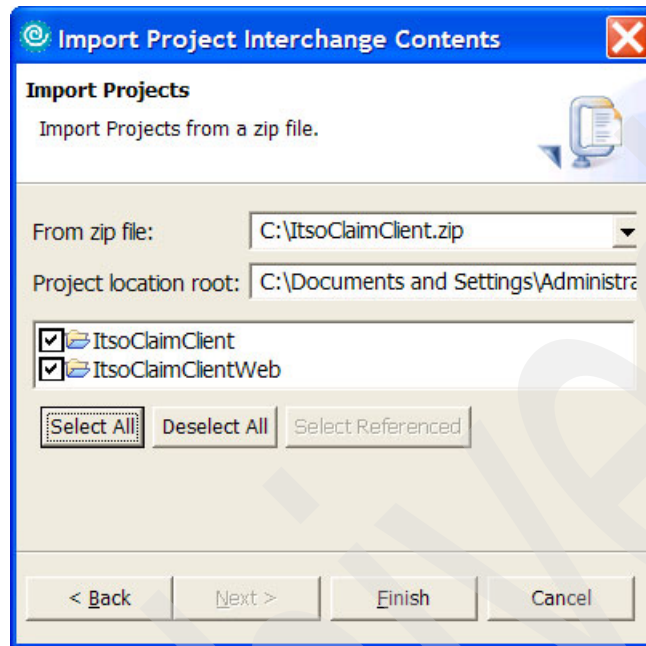


Figure 9-29 Importing the client application

6. Watch the Building Workspace message in the lower right corner of the workbench. When the build completes, there should be no items in the Problems view.

9.4.2 Update the client to use the .NET Web Service

This section illustrates implementation of a client application accessing a Web Service. We update the client application creating a Servlet that will run in the WebSphere Application Server to access the .NET Web Service we implement (business logic tier to business logic tier interoperability). We make use of a browser for interacting with the Servlet (see Figure 9-30 on page 327).

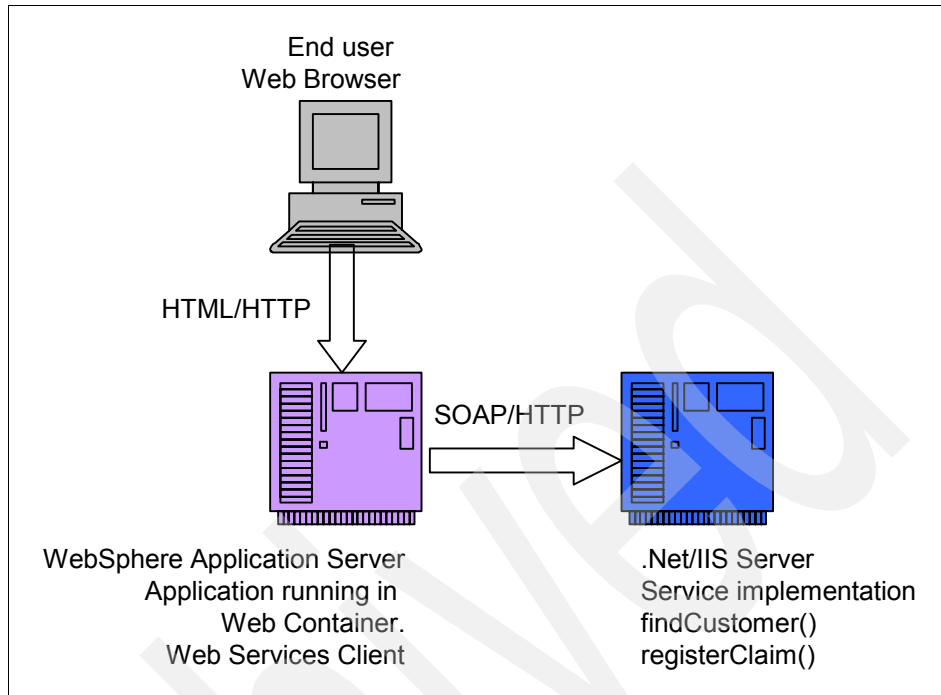


Figure 9-30 First step integrating the applications

Remove EJB based service access

The existing client is EJB based. We must remove the EJB specific code and JARs and replace them with Web Service based access:

1. In the Project Explorer view, expand **Dynamic Web Projects** → **ItsoClaimClientWeb** → **WebContent** → **WEB-INF** → **lib**.
2. Click **ItsoClaimEJB.jar** and press **Delete**. When asked to confirm the deletion, click **Yes** (twice).

Figure 9-31 shows the removal of the EJB jar.

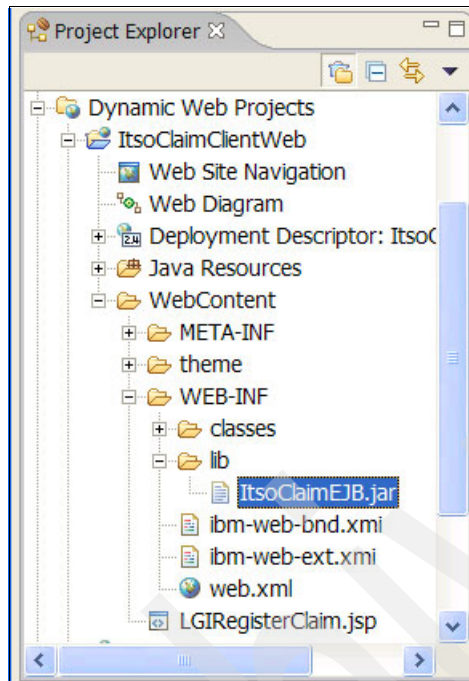


Figure 9-31 Removing the EJB jar

3. Expand **ItsoClaimClientWeb** → **Java Resources** → **JavaSource** → **itso.examples.claim.ejb** and double-click **RegisterClaim.java**.
4. Delete the **import** statements with errors, the **home** field, and the **getClaimRegistration()** method.
5. Delete the statements in the **doGet()** method that are shown in Example 9-2.

Example 9-2 Statements to delete in the doGet() method

```
LGIClaimRegistration ejb = getClaimRegistration();
try {
    String s = ejb.registerClaim(custId, policyId, d,
        accidentDescription, cars);
    result = "Claim " + s + " registered.";
} catch (ClaimException ce) {
    result = "Registration failed: " + ce.getMessage();
}
```

6. Save the file. All errors in the Problems view should be resolved, but there may still be warnings.

Generate the .NET Web Service proxy

Consider the following:

- ▶ Before performing these tasks, a WebSphere Application Server must be configured in the Servers view. The steps to configure a server will vary depending on how Rational Application Developer is installed. See the Rational Application Developer help for detailed instructions.
- ▶ Before generating the Client Proxy, the server should be started and the ItssoClaimClient application should be added to the server.

Perform the following steps:

1. Select **File** → **New** → **Other**. The window in Figure 9-32 should appear.
2. Under Wizards, expand **Web Services** and select **Web Service Client**. Click **Next**.

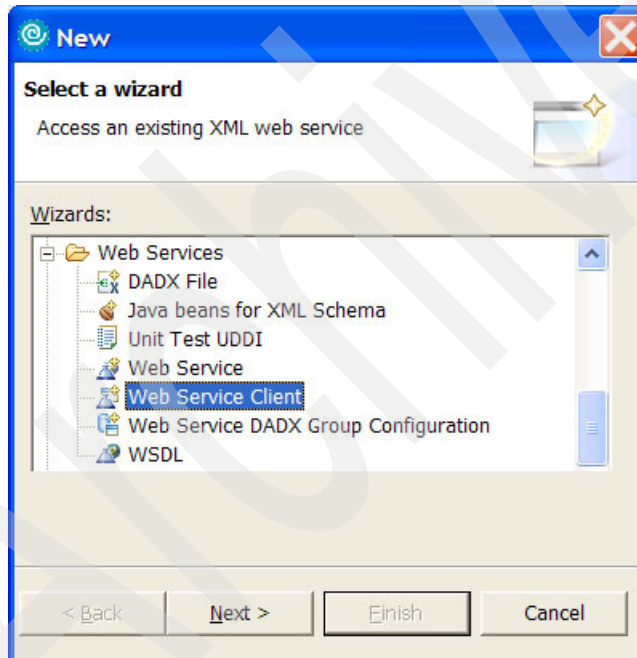


Figure 9-32 Web Services client Wizard

3. On the Web Services page, check **Overwrite files without warning**, and click **Next** (see Figure 9-33).

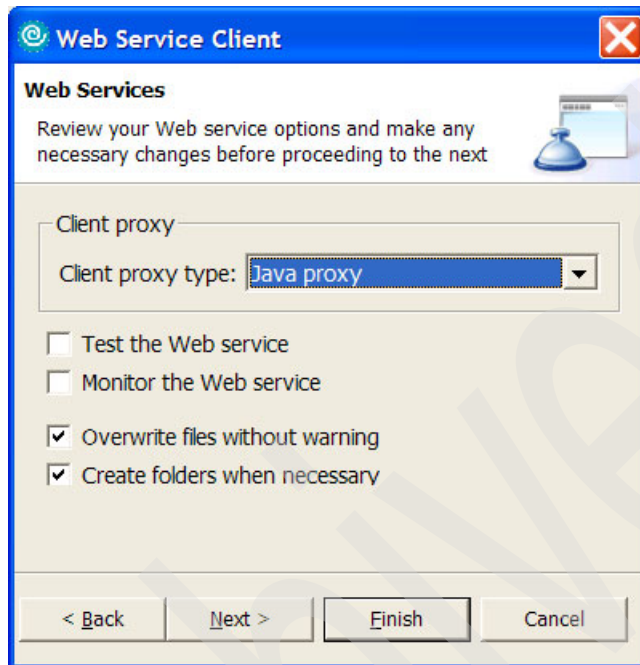


Figure 9-33 Generating a Java Proxy

4. On the Web Service Selection Page, type in the URL `http://<dotnethost>/ItsoClaim/ItsoClaim.asmx?WSDL`.
5. Click **Next**.

Figure 9-34 on page 331 gives an overview of the Web Service Client.

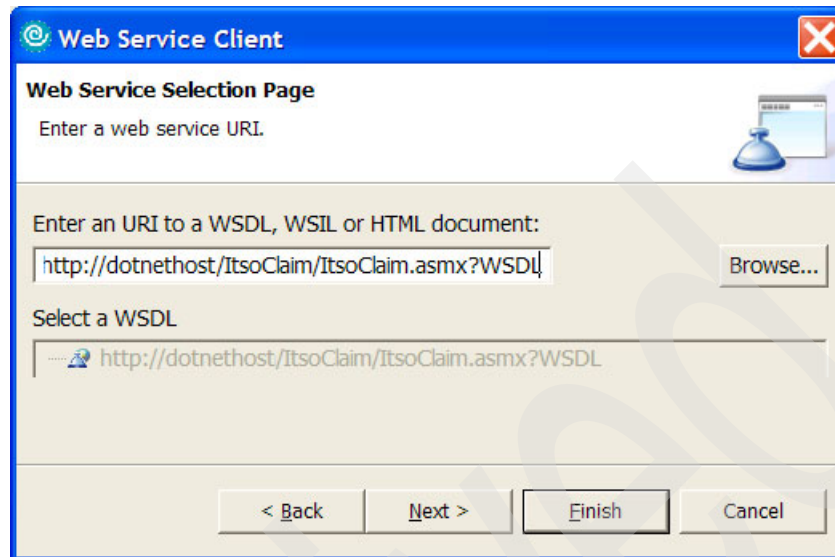


Figure 9-34 Web Service client

6. On the Client Environment Configuration page, leave all the default values and click **Next** (Figure 9-35).

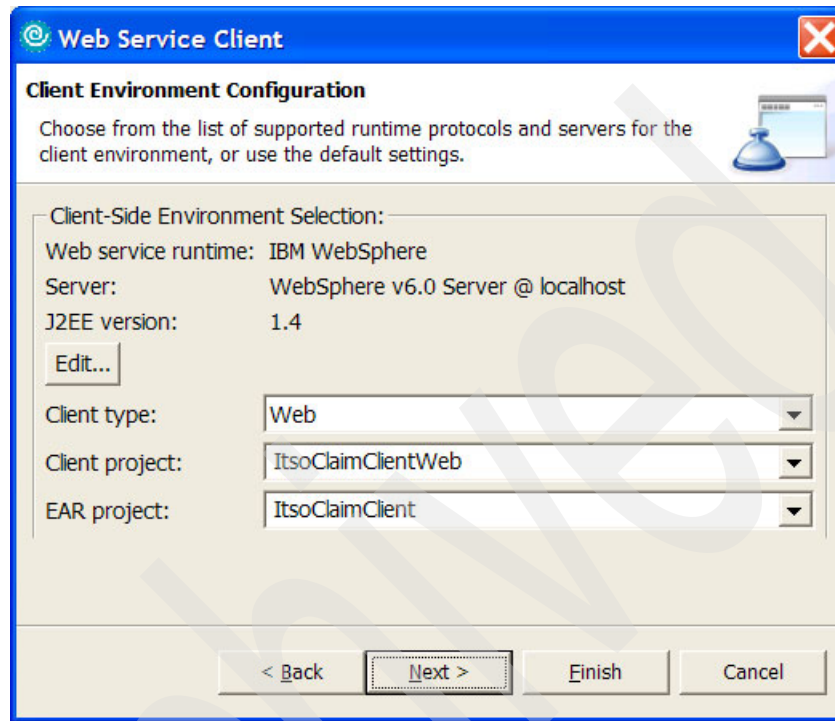


Figure 9-35 Client Environment Configuration

7. On the Web Service Proxy Page, leave all default values and click **Finish**, as shown in Figure 9-36 on page 333.

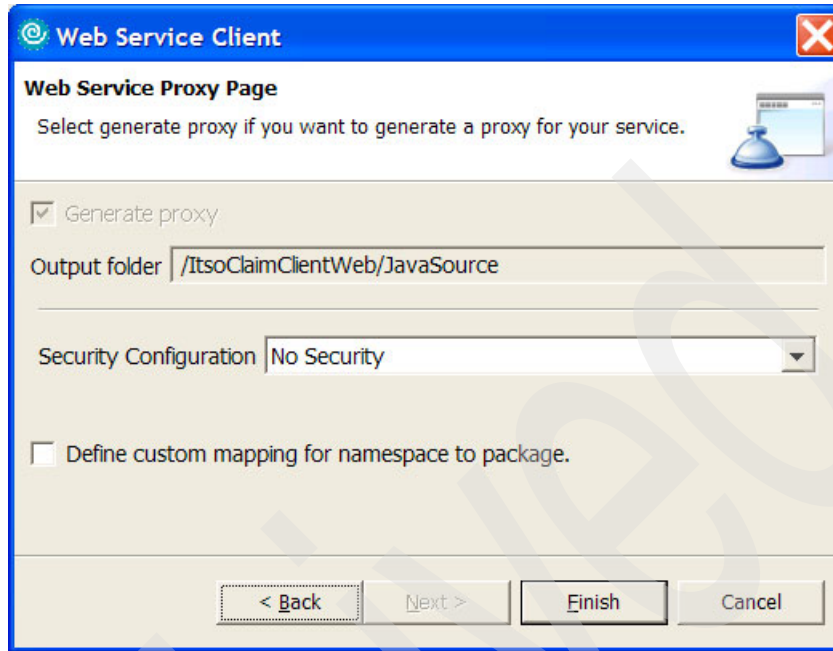


Figure 9-36 Web Service Proxy Page

Generated files

The execution of the wizard results in the creation of the following files in the ItsoClaimClientWeb project:

