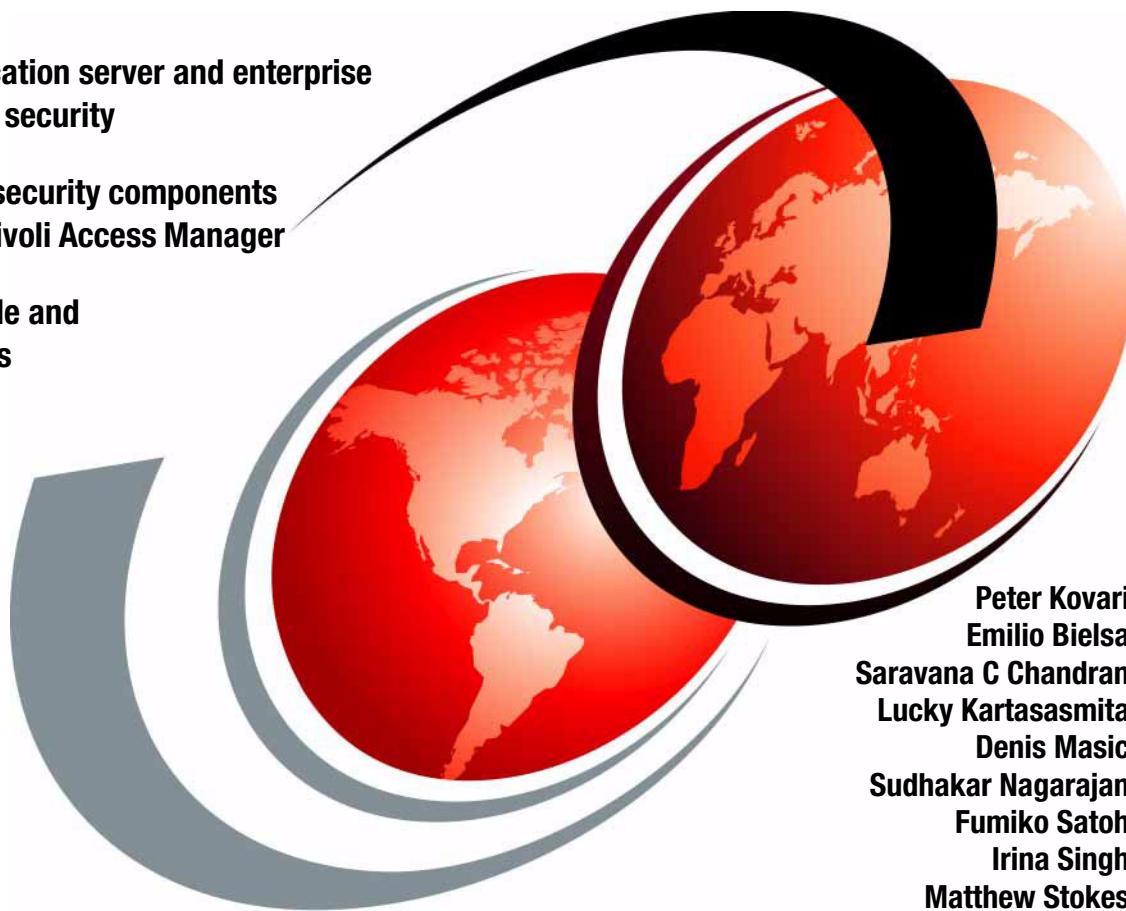


WebSphere Application Server V6 Security Handbook

J2EE application server and enterprise
application security

Additional security components
including Tivoli Access Manager

Sample code and
applications



Peter Kovari
Emilio Bielsa
Saravana C Chandran
Lucky Kartasasmita
Denis Masic
Sudhakar Nagarajan
Fumiko Satoh
Irina Singh
Matthew Stokes

Redbooks



International Technical Support Organization

**WebSphere Application Server V6
Security Handbook**

June 2005

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

First Edition (June 2005)

This edition applies to WebSphere Application Server V6 (base) on IBM AIX V5.2, RedHat Enterprise Linux V3, Windows 2000; WebSphere Application Server Network Deployment V6 on IBM AIX V5.2, RedHat Enterprise Linux V3, Windows 2000; Tivoli Access Manager V5.1 on IBM AIX V5.2, RedHat Enterprise Linux V3, Windows 2000.