- ▶ Proxy implementation files:
 - itso.examples.claim.dotnet.ItsoClaim.java
 - itso.examples.claim.dotnet.ItsoClaimInformation.java
 - itso.examples.claim.dotnet.ItsoClaimLocator.java
 - itso.examples.claim.dotnet.ItsoClaimSoap.java
 - itso.examples.claim.dotnet.ItsoClaimSoapProxy.java
 - itso.examples.claim.dotnet.ItsoClaimSoapStub.java
- ▶ A WSDL file that describes the Web service:
WebContent/WEB-INF/wsd/ItsoClaim.asmx.wsdl

- ▶ Deployment descriptor files that describe the Web service according to the Web Services for J2EE style (JSR 109):
 - WebContent/WEB-INF/ibm-webservicesclient-ext.xml
 - WebContent/WEB-INF/ibm-Webservicesclient-bnd.xml
- ▶ The JAX-RPC mapping file:
WebContent/WEB-INF/ltosClaim.asmx_mapping.xml

Figure 9-37 on page 335 shows the generated client files

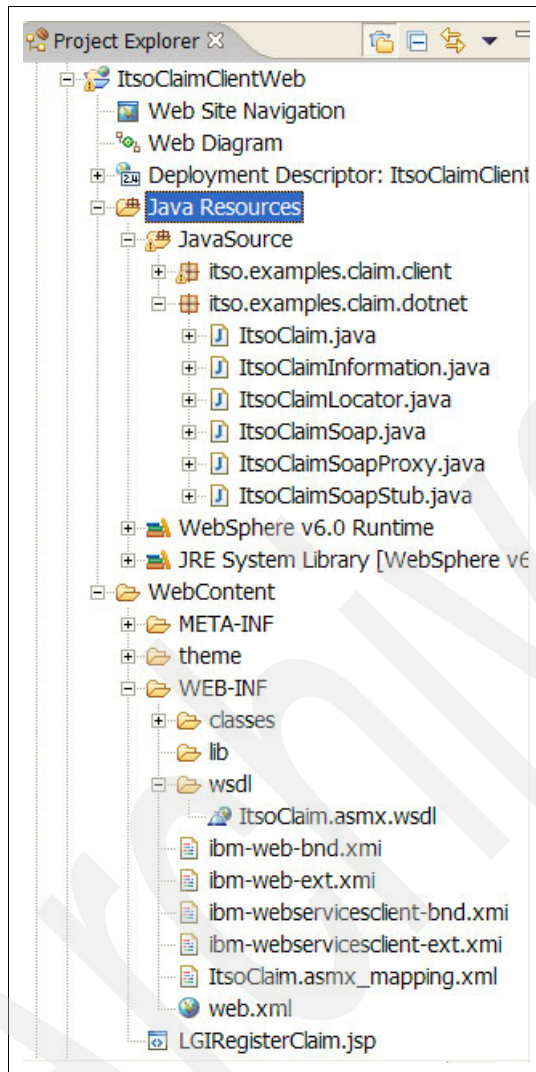


Figure 9-37 Client files generated

Update the Servlet code to use the proxy

To update the Servlet code to use the proxy, perform the following steps:

1. Open **itso.examples.claim.client.RegisterClaim.java**.
2. Add the following import statement:
`import itso.examples.claim.dotnet.ItsoClaimSoapProxy;`
3. Add the shown in Example 9-3 to the end of the `doGet()` method.

Example 9-3 Statements to add in the `doGet()` method

```
String result = null;
ItsoClaimSoapProxy proxy = new ItsoClaimSoapProxy();
try {
    String s = proxy.registerClaim(custId, policyId, cal,
        accidentDescription, cars);
    result = "Claim " + s + " registered.";
} catch (RemoteException re) {
    result = "Registration failed: " + re.toString();
}
req.setAttribute("message", result);
req.getRequestDispatcher("LGIRegisterClaim.jsp").forward(req, resp);
```

4. To resolve the remaining warnings, right-click in the Java Editor and select **Source** → **Organize Imports**.
5. Close and save **RegisterClaim.java**. There should be no errors or warnings associated with this file when this completes.

9.4.3 Test the updated client

Because the `ItsoClaimClient` was added to a WebSphere test server and that server started before generating the client proxy, it is deployed and running and ready to be tested.

1. In a browser on the development system, open:
`http://localhost:9080/ItsoClaimClientWeb/LGIRegisterClaim.jsp`
The window in Figure 9-38 on page 337 should appear.
2. For `customerId`, type `ABC123`.
3. For `policyId`, type `1234567890`.
4. For `accidentDate`, type any correctly formatted date.
5. Type in `accidentDescription`, and descriptions for `car1` and `car2`.
6. Click **Register Claim**.

RegisterClaim	
customerId	ABC123
policyId	1234567890
accidentDate	Jul 13, 2005 (ex: Jan 23, 2005)
accidentDescription	ite Signalled left Then turned right
car1	New Ride
car2	Old Heap
<input type="button" value="Register Claim"/>	

Figure 9-38 Testing .NET service from WebSphere client

A successful request will result in a Claim registered message being added to the form, as shown in Figure 9-39.

RegisterClaim	
customerId	
policyId	
accidentDate	(ex: Jan 23, 2005)
accidentDescription	
car1	
car2	
<input type="button" value="Register Claim"/>	Claim ABC12312345678907/13/2005 registered.

Figure 9-39 Successful claim registration

9.4.4 Update the .NET Service to call WebSphere findCustomer

This section illustrates using a client running as a Servlet in WebSphere Application Server accessing the .NET Web Service implementation. The .NET Web Service implementation will in turn call a Web service running in WebSphere (business logic tier to business logic tier interoperability). There is no direct interaction between the servlet application and the WebSphere Web Service implementation. A browser will be used to interact with the servlet.

Figure 9-40 shows the complete integrated application.

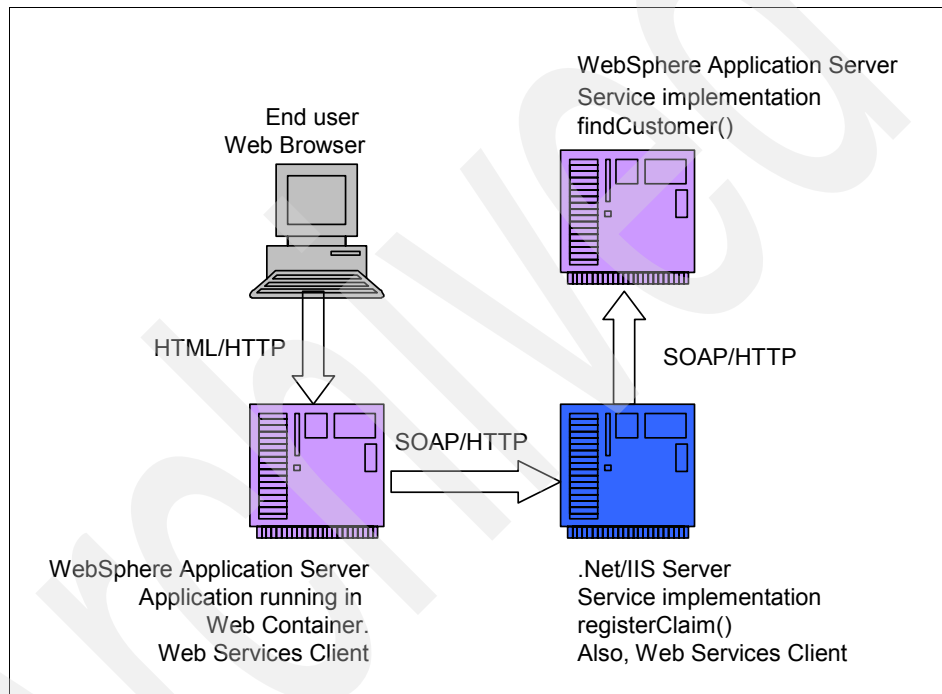


Figure 9-40 The complete integrated application

Add a Web Reference to the ItsoClaim project

Adding a Web Reference is similar to generating a proxy. It imports and builds all the artifacts needed to access the remote service.

1. Launch Visual Studio .NET and open the **ItsoClaim** project.
2. In the Solution Explorer, right-click **References** and select **Add Web Reference**, as shown in Figure 9-41 on page 339.

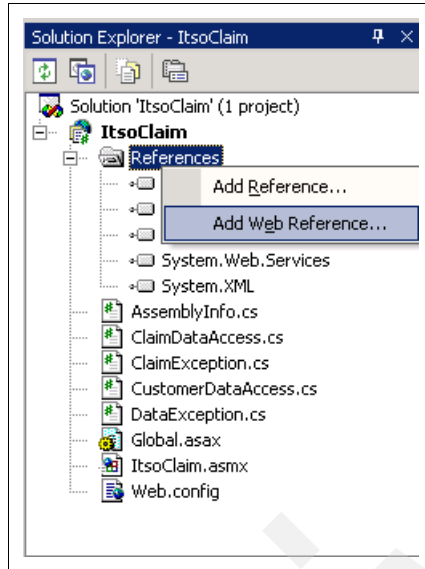


Figure 9-41 Adding WebSphere Service to .Net project

3. Type in the URL:

`http://<washost>:9080/ItsoClaimRouter/services/LGIClaimRegistration/wsd1/LGIClaimRegistration.wsdl`

Where `<washost>` is the host on which the ItsoClaim service is running (see Figure 9-42).

4. Click **Go**.

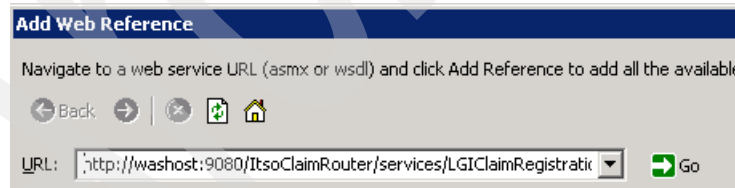


Figure 9-42 Importing the WebSphere Service definition

5. Click the **Add Reference** button to add the reference to the current project.

6. In the Solution Explorer, expand **Web References** and double click the new reference. The Object Browser opens and shows the object associated with the newly imported Web Service reference, as shown in Figure 9-43.

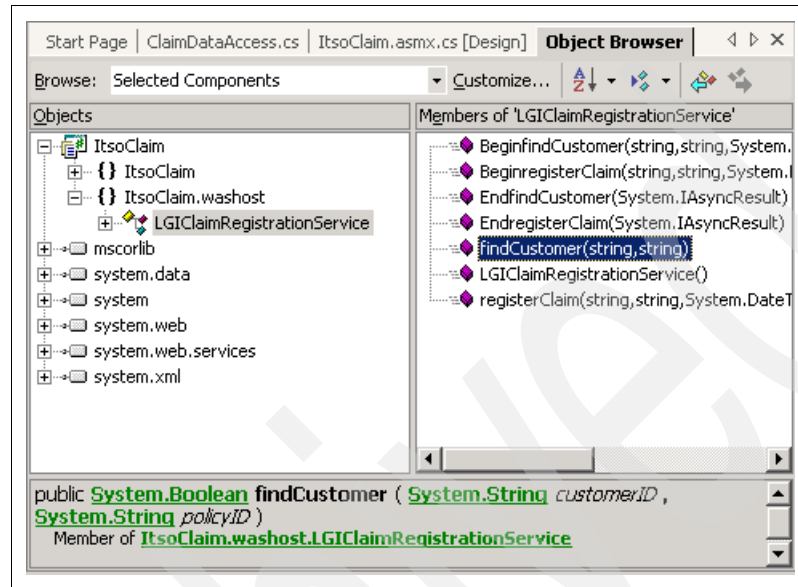


Figure 9-43 The LGIClaimRegistrationService Object

Update the service implementation to call findCustomer

To update the service implementation to call findCustomer:

1. Open the ItsoClaim.asmx file and switch to the code view.
2. In order to check the customer with the WebSphere services, add the statements shown in Example 9-4 to the beginning of the registerClaim method.

Example 9-4 Statements to add to registerClaim method

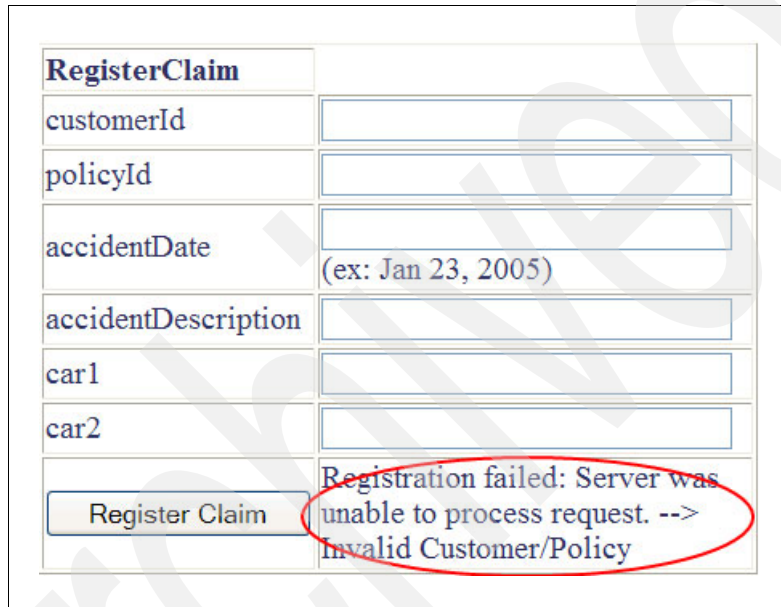
```
washost.LGIClaimRegistrationService service =
    new washost.LGIClaimRegistrationService();
if (!service.findCustomer(customerID, policyID)) {
    throw new ClaimException("Invalid Customer/Policy");
}
```

3. Select **Build** → **Build Solution**. This will compile the project and publish the new version to IIS.

9.4.5 Test the complete solution

The complete solution can be testing by repeating the steps in 9.2.3, “Testing with the Web Services Explorer” on page 306.

Note: Using prior test data that was acceptable for the .NET Web Service registration request (customerId=ABC123, policyId=1234567890), will cause a failure and produce the message shown in Figure 9-44.



RegisterClaim	
customerId	<input type="text"/>
policyId	<input type="text"/>
accidentDate	<input type="text"/> (ex: Jan 23, 2005)
accidentDescription	<input type="text"/>
car1	<input type="text"/>
car2	<input type="text"/>
<input type="button" value="Register Claim"/>	Registration failed: Server was unable to process request. --> Invalid Customer/Policy

Figure 9-44 Registration request data error message

This is because the .NET Web Service implementation now calls the WebSphere Web Service to perform customer lookup before completing the registration. To complete the registration successfully, you must use valid customerId and policyId values for the WebSphere Web Service implementation.

We have so far successfully updated an existing Servlet to use Web Services to access service implementation running on .NET. Similarly, we updated code running in .NET to call the Web Service running in WebSphere. And, we tested the interoperability using a Web browser client application.

9.5 Web Services security

With the WebSphere and .NET Web Service implementations interoperating, the next task is to enable Web Services security. There is a vast array of possible configurations for WS-Security. The particular configurations you apply depends on your security requirements.

We encrypted the service request from the Servlet running in WebSphere to the .NET Web Service and signed the request from the .NET Web Service to the WebSphere Web Service. The responses were neither encrypted or unsigned.

Your Web Services security implementation will almost certainly be more complex and may involve both the digital signing and encryption of all Web Services requests and responses. In this section, we illustrate the methods, considerations, and complexity for implementing both integrity (signing) and confidentiality (encryption) in Web Services security.

Note: In order to perform the tasks in this section, you need to install Web Services Enhancements 2.0 in your .NET development system. For more information about Web Services Enhancements, see:

<http://msdn.microsoft.com/webservices/building/wse/default.aspx>

9.5.1 Encrypting messages to .NET Web Service

Encryption can be used to protect the confidentiality of a request (or a part of a request).

Import the SOAP client certificate on the .NET system

In .NET, X.509 certificates are loaded out of a system store. Before the application can access a certificate, it has to be loaded into the store and the access control list of its private key must be set.

1. Click **Start** → **Run**.
2. Type **mmc** for the name of the file to run and click **OK**.
3. Click **File** → **Add/Remove Snap In**.
4. Click **Add**.
5. Select **Certificates** and click **Add**, as shown in Figure 9-45 on page 343.

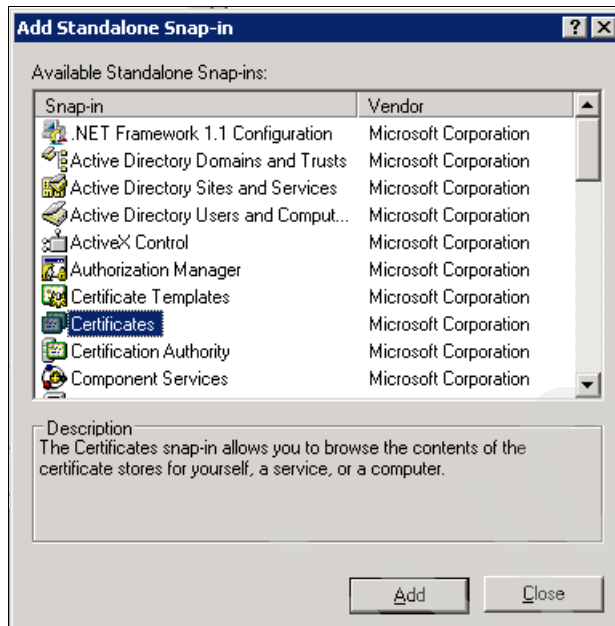


Figure 9-45 Adding the Certificate Snap-in

6. Select **Computer Account** and click **Next**.
7. Select **Local Computer** and click **Finish**.
8. Click **Close** in the Add Standalone Snap-in window.
9. Click **OK** in the Add/Remove Snap-in window.

10. Expand **Certificates**, right click **Personal**, and select **All Tasks** → **Import**, as shown in Figure 9-46.

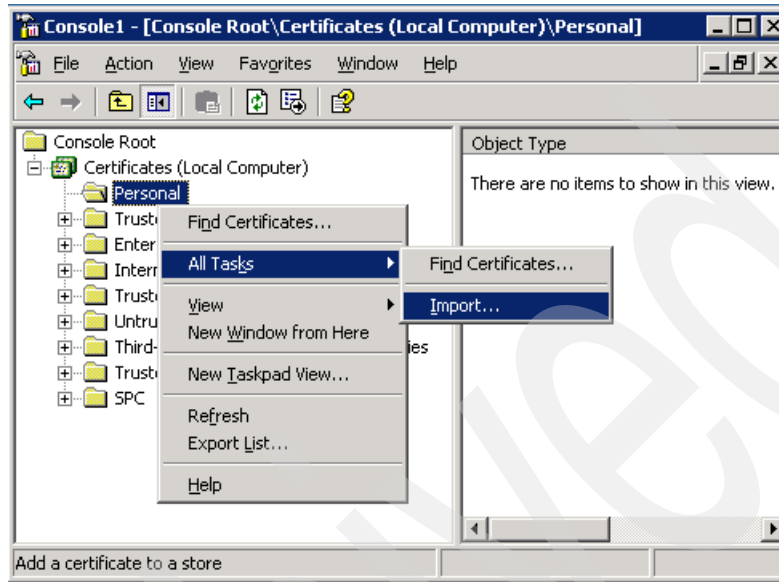


Figure 9-46 Importing the certificates

11. On the Welcome page, click **Next**.
12. On the File to import page, click **Browse**.
13. Change **Files of type** to **Personal Information Exchange** and browse to **enc-receiver.p12**. Click **Open**.
14. Back on the File to import page, click **Next**.
15. On the Password page, type in the password keypass and click **Next**.
16. On the Certificate Store page, leave the default value and click **Next**.
17. Click **Finish** to import the certificate.
18. A message will be displayed that the import was successful.

Allow IIS to access the private key

When you load a certificate, you have access to the certificate's private key. The IIS must also have access to the keys:

1. Start the program by running:
C:\Program Files\Microsoft WSE\v2.0\Tools\Certificates\WseCertificate2.exe
2. Change the Certificate Location to **Local Computer** and the Store Name to **Personal**. Click **Open Certificate**, as shown in Figure 9-47 on page 345.

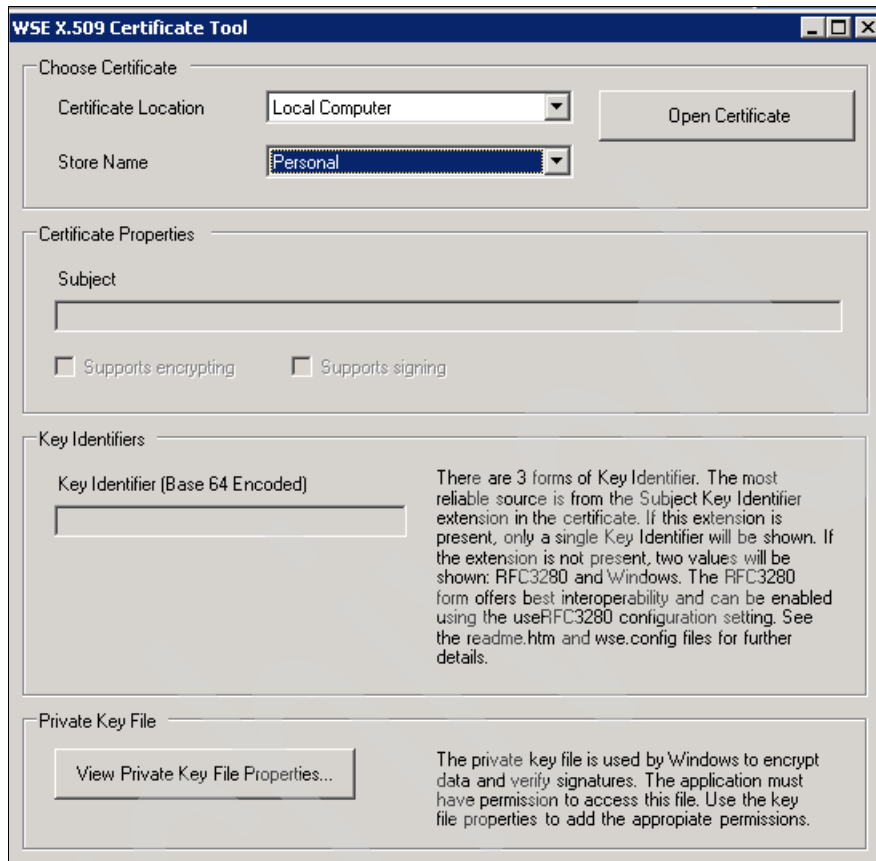


Figure 9-47 Opening the certificate

3. Select the certificate issued to Bob and click **OK**.
4. Click **View Private Key File Properties**.
5. Click the **Security** tab and click **Add**.
6. Add the **ASPNet** user on the local machine and click **OK**.
7. Click **OK** and close the **Certificate Tool**.

Enable Web Service Extensions on the .NET Web Service

Visual Studio .NET projects do not, by default, have access to the libraries needed to perform WS-security operations. This must be explicitly enabled.

1. Launch Visual Studio .NET and open the **ItsoClaim** project.

2. In the Solution Explorer, right click the **ItsoClaim** project and select **WSE Settings 2.0**, as shown in Figure 9-48.

Note: If this menu item does not appear, verify that the Web Services Enhancements 2.0 are correctly installed on the system.

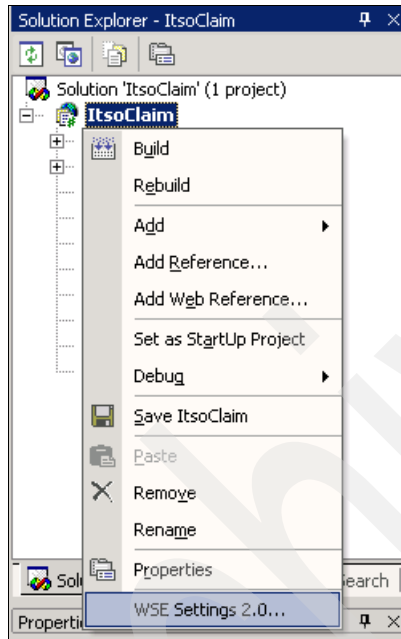


Figure 9-48 Enabling Web Services Enhancements

3. On the General tab, check both the check boxes. Click **OK**, as shown in Figure 9-49 on page 347.



Figure 9-49 Web Service Enhancements General tab

4. Click the **Security** tab.
5. Check **Allow test roots** and **UseRFC3280**. Uncheck **Verify trust**.
6. Click **OK**.

Figure 9-50 shows the Web Service Enhancements Security tab.

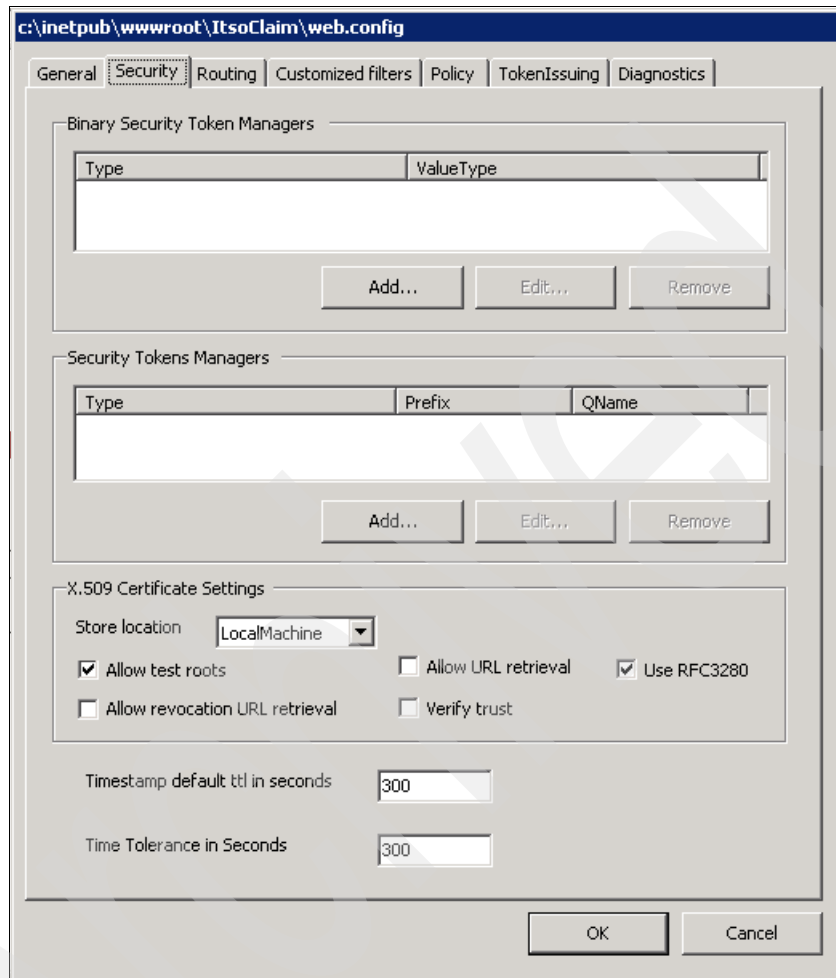


Figure 9-50 Web Service Enhancements Security tab

The .NET ItsoClaim service is now configured to accept encrypted requests.

Enable the WebSphere client to make encrypted requests

To enable the WebSphere client to make encrypted requests:

1. Launch Rational Application Developer and open the workspace containing the ItsoClaimClient application.
2. Expand **Web Services** → **Clients** and double click **ItsoClaimClientWeb**.

3. Select the **WS Extension** tab and expand the **Request Generator Configuration** section.
4. Expand the **Confidentiality** section and click **Add** to bring up the dialog (see Figure 9-51). If the Add button is disabled, be sure that **ItsoClaimSoap** is selected in the Port QName Binding.



Figure 9-51 Web Service Client Security Extensions

5. In the Confidentiality Name textbox, enter Client_Confidentiality.
6. In the Order textbox, enter 2.
7. Under the Message Parts, click the **Add** button. Leave the default value:
 - Message parts dialect:

<http://www.ibm.com/WebSphere/webservices/wssecurity/dialect-was>

- Message parts keyword: bodycontent
 - 8. Click **OK**.
- Figure 9-52 gives an overview of the dialog.

The image shows a 'Confidentiality Dialog' window with a blue title bar and a close button. It contains three main sections: 'Message Parts', 'Nonce', and 'Timestamp'. Each section has a table for configuration and 'Add'/'Remove' buttons. The 'Message Parts' section is currently active, showing a table with 'Message parts dialect' and 'Message parts keyword' columns. The 'Nonce' section has empty rows. The 'Timestamp' section has empty rows and a scroll bar.

Confidentiality Dialog		
Confidentiality Name: Client_Confidentiality		
Order: 2		
Message Parts		
Message parts dialect	Message parts keyword	
webservices/wssecurity/dialect-was	bodycontent	
<input type="button" value="Add"/> <input type="button" value="Remove"/>		
Nonce		
Nonce dialect	Nonce keyword	
<input type="button" value="Add"/> <input type="button" value="Remove"/>		
Timestamp		
Timestamp dialect	Timestamp keyword	Timestamp
<input type="button" value="Add"/> <input type="button" value="Remove"/>		
<input type="button" value="OK"/> <input type="button" value="Cancel"/>		

Figure 9-52 Client Confidentiality Dialog

9. Select the **WS Binding** tab on the bottom of the editor and expand the **Security Request Generator Binding Configuration** section.
10. Expand the **Key Locators** section and click **Add** to bring up the dialog.
11. For the Key locator name, type ClientEncryptKey.

12. For the Key locator class, select:
`com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator`
13. Check the **Use key store** check box.
14. For Key store storepass, type storepass.
15. For Key store path, type:
`${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks`
16. For Key store type, select **JCEKS**.
17. In the Key matrix, click **Add**.
18. For the Alias, type bob.
19. for the Key pass, type keypass.
20. For the Key name, type CN=Bob, O=IBM, C=US.
21. Click **OK**.

Figure 9-53 gives an overview of the data entered.

The Key Locator dialog box contains the following fields and sections:

- Key locator name:** ClientEncryptKey
- Key locator class:** n.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator
- ☒ **Use key store**
- Key store storepass:** storepass
- Key store path:** \${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-
- Key store type:** JCEKS
- Key:**

Alias	Key pass	Key name
bob	keypass	CN=Bob, O=IBM, C=US

Buttons: Add, Remove
- Property:**

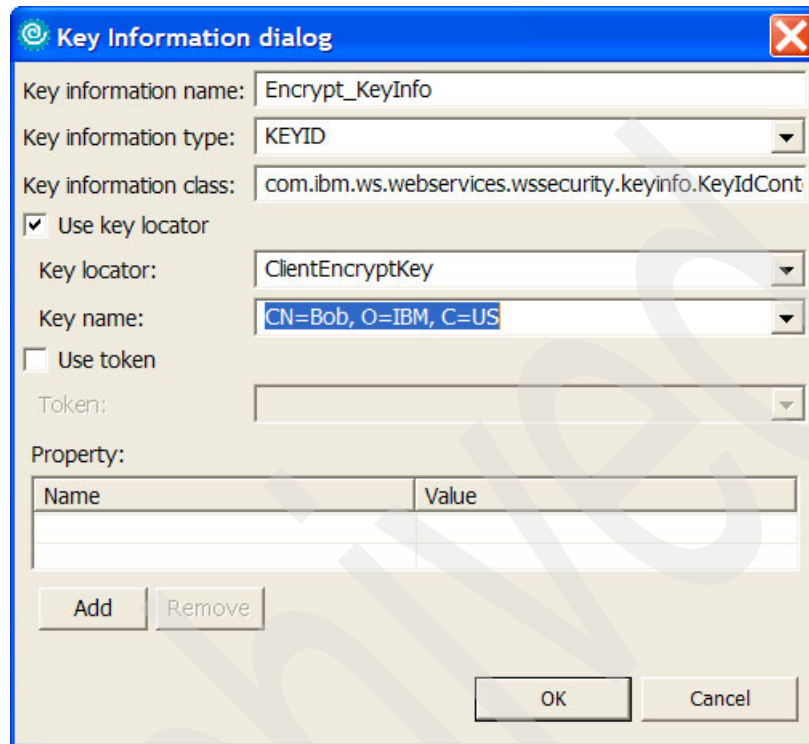
Name	Value
------	-------

Buttons: Add, Remove
- Buttons:** OK, Cancel

Figure 9-53 Client Key locator

22. Expand the **Key Information** section and click **Add**.
23. For the Key information name, type Encrypt_KeyInfo.
24. For the Key information type, select **KEYID**.
25. For the Key information class, leave the value that gets automatically populated:
Com.ibm.ws.webservices.wssecurity.keyinfo.KeyIdContentGenerator
26. Check the **Use key locator** check box.
27. For the Key locator, select **ClientEncryptKey**.
28. For the Key name, select **CN=Bob, O=IBM, C=US**.
29. Click **OK**.