© Copyright International Business Machines Corporation 2005. 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

Noticesxi
Trademarks	xii
Preface	xiii
The team that wrote this redbook	xiv
Become a published author	xvi
Comments welcome	xvi
Part 1. Application server security	1
Chapter 1. Introduction	3
1.1 Focus on security	4
1.2 No introduction to security	4
1.3 Scenario-based chapters	4
1.4 Sample applications	5
1.5 WebSphere Information Center	6
Chapter 2. Configuring the user registry	7
2.1 User registries	8
2.2 LDAP user registry	9
2.2.1 Basic LDAP configuration for WebSphere Application Server	10
2.2.2 Advanced LDAP configuration	13
2.2.3 Configuring LDAP access over SSL	14
2.3 Local OS User Registry	17
2.4 Custom User Registry	20
2.4.1 The UserRegistry interface	20
2.4.2 File-based Custom User Registry sample	23
2.4.3 DB2 Custom User Registry Sample	26
Chapter 3. Global Security	31
3.1 Introduction	32
3.2 Enabling Global Security	32
3.2.1 Main configuration parameters for Global Security	33
3.2.2 Other properties in the Global Security page	37
3.2.3 Stopping the application server	38
3.3 Disabling Global Security	39
3.4 Administrative roles	40
3.5 Naming service security: CosNaming roles	43
3.6 SSL configurations	45

3.6.1 Creating a new SSL entry	46
Chapter 4. JAAS for authentication and authorization in WebSphere	49
4.1 Introduction	50
4.1.1 JAAS in WebSphere	50
4.2 Custom JAAS login in WebSphere	50
4.2.1 Callback handler	50
4.2.2 Login module	51
4.2.3 Principal	55
4.2.4 Configuration	56
4.2.5 Programming authentication	58
4.2.6 Programming authorization	58
4.3 J2C Authentication data	58
Chapter 5. Enterprise application security	61
5.1 Deploying a secured application	62
5.1.1 Role mapping during application installation	62
5.1.2 Role mapping after installation	62
Chapter 6. Securing a Web application	65
6.1 Introduction	66
6.2 Securing the static content	66
6.2.1 Securing transport channel between Web browser and Web server	67
6.2.2 Authentication with Web server	69
6.2.3 Authorization with the Web server	72
6.3 Securing the Web server plugin for WebSphere	73
6.3.1 Securing the transport channel between the Web server and WebSphere	73
6.4 Securing the application server Web container	79
6.4.1 Securing the transport channel	79
6.4.2 Authentication with the Web container	80
6.4.3 Authorization with Web container	83
6.4.4 Programmatic security	92
6.5 Options	97
6.5.1 Configuring te HTTP Digest Authentication method	97
6.5.2 Configuring LDAP authentication with IBM HTTP Server	98
6.5.3 Configuring SSL certificate based client authentication method for IBM HTTP Server	103
6.5.4 Configuring SSL certificate based client authentication method for WebSphere Application Server	107
Chapter 7. Securing an EJB application	119
7.1 Introduction	120
7.2 Programmatic login (server-side) using JAAS	122

7.3 Declarative J2EE security	122
7.3.1 Defining J2EE security roles for EJB modules	122
7.3.2 Security role references	123
7.3.3 Configuring method access control	128
7.3.4 Enterprise Java Bean Run-As delegation policy	133
7.3.5 Bean level delegation	133
7.3.6 Method level delegation	136
7.3.7 Run-as mapping	140
7.4 Programmatic J2EE security	144
7.4.1 EJB security methods	144
7.5 EJB container access security	145
7.5.1 CSIV2 and SAS	145
7.5.2 Container authentication	147
7.5.3 RMI/IOP transport channel protection	150
Chapter 8. Client security	153
8.1 Application clients in WebSphere	154
8.2 Java client authentication protocol	156
8.3 Java client configuration	160
8.4 J2EE application client	164
8.5 Thin application client	167
8.6 Programmatic login	171
8.6.1 JAAS login module in WebSphere	171
8.6.2 Login process, programmatically	173
8.6.3 Client-side programmatic login using JAAS	174
8.7 Securing the connection	179
8.7.1 IOP over SSL, a thin client example	180
Part 2. Extending security beyond the application server	185
Chapter 9. Security attribute propagation	187
9.1 Introduction	188
9.2 Initial Login versus Propagation Login	189
9.3 Token framework	190
9.4 Custom Implementation of Tokens	192
9.4.1 Authorization token	193
9.4.2 Single Sign-On (SSO) token	195
9.4.3 Propagation token	196
9.4.4 Authentication token	199
9.4.5 Changing the Token Factory associated with the default token	199
9.5 Horizontal Propagation	201
9.6 Downstream propagation	204
9.7 Enabling security attribute propagation	206
9.7.1 Security attribute propagation for Horizontal Propagation	207