Figure 9-54 gives an overview of the data entered.



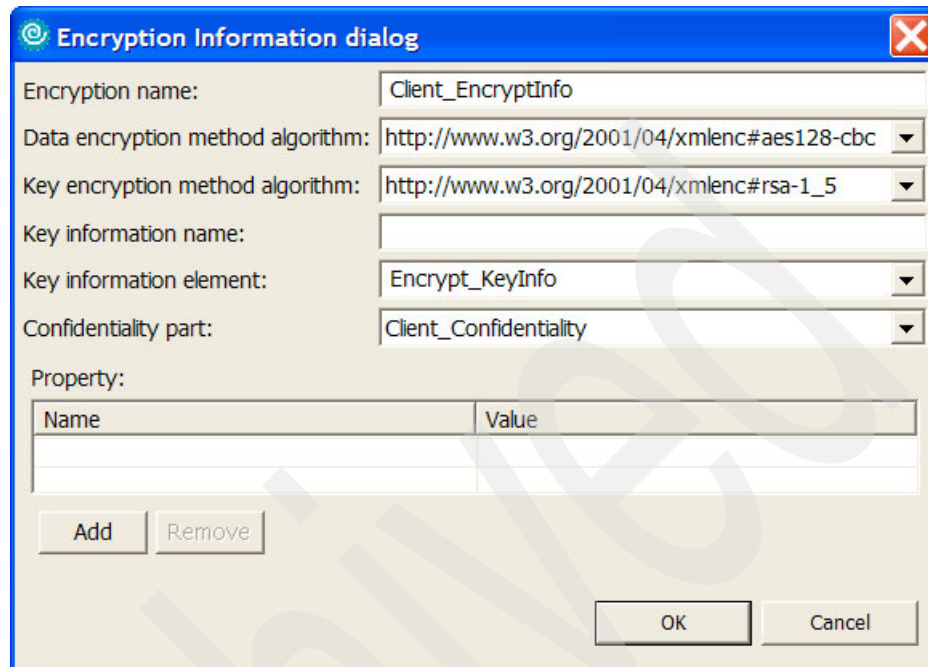
The image shows a 'Key Information dialog' window. It contains several input fields and checkboxes. The 'Key information name' field is set to 'Encrypt_KeyInfo'. The 'Key information type' dropdown is set to 'KEYID'. The 'Key information class' field is set to 'com.ibm.ws.webservices.wssecurity.keyinfo.KeyIdCont'. The 'Use key locator' checkbox is checked, and the 'Key locator' dropdown is set to 'ClientEncryptKey'. The 'Key name' dropdown is set to 'CN=Bob, O=IBM, C=US'. The 'Use token' checkbox is unchecked. The 'Token' dropdown is empty. Below these fields is a 'Property:' section with a table with two columns: 'Name' and 'Value'. The table is currently empty. At the bottom of the dialog are 'Add' and 'Remove' buttons, and at the very bottom are 'OK' and 'Cancel' buttons.

Name	Value
------	-------

Figure 9-54 Client Key information

30. Expand the **Encryption Information** section and click **Add**.
31. For the Encryption name, type Client_EncryptInfo.
32. For the Data encryption method algorithm, select:
<http://www.w3.org/2001/04/xmlenc#aes128-cbc>
33. For the Key encryption method algorithm, select:
http://www.w3.org/2001/04/xmlenc#rsa-1_5
34. For the Key information element, select **Encrypt_KeyInfo**.
35. For the Confidentiality part, select **Client_Confidentiality**.
36. Click **OK**.

Figure 9-55 gives an overview of the data entered.



The image shows a Windows-style dialog box titled "Encryption Information dialog". It contains several fields and dropdown menus for configuring encryption. The fields are: "Encryption name" (text box with "Client_EncryptInfo"), "Data encryption method algorithm" (dropdown with "http://www.w3.org/2001/04/xmlenc#aes128-cbc"), "Key encryption method algorithm" (dropdown with "http://www.w3.org/2001/04/xmlenc#rsa-1_5"), "Key information name" (empty text box), "Key information element" (dropdown with "Encrypt_KeyInfo"), and "Confidentiality part" (dropdown with "Client_Confidentiality"). Below these is a "Property:" section with a table with two columns: "Name" and "Value". The table is currently empty. At the bottom of the table are "Add" and "Remove" buttons. At the bottom right of the dialog are "OK" and "Cancel" buttons.

Name	Value
------	-------

Figure 9-55 Client Encryption Information

37. The client is now configured to encrypt the message body. Close and save the Web Deployment Descriptor editor.

The client application can now be deployed into WebSphere Application Server and tested, as shown in 9.2.3, "Testing with the Web Services Explorer" on page 306.

9.5.2 Signing requests from .NET to WebSphere Web Services

Signing can be used to ensure that a service request (or a part of the request) has not been modified between the sender and receiver, and to prove the identity of the sender. Signing is often used in conjunction with encryption.

Update the WebSphere Web Service to use signed messages

The WebSphere Web Service implementation must be updated to require and accept signed requests:

1. Launch Rational Application Developer and open the workspace containing the ItsoClaim application.

2. Expand **Web Services** → **Services** and double click **LGIClaimRegistrationService**, as shown in Figure 9-56.

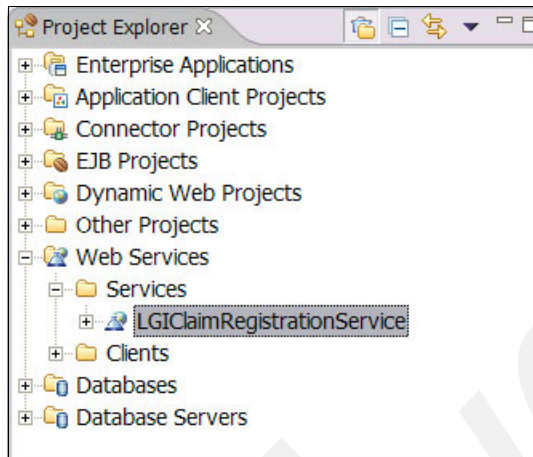


Figure 9-56 Opening the Web Services editor

3. Select the **Extension** tab.
4. In the Web Service Description Extension and Port Component Binding sections, you should already see LGIClaimRegistrationService.
5. Expand the **Request Consumer Service Configuration Details** section.

6. Expand the **Required Integrity** section and click **Add** to bring up the dialog, as shown in Figure 9-57. If the Add button is disabled, be sure that **LGIClaimRegistrationService** is selected in the Port Component Binding section.

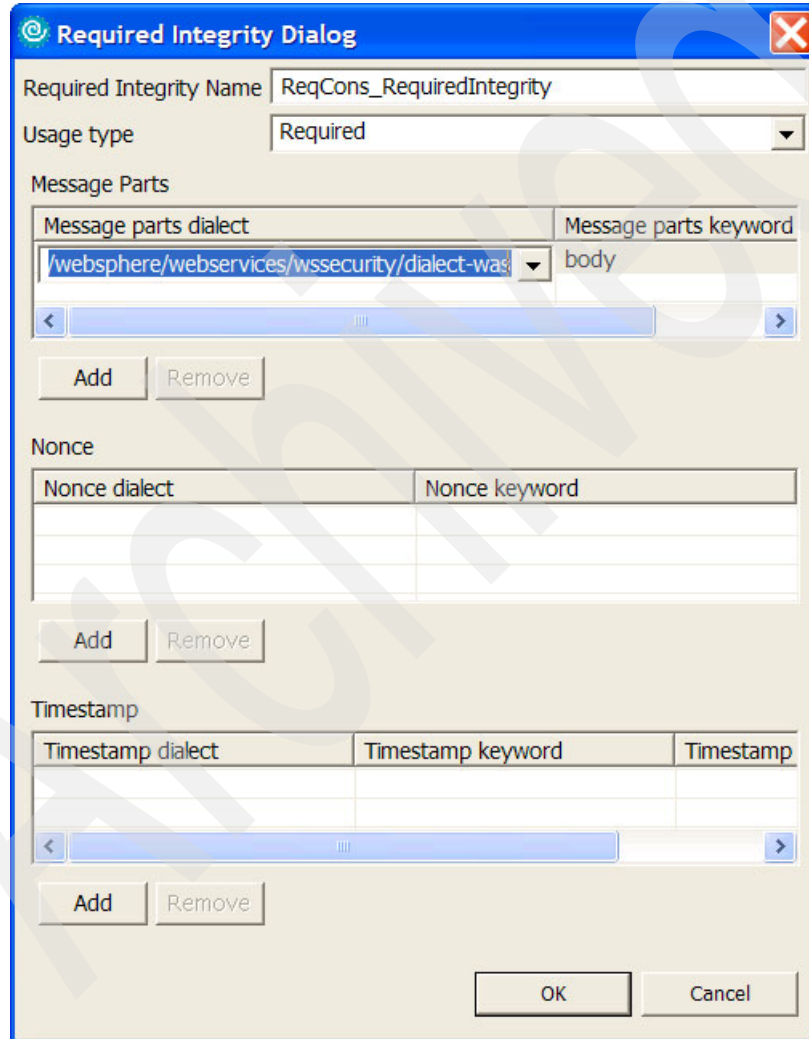


Figure 9-57 Adding Required Integrity

7. In the Required Integrity Name text field, enter ReqCons_RequiredIntegrity.
8. Select **Required** from the Usage type drop-down list.

9. In the Message Parts matrix, click **Add** to enter or select the following:
Message parts dialect:
<http://www.ibm.com/WebSphere/webservices/wssecurity/ dialect-was>
Message parts keyword: body
10. Click **OK**.

Figure 9-58 gives an overview of the settings.



The image shows a 'Required Integrity Dialog' window. At the top, the title bar says 'Required Integrity Dialog'. Below the title bar, there are two input fields: 'Required Integrity Name' with the value 'ReqCons_RequiredIntegrity' and 'Usage type' with a dropdown menu set to 'Required'. Below these is a section titled 'Message Parts'. It contains a table with two columns: 'Message parts dialect' and 'Message parts keyword'. The first row has the value '/websphere/webservices/wssecurity/dialect-was' in the first column and 'body' in the second column. Below the table are 'Add' and 'Remove' buttons. Below the 'Message Parts' section is a section titled 'Nonce'. It contains a table with two columns: 'Nonce dialect' and 'Nonce keyword'. There are two empty rows in this table. Below the table are 'Add' and 'Remove' buttons. Below the 'Nonce' section is a section titled 'Timestamp'. It contains a table with three columns: 'Timestamp dialect', 'Timestamp keyword', and 'Timestamp'. There are two empty rows in this table. Below the table are 'Add' and 'Remove' buttons. At the bottom right of the dialog are 'OK' and 'Cancel' buttons.

Message parts dialect	Message parts keyword
/websphere/webservices/wssecurity/dialect-was	body

Nonce dialect	Nonce keyword

Timestamp dialect	Timestamp keyword	Timestamp

Figure 9-58 Request required integrity

11. Select the **Binding Configurations** tab on the bottom of the editor. The service should already appear in the Port Component Binding section.
12. Expand the **Request Consumer Binding Configuration Details** section.
13. Expand the **Trust Anchor** section and click **Add**. Enter or select the following statements into the fields shown in Figure 9-59:
 - Trust anchor name: TrustAnchor
 - Key store storepass: server
 - Key store path: `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks`
 - Key store type: JKS

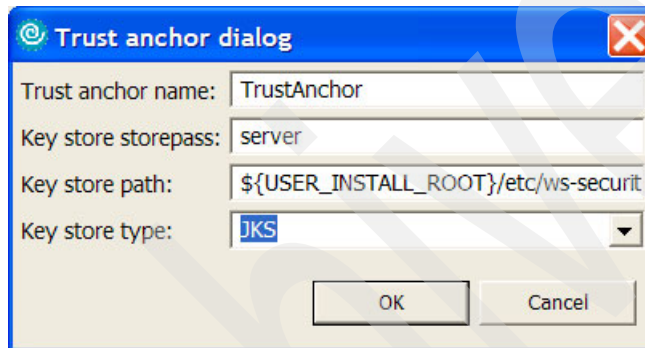


Figure 9-59 Trust anchor dialog

14. Click **OK**.
15. Under Request Consumer Binding Configuration Details, expand **Certificate Store List** → **Collection Certificate Store** and click **Add**. The window in Figure 9-60 on page 359 should appear.
16. For the Name, type CertStore.
17. For the Provider, type IBM CertPath.
18. In the X509 Certificate matrix, click **Add** and enter `${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer` for the path.
19. Click **OK**.

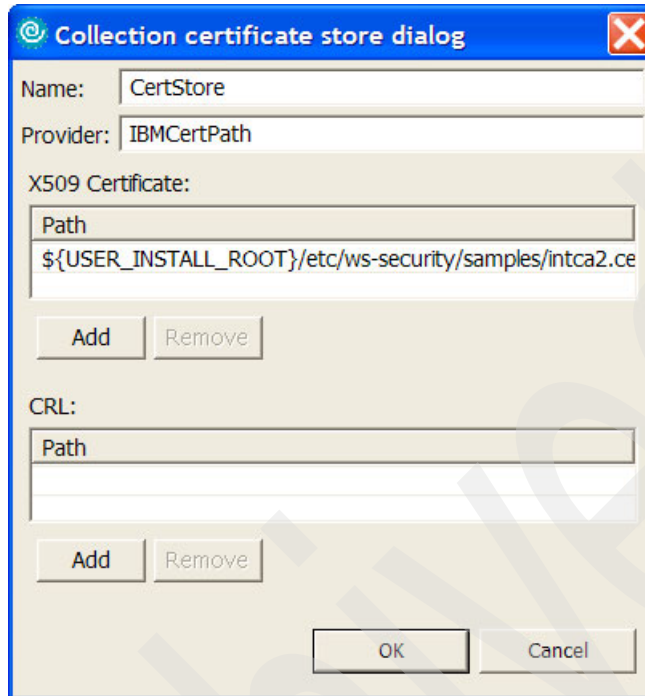


Figure 9-60 Collection certificate store dialog

20. Under Request Consumer Binding Configuration Details, expand the **Token Consumer** section and click **Add**.
21. For the Token consumer name, type Signature_TokenConsumer.
22. For the Token consumer class, select
com.ibm.wsspi.wssecurity.token.X509TokenConsumer.
23. Check the **Use value type** check box
24. For the Value type, select **X509 certificate token v3**.
25. For the Local name, the following should be automatically filled in:
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3>.
26. Check the **Use jaas.config** check box.
27. For the jaas.config name, type system.wssecurity.X509BST.
28. Check the **Use certificate path settings** check box.
29. Check the **Certificate path reference** radio button.
30. For the Trust anchor reference, select **TrustAnchor**.

31. For the Certificate store reference, select **CertStore**.

32. Click **OK**.

Figure 9-61 gives an overview of the data entered.

The image shows a 'Token Consumer dialog' window. The 'Token consumer name' is 'Signature_TokenConsumer' and the 'Token consumer class' is 'com.ibm.wsspi.wsssecurity.token.X509TokenConsumer'. The 'Security token' field is empty. The 'Use value type' checkbox is checked, with 'Value type' set to 'X509 certificate token v3' and 'Local name' set to 'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3'. The 'URI' field is empty. The 'Use jaas.config' checkbox is checked, with 'jaas.config name' set to 'system.wsssecurity.X509BST'. Below this is a table for 'jaas.config property:' with columns 'Name' and 'Value'. The 'Use trusted ID evaluator' checkbox is unchecked, with 'Trusted ID evaluator class:' empty. Below this is a table for 'Trusted ID evaluator property:' with columns 'Name' and 'Value'. The 'Use trusted ID evaluator reference' checkbox is unchecked, with 'Trusted ID evaluator reference:' empty. Below this is a table for 'Property:' with columns 'Name' and 'Value'. The 'Use certificate path settings' checkbox is checked, with 'Certificate path reference' selected. Below this, 'Trust anchor reference' is set to 'TrustAnchor' and 'Certificate store reference' is set to 'CertStore'. The 'Trust any certificate' radio button is unselected. At the bottom right are 'OK' and 'Cancel' buttons.

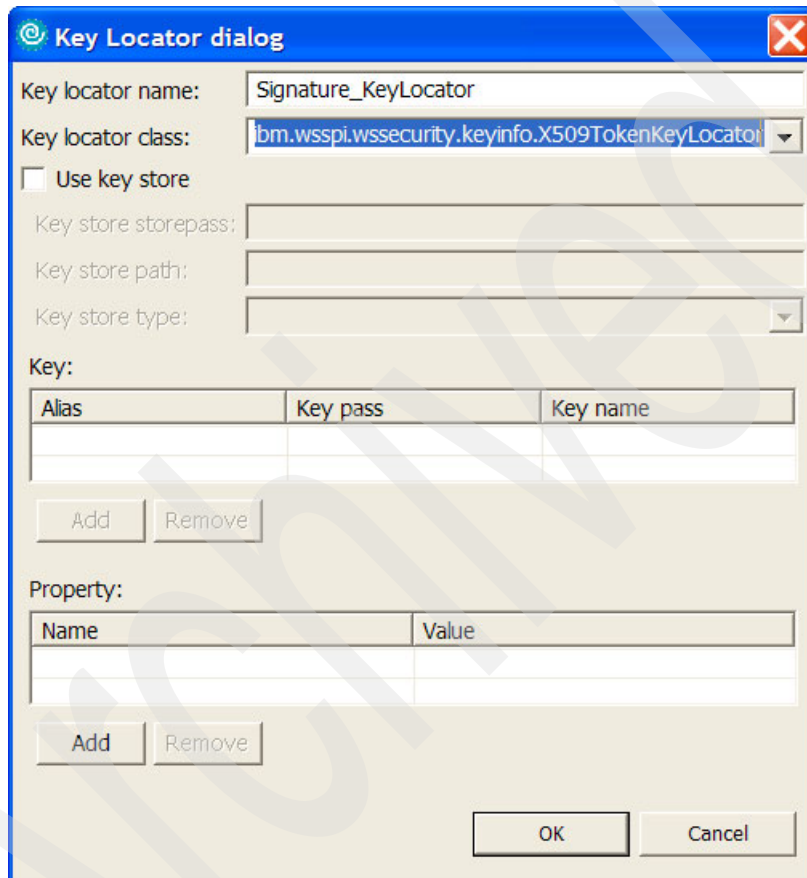
Figure 9-61 Token Consumer Dialog

33. Under Request Consumer Binding Configuration Details, expand the **Key Locator** section and click **Add**.

34. In the Key locator name text field, enter Signature_KeyLocator.

35. In the Key locator class text field, select `com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator`.
36. Click **OK**.

Figure 9-62 shows the Key Locator dialog.



The Key Locator dialog box is shown with the following fields and controls:

- Key locator name:** `Signature_KeyLocator`
- Key locator class:** `com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator` (selected from a dropdown)
- ☐ **Use key store**
- Key store storepass:** (empty text field)
- Key store path:** (empty text field)
- Key store type:** (empty dropdown menu)
- Key:**

Alias	Key pass	Key name
- Property:**

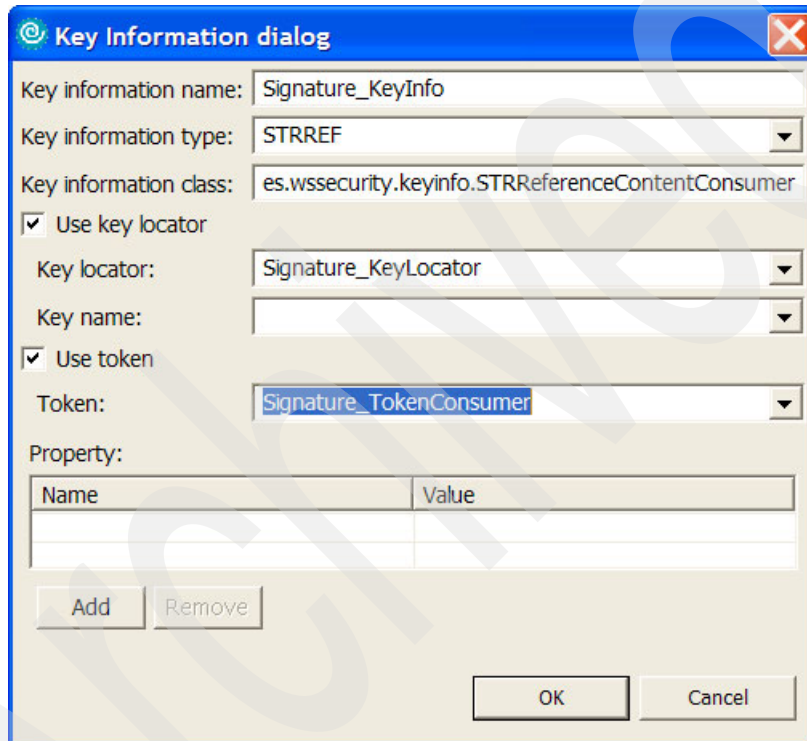
Name	Value
-

Figure 9-62 Key Locator

37. Under Request Consumer Binding Configuration Details, expand the **Key Information** section and click **Add**.
38. For the Key information name, type `Signature_KeyInfo`.
39. For the Key information type, select **STRREF**.
40. The Key information class should be automatically filled in with `com.ibm.ws.webservices.wssecurity.keyinfo.STRReferenceContentConsumer`.

41. Check **Use key locator** check box.
42. For Key locator, select **Signature_KeyLocator**.
43. Check **Use token** check box.
44. For Token, select **Signature_TokenConsumer**.
45. Click **OK**.

Figure 9-63 shows the Key Information dialog.



The Key Information dialog box is shown with the following configuration:

- Key information name: Signature_KeyInfo
- Key information type: STRREF
- Key information class: es.wssecurity.keyinfo.STRReferenceContentConsumer
- ☒ Use key locator
 - Key locator: Signature_KeyLocator
 - Key name: (empty)
- ☒ Use token
 - Token: Signature_TokenConsumer
- Property:

Name	Value

Buttons: Add, Remove, OK, Cancel

Figure 9-63 Key information

46. Under Request Consumer Binding Configuration Details, expand the **Signing Information** section and click **Add**.
47. For the Signing information name, type ReqCons_SigningInfo.
48. For the Canonicalization method algorithm, select the following:
<http://www.w3.org/2001/10/xml-exc-c14n#>
 This is the first option in the drop-down box.

49. For the Signature method algorithm, select:

<http://www.w3.org/2000/09/xmldsig#rsa-sha1>

50. In the Signing key information matrix, click **Add**.

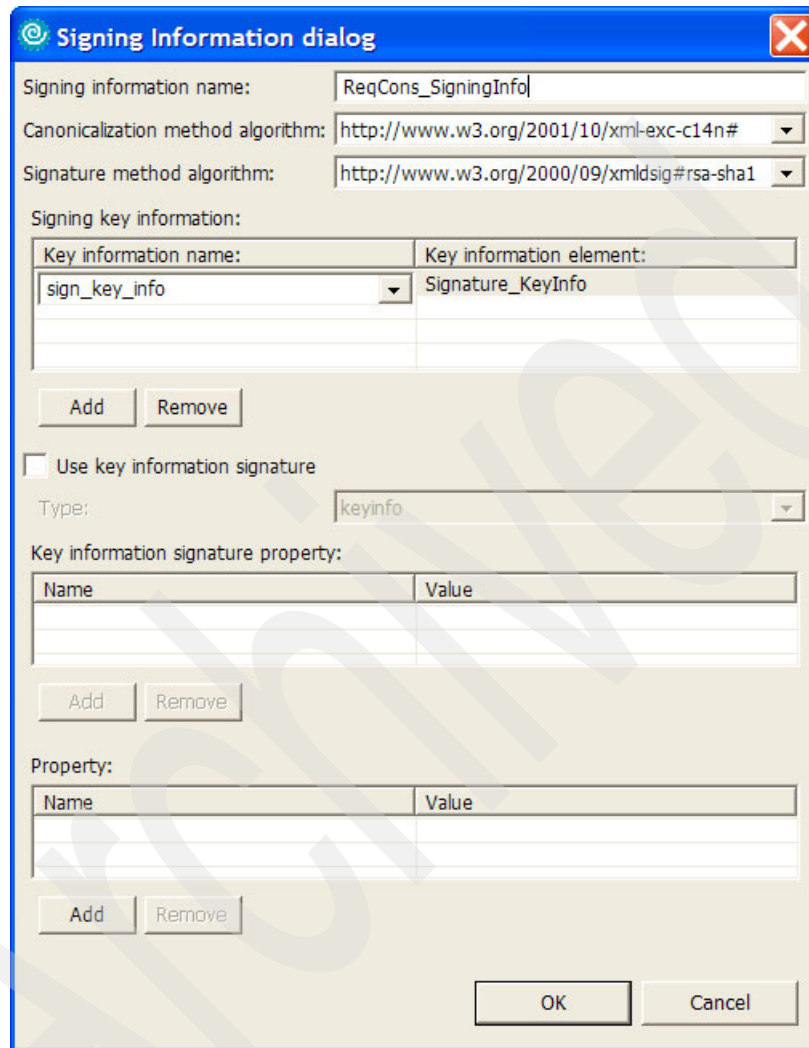
51. For the Key information name, type sign_key_info.

52. For the Key information element, select **Signature_KeyInfo**.

53. Click **OK**.

Archived

Figure 9-64 shows the Signing Information dialog.



The image shows a 'Signing Information dialog' window. It contains several sections for configuring signing information. The 'Signing information name' is 'ReqCons_SigningInfo'. The 'Canonicalization method algorithm' is 'http://www.w3.org/2001/10/xml-exc-c14n#'. The 'Signature method algorithm' is 'http://www.w3.org/2000/09/xmldsig#rsa-sha1'. The 'Signing key information' section has a table with one entry: 'sign_key_info' as the key information name and 'Signature_KeyInfo' as the key information element. Below this table are 'Add' and 'Remove' buttons. There is a checkbox for 'Use key information signature' which is currently unchecked. The 'Type' is set to 'keyinfo'. The 'Key information signature property' section has a table with two columns: 'Name' and 'Value'. Below this table are 'Add' and 'Remove' buttons. The 'Property' section also has a table with 'Name' and 'Value' columns, and 'Add' and 'Remove' buttons. At the bottom right are 'OK' and 'Cancel' buttons.

Signing information name: ReqCons_SigningInfo

Canonicalization method algorithm: http://www.w3.org/2001/10/xml-exc-c14n#

Signature method algorithm: http://www.w3.org/2000/09/xmldsig#rsa-sha1

Signing key information:

Key information name:	Key information element:
sign_key_info	Signature_KeyInfo

Add Remove

☐ Use key information signature

Type: keyinfo

Key information signature property:

Name	Value
------	-------

Add Remove

Property:

Name	Value
------	-------

Add Remove

OK Cancel

Figure 9-64 Signing information

54. With the newly created signing information highlighted, expand the **Part Reference** section and click **Add**.

55. For the Part reference name, select **ReqCons_PartRef**.

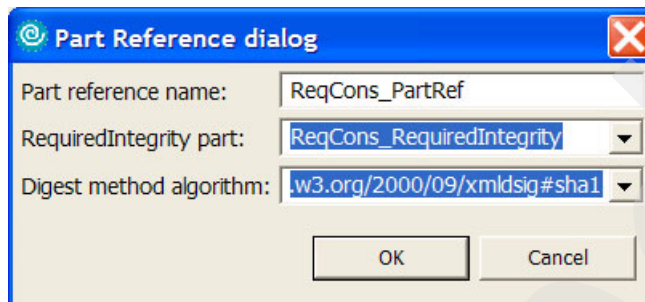
56. For the RequiredIntegrity part, select **ReqCons_RequiredIntegrity**.

57. For the Digest method algorithm, select:

http://www.w3.org/2000/09/xmldsig#sha1

58. Click **OK**

Figure 9-65 shows the Part Reference dialog.



The Part Reference dialog box has a blue title bar with a gear icon and a close button. It contains three labeled text fields: 'Part reference name:' with the value 'ReqCons_PartRef', 'RequiredIntegrity part:' with a dropdown menu showing 'ReqCons_RequiredIntegrity', and 'Digest method algorithm:' with a dropdown menu showing 'w3.org/2000/09/xmldsig#sha1'. At the bottom are 'OK' and 'Cancel' buttons.

Figure 9-65 Part Reference

59. With the newly created part reference highlighted, expand the **Transform section** and click **Add**.

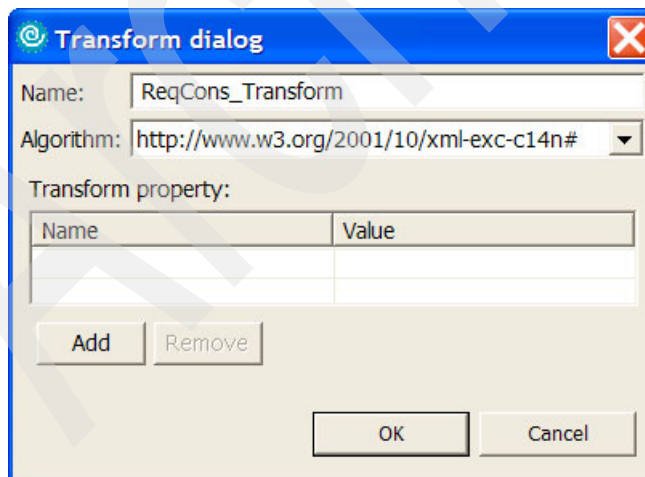
60. For the Name, type ReqCons_Transform.

61. For the Algorithm, select:

`http://www.w3.org/2001/10/xml-exc-c14n#`

62. Click **OK**.

Figure 9-66 shows the Transform dialog.



The Transform dialog box has a blue title bar with a gear icon and a close button. It contains two labeled text fields: 'Name:' with the value 'ReqCons_Transform' and 'Algorithm:' with a dropdown menu showing 'http://www.w3.org/2001/10/xml-exc-c14n#'. Below these is a section titled 'Transform property:' containing a table with two columns, 'Name' and 'Value'. The table is currently empty. Below the table are 'Add' and 'Remove' buttons. At the bottom are 'OK' and 'Cancel' buttons.

Name	Value

Figure 9-66 Transform

63. Save and close the Web Services editor. The service can now be deployed to WebSphere, as in 9.2.3, “Testing with the Web Services Explorer” on page 306.

Enable the .NET Web Service client to sign requests

Now that the WebSphere service requires request signing, the .NET Web Service that calls it must be updated to sign its requests.

Import the signing certificate

To import the signing certificate:

1. Open the **Certificates (Local Computer)** MMC snap-in, as in 9.5.1, “Encrypting messages to .NET Web Service” on page 342.
2. Expand **Certificates**, right-click **Personal**, and select **All Tasks** → **Import**.
3. On the Welcome page, click **Next**.
4. On the File to import page, click **Browse**.
5. Change **Files of type** to **Personal Information Exchange** and browse to **soaprequester.p12**. Click **Open**.
6. Back on the File to import page, click **Next**.
7. On the Password page, type in the password client and click **Next**.
8. On the Certificate Store page, leave the default value and click **Next**.
9. Click **Finish** to import the certificate.
10. A message will be displayed that the import was successful.

Allow IIS to access the private key

To allow IIS to access the private key:

1. Start the program:
C:\Program Files\Microsoft WSE\v2.0\Tools\Certificates\WseCertificate2.exe
2. Change the Certificate Location to **Local Computer** and the Store Name to **Personal**. Click **Open Certificate**.
3. Select the certificate issued to **SOAPRequester** and click **OK**.
4. Click **View Private Key File Properties**.
5. Click the **Security** tab and click **Add**.
6. Add the **ASPNet** user on the local machine and click **OK**.
7. Click **OK** and close the **Certificate Tool**.

Update the ItsoClaim application

The code that calls the WebSphere service must be updated to send a signed request.

1. Add the code shown in Example 9-5 to the top of ItsoClaim.asmx.cs.

Example 9-5 Statements to add to ItsoClaim.asmx.cs

```
using System;

using Microsoft.Web.Services2.Security;
using Microsoft.Web.Services2.Security.X509;
using Microsoft.Web.Services2.Security.Tokens;

using System.Collections;
```

2. Modify the registerClaim method. Change the line from first section to the second section, as shown in Example 9-6.

Example 9-6 Change line to second section

```
washost.LGIClaimRegistrationService service =
    new washost.LGIClaimRegistrationService();

to:

washost.LGIClaimRegistrationServiceWse service =
    new washost.LGIClaimRegistrationServiceWse();
```

3. Immediately after the modified line, add the code shown in Example 9-7.

Example 9-7 Add code

```
X509Certificate signCert = null;
X509CertificateStore signCertStore = null;
signCertStore = X509CertificateStore.LocalMachineStore
    (X509CertificateStore.MyStore);
signCertStore.OpenRead();
signCert =
    signCertStore.FindCertificateBySubjectString("SOAPRequester")[0];
X509SecurityToken secSignatureToken = new X509SecurityToken(signCert);
// sign message body
MessageSignature sig = new MessageSignature(secSignatureToken);
service.RequestSoapContext.Security.Elements.Add(sig);
service.RequestSoapContext.Security.Tokens.Add(secSignatureToken);
```

4. Rebuild the project. This will cause the updated project to be published to IIS.

The complete system with security can now be tested, as in 9.2.3, “Testing with the Web Services Explorer” on page 306.

9.6 Difference between the two Web Services

The method signatures in WebSphere and .NET that were used to generate Web Services are virtually identical. However, the Web Services that are generated by the two platforms have some significant differences. The main differences between the Microsoft .NET generated service definition file and the WebSphere one are:

- ▶ Exception handling
- ▶ Object array management
- ▶ Parameter multiplicity specification

For each difference, we provide a specific description in the following sections.

9.6.1 Exception handling

Business logic component methods belonging to both development environments throw a `ClaimException` to report some errors that could occur during the method execution.

No SOAP fault information is included in the Microsoft .NET WSDL file. The lack of detail is probably related to the fact that WSDL files generated in Microsoft .NET start from a C# class; since C# does not have an analog of the Java throws clause in method signatures, the method signature does not contain any information about exceptions thrown during the method execution.

In the WebSphere Studio generated WSDL file, a complex type is defined to map the `ClaimException` and the SOAP fault is associated with this type. Example 9-8 shows the exception handling in the WebSphere Studio generated WSDL file.

Example 9-8 Exception handling in the WebSphere Studio generated WSDL file

```
<wsdl:types>
.....
  <schema elementFormDefault="qualified".....>
    <complexType name="ClaimException">
      <sequence>
        <element name="message" nillable="true" type="xsd:string" />
      </sequence>
    </complexType>
    <element name="ClaimException" nillable="true" type="tns2:ClaimException" />
  </schema>
</wsdl:types>
....
<wsdl:message name="ClaimException">
  <wsdl:part element="tns2:ClaimException" name="fault" />
</wsdl:message>
```

```

</wsdl:message>
.....
<wsdl:portType name="LGIClaimRegistration">
  <wsdl:operation name="findCustomer">
    <wsdl:input message="intf:findCustomerRequest"
      name="findCustomerRequest" />
    <wsdl:output message="intf:findCustomerResponse"
      name="findCustomerResponse" />
    <wsdl:fault message="intf:ClaimException" name="ClaimException" />
  </wsdl:operation>
  <wsdl:operation name="registerClaim">
    <wsdl:input message="intf:registerClaimRequest"
      name="registerClaimRequest" />
    <wsdl:output message="intf:registerClaimResponse"
      name="registerClaimResponse" />
    <wsdl:fault message="intf:ClaimException" name="ClaimException" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="LGIClaimRegistrationSoapBinding"
  type="intf:LGIClaimRegistration">
  <wsdlsoap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="findCustomer">
    <wsdlsoap:operation SOAPACTION="" />
    <wsdl:input name="findCustomerRequest">
      <wsdlsoap:body use="literal" />
    </wsdl:input>
    <wsdl:output name="findCustomerResponse">
      <wsdlsoap:body use="literal" />
    </wsdl:output>
    <wsdl:fault name="ClaimException">
      <wsdlsoap:fault name="ClaimException" use="literal" />
    </wsdl:fault>
  </wsdl:operation>
  <wsdl:operation name="registerClaim">
    <wsdlsoap:operation SOAPACTION="" />
    <wsdl:input name="registerClaimRequest">
      <wsdlsoap:body use="literal" />
    </wsdl:input>
    <wsdl:output name="registerClaimResponse">
      <wsdlsoap:body use="literal" />
    </wsdl:output>
    <wsdl:fault name="ClaimException">
      <wsdlsoap:fault name="ClaimException" use="literal" />
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
.....

```

Starting with the WSDL file, the Web Service proxy wizard generates an `itso.examples.claim.exception` package, a `ClaimException` class and all other related classes needed for the SOAP serialization and deserialization of the `ClaimException` itself.

The `SoapBindingStub` class, which wraps the methods exposed by the Web Service, throws both a `java.rmi.RemoteException` and a `itso.examples.claim.exception.ClaimException`.

Part of the code implementing the generated method is shown in Example 9-9.

Example 9-9 Exception handling in the stub generated from a WebSphere WSDL file

```
try {
    .....
} catch (com.ibm.ws.webservices.engine.WebServicesFault wsf) {
    Exception e = wsf.getUserException();
    if (e != null) {
        if (e instanceof itso.examples.claim.exception.ClaimException) {
            throw (itso.examples.claim.exception.ClaimException) e;
        }
    }
    throw wsf;
}
```

If we compare the code listed in Example 9-9 with the corresponding code in Example 9-10 generated from the Microsoft .NET WSDL file, we can observe that in the second case, the only handled exception is the standard `WebServicesFault`. Both the `findCustomer` and `registerClaim` methods throw only a `java.rmi.RemoteException`.

Example 9-10 Exception handling in the stub generated from a Microsoft .NET WSDL file

```
try {
    .....
} catch (com.ibm.ws.webservices.engine.WebServicesFault wsf) {
    throw wsf;
}
```

The conclusion is that for bottom up development of a Microsoft .NET Web Service, application specific SOAP exceptions cannot be generated. Microsoft .NET Studio does not add any SOAP fault message in the WSDL files and all exceptions thrown by the Web Service are managed as simple SOAP server fault code. This different implementation of the exception management in the SOAP message, however, does not impact the interoperability between the two platforms.

To produce a detailed SOAP fault report from a Microsoft Web Service requires some coding. Some good advice is given in the MSDN® article *Using SOAP Faults*, found at:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnservice/html/service09172002.asp>

9.6.2 Object array management

In the registerClaim method, an array of strings is required as an input parameter. In both cases, the Web Services source code is developed using the basic String[] type, but if we compare the two generated WSDL files, we can find a different parameter declaration approach.

The WebSphere Studio generated WSDL file uses the basic type xsd:string with the maxOccurs property set to unbounded; the related part of the WSDL file is shown in Example 9-11.

Example 9-11 Object array type specification in WebSphere Studio generated WSDL file

```
...
  <element maxOccurs="unbounded" name="involvedCars" type="xsd:string"/>
....
```

The Microsoft Visual Studio .NET 2003 generated WSDL file instead uses the complex type ArrayOfString, as shown in Example 9-12.

Example 9-12 Object array type specification in Microsoft Visual Studio .NET 2003 generated WSDL file

```
...
<s:element name="registerClaim">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="customerID"
        type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="policyID"
        type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="accidentDate"
        type="s:dateTime" />
      <s:element minOccurs="0" maxOccurs="1" name="accidentDescription"
        type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="involvedCars"
        type="s:ArrayOfString" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:complexType name="ArrayOfString">
```

```

    <s:sequence>
      <s:element minOccurs="0" maxOccurs="unbounded" name="string"
        nillable="true" type="s:string" />
    </s:sequence>
  </s:complexType>
  ...

```

Starting from the Microsoft Visual Studio .NET 2003 generated WSDL file, the WebSphere Studio wizard generates an `ArrayOfString` class and all other related classes used to manage SOAP serialization and deserialization; this means that `ArrayOfString` is managed as a nonstandard object.

The difference in the SOAP request is shown in the following two examples, where Example 9-13 refers to the SOAP request to the WebSphere Web Service, while Example 9-14 refers to the SOAP request to the Microsoft .NET Web Service.

Example 9-13 SOAP request to the WebSphere Web Service

```

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <registerClaim xmlns="http://ejb.claim.examples.itso">
      <customerID>ABC123</customerID>
      <policyID>P00245</policyID>
      <accidentDate>2004-09-26T04:00:00.000Z</accidentDate>
      <accidentDescription>Car crash</accidentDescription>
      <involvedCars>NC-SH1</involvedCars>
      <involvedCars>SA-NUM2-00</involvedCars>
      <involvedCars>DH-CICS3</involvedCars>
    </registerClaim>
  </soapenv:Body>
</soapenv:Envelope>

```

Example 9-14 SOAP request to the Microsoft .NET Web Service

```

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <registerClaim xmlns="http://tempuri.org/">
      <customerID>AAAA</customerID>
      <policyID>BBBBB</policyID>
    </registerClaim>
  </soapenv:Body>
</soapenv:Envelope>

```



```
<accidentDate>2004-09-26T04:00:00.000Z</accidentDate>
<accidentDescription>CCCCC</accidentDescription>
<involvedCars>
  <string>SH1</string>
  <string>NUM2</string>
  <string>CICS3</string>
</involvedCars>
</registerClaim>
</soapenv:Body>
</soapenv:Envelope>
```

According to 8.4.4, “Web Services description constraints” on page 285, we found the following recommendation (see *R2112*).

- ▶ In a *description*, elements should not be named using the convention *ArrayOfXXX*.
- ▶ The correct way to define arrays is to define a basic type with *maxOccurs=unbounded*.

There is no specific unrespected **MUST** in the Microsoft .NET WSDL file, and the WebSphere Studio wizard is able to generate the correct client; the interoperability is then guaranteed between the two platforms.

9.6.3 Parameter multiplicity specification

In both platform generated WSDL files, the methods' input parameters are considered optional: *minOccurs* is set to 0 in the Microsoft .NET WSDL file, while *nillable* is set to *true* for the WebSphere WSDL file.

The difference in type declaration does not influence the proxy generation wizard in WebSphere Studio and Microsoft Visual Studio .NET 2003, and neither tool shows any problem in generating the Web Service proxy starting from a WSDL file generated with a different platform.

We also tried a manual update of the WSDL file, forcing the value of *minOccurs* to 1 and the *nillable* to *false*. The aim was for a client to be able to raise an exception before invoking a service if null values were set for mandatory inputs. However, even if we regenerated the proxy, we were not able to obtain such a behavior; the proxy generation is not influenced by these new values and we were able to invoke the service even passing null values for mandatory inputs and receiving an exception raised from the service. The lesson is that WSDL definitions should not be taken as a guaranteed precondition of how a service behaves. The author of a Web Service must check input arguments, even invalid values that are not allowed in the WSDL file.



Part 4

Appendices

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/SG246799>

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, SG246799.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
SG247027.zip	Sample application in a zipped archive

System requirements for downloading the Web material

The following system configuration is recommended:

- Hard disk space:** 40 GB Disk
- Operating System:** Windows XP Professional, Windows 2003 server
- Processor:** Minimum 1 GHz Intel Pentium® or equivalent
- Memory:** Minimum 1 GB RAM
- ▶ IBM Rational Application Developer for WebSphere Software v6.0
- ▶ WebSphere Application Server V6.0
- ▶ Windows XP Professional with upgrades or better, capable of running:
 - Microsoft Visual Studio .Net 2003
 - Microsoft .Net Framework 1.1
 - IIS 6.0
 - Web Services Enhancements 2.0

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.

Component interoperability

The source material consist of the following:

- ▶ Binaries
 - This folder contains the .NET client executable (DotNetClient.exe) and the ear file to be deployed in WebSphere Application Server (ComponentInterop.ear).
- ▶ Sources
 - This folder contains all the sources:
 - DotNetClient: Contains the .NET sources and the required project files for Visual Studio
 - ComponentInteropEJB: Contains the Java sources and the required project files for RSA
- ▶ requiredLibsAndSchema
 - This folder contains the following JARs, DLLs, and schema files.
 - The Calculator.jar.
 - The required JAXB jar files.
 - The MQ jars required by the EJB.
 - The MQ dlls requires by .NET code to communicate with MQ and the XML schema file.

- The CalculatorSchema.xsd schema file.

Instructions for installation and the details for the scenario can be found in Chapter 6, “Component interoperability scenario” on page 191.

Web Services interoperability

In addition to the source materials and .ear files needed for the samples, there are also sources and binaries for .NET, and certificates for WS-Security. Instructions for installation and the details for the scenario can be found in Chapter 9, “Web Services interoperability scenario” on page 293.

Abbreviations and acronyms

.NET	"dot NET" Microsoft Windows Web Services platform	GAC	Global Assembly Cache
ACL	Access Control List	GRE	Generic Routing Encapsulation
ADO	Active Data Objects	GUI	Graphical User Interface
ASP	Active Server Page	HTML	Hypertext Markup Language
BP EL	Business Process Execution Language	HTTP	HyperText Transfer Protocol
BP EL4WS	Business Process Execution Language for Web Service	HTTPS	Secure HTTP
BSF	Bean Scripting Framework	IBM	International Business Machines Corporation
CCW	COM Callable Wrapper	IDE	Integrated Development Environment
CLR	Common Language Runtime	IDL	Interface Definition Language
CLS	Common Language Specification	IETF	Internet Engineering Taskforce
CMP	Container Managed Persistence	IIOP	Internet Inter-ORB Protocol
COM	Component Object Model	IIS	Internet Information Server
COM+	Current version of the Microsoft Component Object Model	IIS	Internet Information Services
CORBA	Common Object Request Broker Architecture	IL	Intermediate Language
CTS	Common Type System	IP	Internet Protocol
DCOM	Distributed COM	ISO	International Standards Organization
DD	Deployment Descriptor	ITSO	International Technical Support Organization
DHTML	Dynamic HTML	J#	J-Sharp (Java on Windows)
DLL	Dynamic Link Library	J2EE	Java 2 Enterprise Edition
DNA	Distributed Internet Applications	J2SE	Java 2 Standard Edition
DTC	Distributed Transaction Coordinator	JAAS	Java Authentication and Authorization Service
EAR	Enterprise Archive	JAR	Java Archive
EIS	Enterprise Information System	JAXB	Java Architecture for XML Binding
EJB	Enterprise JavaBean	JAXP	Java for XML Parsing
		JAX-RPC	Java for XML Remote Procedure Call

JCA	Java Connector Architecture	RCW	Runtime-Callable Wrapper
JCP	Java Community Process	RFC	Request for Comment
JDBC	Java Database Connectivity	RMI	Remote Method Invocation
JIT	Just-In-Time (compiler)	RMI/IIOP	Remote Method Invocation over InterOperable Object Protocol
JMS	Java Message Service		
JMX™	Java Management Extensions	RPC	Remote Procedure Call
JNDI	Java Naming and Directory Interface	SCM	Service Control Manager
JNI	Java Native Interface	SOA	Service-Oriented Architecture
JRE	Java Runtime Environment	SOAP	Simple Object Access Protocol (now simply SOAP)
JScript	Java Script	SPML	Service Provisioning Markup Language
JSP	JavaServer Pages	SQL	Structured Query Language
JSR	Java Specification Request	SSL	Secure Sockets Layer
JTA	Java Transaction Architecture	SwA	SOAP with Attachments
JVM	Java Virtual Machine	SWAM	Simple WebSphere Authentication Mechanism
LTPA	Lightweight Third Party Authentication	TCP/IP	Transport Control Protocol/Internet Protocol
MDB	Message Driven Bean	TLS	Transport Layer Security
MIME	Multipurpose Internet Mail Extensions	UDDI	Universal Description. Discovery and Integration
MMC	Microsoft Management Console	UDP	User Datagram Protocol
MOM	Message-oriented Middleware	UML	Unified Modelling Language
MOM	Microsoft Operations Manager	URI	Universal Resource Identifier
MSDN	Microsoft Developer Network	URL	Universal Resource Locator
MSIL	Microsoft Intermediate Language	UTP-16	A universal 2 byte character encoding scheme
MTS	Microsoft Transaction Services	UTP-8	A universal mixed one and two byte character encoding scheme
ODBC	Open Database Connectivity	VBScript	Visual Basic Script
ORB	Object Request Broker	W3C	World Wide Web Consortium
OS	Operating System	WAR	Web Archive
PMI	Performance Monitoring Infrastructure	WLM	Workload Manager
RAR	Resource Adapter Archive	WMI	Windows Management Instrumentation

WS-	Web Service
WS-CAF	Web Service Composite Application Framework
WSDL	Web Services Definition Language
WSDL	Web Services Description Language
wsdl2java	Web Services to Java (converts WSDL to Java object)
WS-I	Web Services Interoperability organization
WSIF	Web Services Invocation Framework
WSIL	Web Services Inspection Language
WS-RM	Web Services Reliable Messaging
WSRP	Web Services for Remote Portals
WS-TXM	Web Services transaction Management
X.509	Standard for Public-Key Infrastructure
XACML	Extensible Access Control Markup Language
XDE	Extended Development Environment (IBM)
XMI	XML metadata interchange
XML	Extensible Markup Language
XOP	XML Binary Optimized Package
XSD	XML Schema Definition

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 390. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *EJB 2.0 Development with WebSphere Studio Application Developer*, SG24-6819
- ▶ *Rational Application Developer V6 Programming Guide*, SG24-6449
- ▶ *WebSphere and .NET Coexistence*, SG24-7027
- ▶ *WebSphere and .Net Interoperability Using Web Services*, SG24-6395
- ▶ *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688
- ▶ *WebSphere Application Server V6 Planning and Design WebSphere Handbook Series*, SG24-6446
- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392
- ▶ *WebSphere Application Server V6 Security Handbook*, SG24-6316
- ▶ *WebSphere MQ Security in an Enterprise Environment*, SG24-6814
- ▶ *WebSphere MQ Solutions in a Microsoft .NET Environment*, SG24-7012
- ▶ *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461

IBM Redpapers

- ▶ *WebSphere Application Server V6: System Management Problem Determination*, REDP-4067
- ▶ *WebSphere Application Server V6 Technical Overview*, REDP-3918

Other publications

These publications are also relevant as further information sources:

- ▶ *WebSphere MQ Using Java*, SC34-6066
- ▶ Richter, *Applied Microsoft .NET Framework Programming*, Microsoft Press, 2002, ISBN 0735614229

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ .NET information

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/gnconwebservicesdirectivesyntax.asp>
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconregisteringassemblieswithcom.asp>
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptools/html/cpgrfwebservicesdescriptionlanguagegettoolwsdl.exe.asp>

- ▶ .NET remoting information

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconnetremotingoverview.asp>

- ▶ .NET runtime

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vstchdeployingvsusingactivedirectory.asp

- ▶ .NET security

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh19.asp>
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh06.asp>

- ▶ Apache Ant project

<http://ant.apache.org/>

- ▶ Apache Jakarta project

<http://jakarta.apache.org/log4j>

- ▶ Apache WSIF project

<http://ws.apache.org/wsif/>

- ▶ ASP.NET Web applications

<http://support.microsoft.com/default.aspx?scid=kb;EN-US;q325056>

- ▶ Asynchronous Web Services Web site
<http://www-106.ibm.com/developerworks/webservices/library/ws-asynch1.html>
- ▶ Enterprise Patterns
<http://www.enterpriseintegrationpatterns.com/>
- ▶ GNU Licence information
<http://www.gnu.org/copyleft/lesser.html>
- ▶ IBM Redbooks Web site
<http://www.redbooks.ibm.com>
- ▶ IBM WebSphere Application Server Web site
<http://www-3.ibm.com/software/webservers/appserv/>
- ▶ IBM WebSphere Integrator Web site
<http://www-306.ibm.com/software/integration/wmq/>
- ▶ IBM WebSphere Performance and Scalability best practices
http://www.ibm.com/software/webservers/appserv/ws_bestpractices.pdf
- ▶ IIOP.NET Web site
<http://iiop-net.sourceforge.net/>
- ▶ Interface Tool for Java (formerly known as IBM Bridge2Java) Web site
<http://www.alphaworks.ibm.com/tech/bridge2java>
- ▶ J2EE RequestDispatcher information
<http://java.sun.com/j2ee/1.4/docs/api/javax/servlet/RequestDispatcher.html>
- ▶ Ja.NET Web site
<http://ja.net.intrinsyc.com/ja.net/info/>
- ▶ Janeva Web site
<http://www.borland.com/janeva/>
- ▶ Java Community Process
<http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html>
- ▶ Java RMI Web site
<http://java.sun.com/products/jdk/rmi/>
<http://java.sun.com/marketing/collateral/javarmi.html>
- ▶ JNBridge Web site
<http://www.jnbridge.com>

- ▶ Microsoft ASP.NET information
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconpage.asp>
- ▶ Microsoft .NET information
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconcontrolexecutionlifecycle.asp>
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconintroductiontontserviceapplications.asp>
- ▶ Microsoft Active Directory
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vstchdeployingvsusingactivedirectory.asp
- ▶ Microsoft clustering
<http://www.microsoft.com/windows2000/technologies/clustering/>
- ▶ Microsoft IIS information
<http://www.microsoft.com/windows2000/en/server/iis/default.asp>
- ▶ Microsoft Patterns
<http://www.microsoft.com/resources/practices/default.asp>
- ▶ Microsoft Server clustering
<http://www.microsoft.com/windows2000/technologies/clustering/default.asp>
- ▶ Microsoft SPNEGO details
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecure/html/http-sso-1.asp>
- ▶ Microsoft Systems Management Server
<http://www.microsoft.com/smsgmt/default.asp>
- ▶ Microsoft Tlbimp tool information
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptools/html/cpgrfTypeLibraryImporterTlbimpexe.asp>
- ▶ Microsoft Web site
<http://www.microsoft.com>

- ▶ Microsoft WMI information
<http://msdn.microsoft.com/library/techart/mngwmi.htm>
- ▶ OMG Web site
<http://www.omg.org>
- ▶ Patterns for e-business
<http://www.ibm.com/developerworks/patterns/>
- ▶ SOAP encoding performance
<http://www-106.ibm.com/developerworks/webservices/library/ws-soapenc/>
- ▶ Sun EJB information
<http://java.sun.com/products/ejb/>
- ▶ Sun JAAS information
<http://java.sun.com/products/jaas>
- ▶ Sun J2EE information
<http://java.sun.com/j2ee/>
- ▶ Sun JSP information
<http://java.sun.com/products/jsp/>
- ▶ UDDI Web site
<http://www.uddi.org>
- ▶ W3C DOM specification
<http://www.w3.org/TR/DOM-Level-3-Core/core.html#ID-1590626202>
- ▶ W3C SOAP specification
<http://www.w3.org/TR/SOAP/>
- ▶ Web Services Quality of Service
<http://www-106.ibm.com/developerworks/webservices/library/ws-soapenc/>
- ▶ Web Services security roadmap
<http://www.ibm.com/developerworks/webservices/library/ws-secmap/>
- ▶ WebSphere Application Server prerequisites
<http://www-3.ibm.com/software/webservers/appserv/doc/latest/prereq.html>
- ▶ WebSphere InfoCenter
<http://www-3.ibm.com/software/webservers/appserv/infocenter.html>
- ▶ WebSphere MQ MA7P Support pack
<http://www-3.ibm.com/software/integration/support/supportpacs/individual/ma7p.html>

- ▶ WebSphere prerequisites Web site
<http://www-3.ibm.com/software/webservers/appserv/doc/v50/prereqs/prereq502.html>
- ▶ WS-Addressing
<http://www.ibm.com/developerworks/webservices/library/ws-add/>
- ▶ WS-Coordination
<http://www.ibm.com/developerworks/webservices/library/ws-coor/>
- ▶ WS-I
<http://www.ws-i.org/>
- ▶ WS-Security
<http://www.ibm.com/developerworks/library/ws-secure/>
- ▶ WS-Transaction
<http://www.ibm.com/developerworks/webservices/library/ws-transpec/>

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

- (SOAP) Simple Object Access Protocol 70
- .aspx file 117
- .NET COM interop 184
- .NET components 156
- .NET configuration 138
- .NET configuration files 109
- .NET debug configuration 128
- .NET enterprise servers 119
- .NET enterprise services 97
- .NET environment 96
- .NET evolution 96
- .NET framework 101
- .NET initiative 92
- .NET key values 96
- .NET languages 101
- .NET platform 91
- .NET redistributable 101
- .NET Remoting 100
- .NET Service
 - WebSphere findCustomer 338
- .NET Software Development Kit 140
- .NET suite 100
- .NET Web Service 326
 - client
 - sign requests 366
 - encrypting messages 342
 - Web Service Extensions 345
- .NET Web Service proxy 329
- .NET Windows Form client application 222

A

- Abstract Window Toolkit (AWT) 151
- access control 48
- Access Data Object (ADO) 93
- ACT (Application Center Test) 132
- Active Directory 138
- Active Server Page.NET (ASP.NET) 99
- Active Server Pages (ASP) 93
- ActiveX
 - control 152
- ActiveX Data Object (ADO) 96
- Additional material 377

- administrative tools 136
- ADO (Access Data Object) 93
- ADO (ActiveX Data Object) 96
- ADO.NET 139
- AJAX (Asynchronous JavaScript and XML) 13
- Appendices 375
- application
 - assembler 34
 - behavior 80
 - cells 65
 - client module 44
 - code 81
 - component provider 34
 - configuration file 109
 - connectivity 57
 - developer 54
 - development 76
 - logging 144
 - nodes 65
 - packaging 43
 - servers 65
- application architecture model 6
 - business logic tier 9
 - client tier 7
 - data tier 9
 - presentation tier 8
- Application Center Test (ACT) 132
- Application considerations 166
- application interoperability 3
 - drivers 4
 - models 6
 - stack 27
 - application layer 28
 - control layer 29
 - data layer 29
 - transport layer 29
- Application interoperability models 6
- Application interoperability stack 27
- approaches
 - bridging 181
 - cross compilation 181
 - messaging and queuing 181
 - porting 181
 - remote procedure calls 181

- Approaches for interoperability 25
- architecture
 - model 73
 - client server n-tier 6
- Architectural overview of stand-alone WebSphere
 - application server 63
- ASP (Active Server Pages) 95
- ASP.NET
 - authentication 143
 - component trace 145
- ASP.NET (Active Server Page.NET) 99
- Assemblies 107
- assembly 107
 - metadata 108
 - resources 108
 - versioning 110
- asynchronous
 - interaction 169
 - message queuing 197
 - processing 174
 - service 234
- Asynchronous JavaScript and XML (AJAX) 13
- authentication
 - mechanisms 52
 - services 143
- AWT (Abstract Window Toolkit) 151

B

- B2B (business-to-business) 37
- basic interaction outline
 - calculator component 192
 - client component 192
- bean managed persistence (BMP) 155
- Best of Breed (BoB) 5
- binding
 - SOAP 278
- BMP (bean managed persistence) 155
- BoB (Best of Breed) 5
- BPEL4WS (Business Process Execution Language for Web Services) 260
- Building the claims .NET Web Service 313
- Building the claims Websphere Web Service 295
- business
 - integration 57
 - layer 99
 - layer components 157
 - logic 154
 - logic tier 39

- objects 100
- portals 59
- Business Process Execution Language for Web Services (BPEL4WS) 260
- business-to-business (B2B) 37

C

- calculator service 194
- Calculator service class diagram 195
- callbacks 179
- CLI (Common Language Infrastructure) 102
- client
 - activation 158
 - application 324
 - applications 32
 - components 41
 - console 152
 - container 46
 - side components 150
- client server application
 - business logic 6
 - data tiers 6
 - the presentation 6
- client tier
 - interoperability
 - client to business 12
 - client to client 10
 - client to data 16
 - client to presentation 11
 - rich application clients 7
 - smart clients 7
 - thin application clients 7
- Client tier to presentation tier interoperability 11
- Client to client tier interoperability 10
- CLR (Common Language Runtime) 96
- CLS (Common Language Specification) 101
- CMP (container managed persistence) 155
- code animation 129
- COM (Component Object Model) 93
- COM service proxy 184
- COM+ Services 94
- Common Language Infrastructure (CLI) 102
- Common Language Runtime (CLR) 101
- Common Language Specification (CLS) 103
- Common Object Request Broker Architecture (CORBA) 54
- Common Type System (CTS) 107
- communication technologies 53

- compatibility testing 50
- Component interoperability 147, 247
- component interoperability 147, 149
 - designing 165
- Component interoperability scenario 191
- Component Object Model (COM) 93
- component technologies 52
- Components overview 150
- Configure the Development Environment 295
- container managed persistence (CMP) 155
- Control considerations 179
- CORBA (Common Object Request Broker Architecture) 54
- Creating messaging resources, schema and classes 211
- CTS (Common Type System) 107

D

- data
 - considerations 176
 - encryption 143
 - formats 54
 - layer 100
 - modeler 74
- Data considerations 176
- database connection pooling 139
- DB2 70
 - stored procedures 78
 - UDB 77
 - XML extender 78
- DCI (DirectCarInsure.com) 294
- debug
 - builds 128
 - settings 128
- debugging instrumentation 130
- deployer 34
- deployment 134
 - copying files 134
 - failover 144
 - project types 135
 - setup 134
- deserialization 279
- design patterns 81
- Develop the interoperability adapter 222
- Developing .NET applications 121
- Developing the .NET Windows Form client 222
- Developing the WebSphere service 234
- development

- platforms 66
- DirectCarInsure.com (DCI) 294
- distributed server configuration 63
- Distributed Transaction Coordinator (DTC) 141
- Document Type Definitions (DTD) 79
- document/encoded 280
- document/literal 280
 - wrapped 281
- Drivers for interoperability 4
- DTC (Distributed Transaction Coordinator) 141
- DTD (Document Type Definitions) 79
- DTD editor 79

E

- EAR (Enterprise Archive) 43
- e-business 57
- Eclipse Modeling Framework (EMF) 68
- edge servers 57
- EGL (Enterprise Generation Language) 79
- EIS (Enterprise Information Systems) 24, 53
- EJB
 - container 40
 - module 44
- EJB (Enterprise JavaBeans) 9
- Elements of interoperability 27
- Elements of Web Services interoperability 270
- EMF 68
- EMF (Eclipse Modeling Framework) 68
- encoding
 - SOAP 279
- Encrypting messages to .NET Web Service 342
- Enterprise Archive (EAR) 43
- Enterprise Generation Language (EGL) 79
- Enterprise Information Systems (EIS) 24
- Enterprise JavaBeans (EJB) 9, 32
- entity beans 155
- exception handling 368
- Exception handling in the stub generated from a Microsoft .NET WSDL file 370
- execution support 106
- extensibility 49
- Extensible Markup Language (XML) 79

F

- FCL (Framework Class Library) 112
- Framework Class Library (FCL) 101

G

GAC (Global Assembly Cache) 109
garbage collection 48
garbage collector 104
Generate the .NET classes corresponding to the XML schema 221
Generate the Java classes corresponding to the XML schema 221
Global Assembly Cache (GAC) 111
graphical user interface (GUI) 83
GUI (graphical user interface) 76

H

HTTP (Hypertext Transfer Protocol) 53
Hypertext Transfer Protocol (HTTP) 53

I

IBM

MultiSite 85
Rational
 developer network 89
 functional tester 88
 manual tester 83
 performance tester 83
 portfolio manager 87
 ProjectConsole 87
 PurifyPlus 88
 RequisitePro 73
 robot 88
 SoDA 87
 software architect 80
 software modeler 73
 suite 88
 TestManager 82
Rational Application Developer
 WebSphere Software 76
Rational ClearCase 85
 Change Management Solution 85
 Enterprise Edition 85
 LT 85
 MultiSite 85
Rational Rose
 data modeler 73
 technical developer 81
 XDE Developer Plus 88
Rational Team
 unifying platform 87
Rational Unified Process (RUP) 88

Rational Web Developer
 WebSphere Software 75
RUP (Rational Unified Process) 88
SDP (Software Development Platform) 72
Software Development Platform 68
 construction 72
 design 72
 Design and construction 74
 portfolio management 72
 process 86
 Requirements and analysis 72
 requirements and analysis 72
 Software configuration management 85
 software configuration management 72
 Software quality 82
 software quality 72
Software Development Platform (SDP)
 products 72
WebSphere 61
 Business Integration Server Foundation 59
 MQ Workflow 58
 Studio Asset Analyzer 81
 Studio Enterprise Developer 81
IBM MDD platform 70
IBM Software Development Platform 66
IDE (Integrated Development Environment) 68
IDL (Java Interface Definition Language) 54
IIOP (Internet Inter-ORB Protocol) 54
IIOP.NET 186
IIS (Internet Information Server) 93
IIS 6.0 378
information technology (IT) 4
Installing and configuring the queues 203
Integrated Development Environment (IDE) 68
interaction dynamics 166
interaction management 166
intermediate language 105
Internet Information Server (IIS) 95
Internet Inter-ORB Protocol (IIOP) 54
Internet Protocol (IP) 53
Internet Services Manager 137
interoperability
 approaches 25
 class level 25
 service-oriented 25
 best of breed integration 5
 client tier
 client to client 10
 elements 27

- activities 29
- constraints 29
- extended enterprise 5
- scenarios 9
 - business tier 21
 - client tier 9
 - data tier 24
 - presentation tier 17
- technology transition
 - emerging technologies 4
 - mergers/acquisitions 4
 - platform migration 4
- Interoperability architecture elements 27
- Introduction 1
- Introduction to component interoperability 159
- Introduction to the claims scenario 294
- Introduction to Web Services 250
- Introduction to Web Services Interoperability 249
- IP (Internet Protocol) 53
- IT (information technology) 4
- ItsoClaim Project
 - add Web reference 338

J

J2EE

- application environment 42
- architecture
 - technology support 52
- component model 38
- data resources tier 37
- n-tier architecture 35
- presentation tier 36
- security 48
 - auditing 49
 - authentication 50
 - data integrity 49
 - data privacy 49
 - flexibility 50
- specification 32
- J2EE (Java 2 Enterprise Edition) 31
- J2EE Connector Architecture (JCA) 24
- J2SE (Java 2 Standard Edition platform) 39
- JAF (JavaBeans Activation Framework) 54
- Janeva 187
- Java
 - classes 80
 - programming language 43
 - runtime environment 42

- Servlet API 153
- Java 2 Enterprise Edition (J2EE) 324
- Java 2 Standard Edition platform (J2SE) 39
- Java Connector Architecture (JCA) 45
- Java Database Connectivity (JDBC) 52
- Java Emitter Templates (JET) 68
- Java Interface Definition Language (IDL) 54
- Java Message Service (JMS) 54
- Java Naming and Directory Interface (JNDI) 53
- Java Server Pages (JSP) 154
- Java Server Pages Standard Tag Library (JSTL) 154
- Java Transaction API (JTA) 52
- Java Virtual Machine (JVM) 42
- Java Web Services Developer Pack (JWS DP) 221
- JavaBeans 78
- JavaBeans Activation Framework (JAF) 54
- JavaMail 54
- JavaServer
 - faces 71
 - pages 75
- JCA (J2EE Connector Architecture) 24
- JCA (Java Connector Architecture) 45
- JDBC (Java Database Connectivity) 52
- JET (Java Emitter Templates) 68
- J-Integra 187
- JIT (Just-In-Time) 129
- JMS
 - TextMessage 241
- JMS (Java Message Service) 54
- JNBridge 187
- JNDI (Java Naming and Directory Interface) 53
- JSP (Java Server Pages) 154
- JSTL (Java Server Pages Standard Tag Library) 154
- JTA (Java Transaction API) 52
- Just-In-Time (JIT) 105
- JVM (Java Virtual Machine) 42
- JWS DP (Java Web Services Developer Pack) 221

L

- LGI (Lord General Insurance) 294
- life cycle management 47
- Lightweight Third Party Authentication (LTPA) 52
- listener port 218
- literal
 - encoding 279
 - XML 280

- load
 - balancing 144
 - testing 131
- Lord General Insurance (LGI) 294
- LTPA (Lightweight Third Party Authentication) 52

M

- machine configuration file 109
- managed code 99
- marshalling 279
- MDAC (Microsoft Data Access Components) 96
- MDD (Model-Driven Development) 69
- mergers and acquisitions scenario
 - ClaimException 369
 - findCustomer 369
 - registerClaim 369
- message
 - format 193
 - queuing 95
 - routing 174
 - transformation 174
- message driven bean 234
- Message Transmission Optimization Mechanism 282
- messaging
 - authentication 143
 - classes 211
 - middleware 170
 - resources 211
 - schema 211
 - technologies 54
- Microsoft
 - operations manager 137
 - patterns 97
 - Visual Studio .Net 2003 321, 378
- Microsoft Data Access Components (MDAC) 96
- Microsoft intermediate language (MSIL) 108
- Microsoft Message Queue (MSMQ) 93
- Microsoft Transaction Server (MTS) 93
- Model View Controller (MVC) 35
- Model-Driven Development (MDD) 70
- Model-View-Controller 40
- MSIL (Microsoft intermediate language) 108
- MSMQ (Microsoft Message Queuing) 141
- MTOM 282
- MTS (Microsoft Transaction Server) 93
- MVC (Model View Controller) 35

N

- Network Load Balancing 144
- Network News Transfer Protocol (NNTP) 95
- n-layer model 98
- NNTP (Network News Transfer Protocol) 95
- non-distributed transactions 141
- non-repudiation 49

O

- object
 - array management 371
 - pooling 48
- Object Management Group (OMG) 54
- OMG (Object Management Group) 54
- onMessage() method 234
- operation
 - overloading 281
- Overview 1
- Overview of the Microsoft architecture 97
- Overview of Web Services Interoperability 262

P

- parameter multiplicity specification 373
- performance 139
 - counter 131
 - monitor 131
- portable executable (PE) 107
- presentation layer 99
 - components 156
- private assembly 109
- process integration 58
- product
 - provider 34

Q

- queues
 - configuration 203
 - installation 203

R

- RAD (Rapid Action Development) 124
- Rapid Action Development (RAD) 124
- RAR (Resource Adapter Archive) 45
- Registration request data error message 341
- Remote Method Invocation (RMI) 53
- remoting 139
 - activation 158

- authentication 143
- channel 157
- formatter 157
- transport 157
- resource
 - adapter module 45
- Resource Adapter Archive (RAR) 45
- resource layer 100
- RMI (Remote Method Invocation) 53
- role-based
 - security 142
 - solutions portfolio 71
- RPC
 - style 278
- RPC/encoded 280
- RPC/literal 280
- Running .NET applications 133
- Running the .NET application 246

S

- SAAJ 282
- Sample type definition 274
- Scenario Description 192
- SCM (software configuration management) 85
- SDO (Service Data Object) 70
- Secure Socket Layer (SSL) 53
- security 142
 - configuration file 110
- serialization 279
- server
 - activation 158
 - clustering 144
- service
 - façade 170
 - technologies 52
- Service Data Object (SDO) 70
- served components 141
- session
 - beans 155
 - EJB 40
- Setting up the MQServer Environment Variable on the .NET machine 210
- Shared assembly 109
- Simple Mail Transfer Protocol (SMTP) 95
- Simple Object Access Protocol (SOAP) 70
- Simple WebSphere Authentication Mechanism (SWAM) 52
- SimpleDataGateway 117

- SMTP (Simple Mail Transfer Protocol) 95
- SOAP
 - binding 278
 - body 369
 - encoding 279
 - fault 368
- SOAP Encoding 279
- SOAP Messages with Attachments 282
- SOAP with Attachments API for Java 282
- software configuration management (SCM) 85
- source code management 127
- Spy++ 130
- SSL (Secure Socket Layer) 53
- stand-alone server configuration 62
- state
 - limitations 172
 - management 172
 - object 168
- state management 166
- state-dependent operations 173
- stateful
 - asynchronous interaction 172
 - interaction 166
 - messages 173
 - synchronous interaction 166
- stateless
 - asynchronous interaction 170
 - interaction 167
 - synchronous interaction 168
- Stateless synchronous interaction 167
- Structure of Common Language Runtime (CLR) 103
- stub 53
- style 280
- SWAM (Simple WebSphere Authentication Mechanism) 52
- swing 151
- system
 - administrator 34
 - prerequisites 201

T

- tamper-proof assemblies 142
- TCP (Transport Control Protocol) 53
- Testing the sample application 244
- Testing with the Web Services Explorer 306
- Text form of CalculatorSchema 220
- The .NET initiative 92

- The .NET suite 100
- The client application 324
- The WebSphere platform 54
- tracing instrumentation 130
- transaction management 141
- transparency 49
- Transport considerations 187
- Transport Control Protocol (TCP) 53
- TroubleShooting 246
- type
 - definition 274
 - mapping 177
 - metadata 108
 - mismatch 275

U

- unit testing 127
- unmanaged code 99
- unmarshalling 279
- user registry 52
- utility
 - JAR 45

V

- VES (Virtual Execution System) 103
- Virtual Execution System (VES) 103
- Visual Source Safe (VSS) 127
- Visual Studio .NET 124
 - debugger 128
- VSS (Visual Source Safe) 127

W

- WAR (Web archive) 44
- WAS ND (WebSphere Application Server Network Deployment) 61
- Web
 - applications 32
 - components 39
 - container 46
 - module 44
 - references 152
- Web archive (WAR) 44
- Web components
 - filters 39
- Web Services
 - architecture model 261
 - authentication 143

- background 250
- differences 368
- interoperability 247
 - designing 269
- model 251
- runtime 283
- security 342
- Web Services activities
 - description activities 270
 - invocation activities 271
- Web Services categories 253
- Web Services Choreography Description Language (WS-CDL) 260
- Web Services constraints 281
- Web Services description 273
- Web Services Description Language (WSDL) 14
- Web Services Enhancements 2.0 378
- Web Services Interoperability 249
- Web Services interoperability 247, 375
- Web Services interoperability scenario 293
- Web Services invocation 278
- WebSphere 54
 - application server 57, 309
 - JMS resources 211
 - network deployment 57
 - business integration
 - adapters 57
 - connect 58
 - event broker 58
 - express 58
 - message broker 57
 - modeler 59, 73
 - monitor 58, 73
 - server 58
 - Server Express Plus 59
 - server foundation 59
 - workbench server 58
 - calculator component 234
 - client
 - encrypted requests 348
 - commerce 60
 - business edition 60
 - professional edition 61
 - data interchange 57
 - extended deployment 57
 - host Integration server 57
 - integrator 174
 - InterChange Server 58
 - platform 31

- product center 59
- programming model 70
- Web service support 283
- WebSphere and .NET platforms 147
- WebSphere application server distributed server environment. 64
- WebSphere Application Server Network Deployment (WAS ND) 61
- WebSphere MQ 58
 - messaging 196
 - queue destinations 214
 - server
 - configure 203
 - install 203
 - workflow 58
- WebSphere Studio Application Developer 378
- WebSphere Web Service
 - signed messages 354
- WebSphere/J2EE for .NET developers 32
- Why choose component interoperability 162
- Windows
 - applications 126
 - DNA 92
 - event log 145
 - Form .NET client 230
 - project types 126
- Windows 2000
 - native services 95
- Windows Management Interface (WMI) 145
- WMI (Windows Management Interface) 145
- Writing C# 121
- WS-CDL (Web Services Choreography Description Language) 260
- WSDL
 - conversion 281
- WSDL (Web Services Definition Language) 70
- WSDL (Web Services Description Language) 14
- WS-I
 - validation tools 284
- WS-manageability 261
- WS-Security
 - interoperability 291

X

- XKMS (XML Key Management Specification) 259
- XMI 280
- XML
 - editor 79
 - metadata interchange
 - see XMI
 - schema
 - .NET classes 221
 - editor 79
 - Java classes 221
 - messages 219
 - XML (Extensible Markup Language) 79
 - XML Key Management Specification (XKMS) 259
 - XSD 368
 - XSL editor 79



IBM WebSphere and Microsoft .NET Interoperability

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



IBM WebSphere and Microsoft .NET Interoperability



Application interoperability overview

This IBM Redbook explores interoperability design between applications running on IBM WebSphere and Microsoft's .NET platforms at the component level and using Web Services technologies. It is a good source of information for IT architects, IT specialists, application integrators and developers who have to design and develop interoperability solutions.

Component and Web Services interoperability

Part 1, "Introduction", provides an overview of application interoperability starting with the business and technology drivers and introduces the concept of the Application Interoperability Stack, which defines a structured approach for application interoperability considerations and design.

Inteoperability scenarios

Part 2, "Component interoperability", provides an overview for component level interoperability and considerations to help with component interoperability solution design. A sample scenario shows interoperability between the WebSphere Java service component and .NET Windows Forms application.

Part 3, "Web Services interoperability", introduces application interoperability implementation using Web Services technologies. It includes considerations to aid solution design and a sample scenario implementation showing WebSphere to .NET interoperability using Web Services technologies.

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