9.7.2 Enabling Downstream Propagation.....	208
9.8 Advantages of security attribute propagation	209
Chapter 10. Securing a WebSphere application using Tivoli Access Manager.....	211
10.1 Tivoli Access Manager introduction.....	212
10.1.1 Benefits	212
10.1.2 Access Manager Secure Domain	213
10.1.3 Access Manager and WebSphere integration.....	216
10.2 IBM Tivoli Access Manager security model.....	218
10.2.1 User registry	219
10.2.2 Master authorization (policy) database	219
10.3 Lab environment	222
10.4 Role of Tivoli Access Manager in WebSphere V6.....	223
10.4.1 Embedded Tivoli Access Manager Client architecture	224
10.5 WebSEAL authentication	229
10.5.1 Basic authentication	229
10.5.2 Form-based authentication	230
10.5.3 Client certificate based authentication	231
10.5.4 Token authentication.....	233
10.5.5 HTTP header authentication	233
10.5.6 Kerberos and SPNEGO authentication.....	233
10.6 WebSEAL junctions.....	234
10.6.1 Simple junctions	235
10.6.2 Single Sign-On junctions.....	238
10.7 Trust Association Interceptor (TAI)	242
10.7.1 New, enhanced TAI Interface	243
10.8 IBM WebSphere Application Server and WebSEAL integration	244
10.8.1 Integration options	244
10.8.2 Configuration for the Trust Association Interceptor approach	246
10.8.3 Configuration for the LTPA approach	255
10.9 Integration of IBM WebSphere Application Server and Tivoli Access Manager	261
10.9.1 aznAPI	262
10.9.2 Tivoli Access Manager and J2EE Security	262
10.9.3 Embedded Tivoli Access Manager in WebSphere Application Server V6	264
Chapter 11. Externalizing authorization with JACC	269
11.1 The specification	270
11.2 Deployment tools contract.....	271
11.3 Container contract	273
11.4 Provider contract	274

11.5 Why JACC?	274
11.6 JACC in WebSphere V6	275
11.6.1 JACC access decisions in WebSphere V6	276
11.6.2 JACC policy context identifiers in WebSphere V6	279
11.6.3 WebSphere Extensions to the JACC Specification	280
11.6.4 JACC policy propagation in WebSphere V6	280
11.6.5 Dynamic module updates in WebSphere V6 for JACC	284
11.7 Integrating Tivoli Access Manager as an external JACC provider	284
11.7.1 Disabling the embedded Tivoli Access Manager	286
11.8 Sample application for JACC	287
Chapter 12. Web services security	289
12.1 Security overview	290
12.2 Web service security exposures	291
12.3 WS-Security	294
12.3.1 WS-Security concepts	294
12.3.2 Evolution of the WS-Security specification	295
12.3.3 WS-Security Roadmap	297
12.3.4 Example of WS-Security	299
12.4 Transport-level security	304
12.4.1 SOAP/HTTP transport-level security	304
12.5 Summary	305
Chapter 13. Securing messaging oriented middleware	307
13.1 Java Messaging Service API	308
13.2 Default JMS provider	310
13.2.1 Messaging components of the Service Integration Bus	310
13.2.2 Embedded messaging security overview	313
13.2.3 Administering Service Integration Bus security	317
13.2.4 Administering destination security	318
13.2.5 Administering topic space roles	320
13.3 Application server and WebSphere MQ	321
13.3.1 WebSphere MQ messaging components	321
13.3.2 Authentication	323
13.3.3 Authorization	324
13.3.4 Transport security	325
13.3.5 Administering foreign Service Integration Bus security	326
13.3.6 Administering WebSphere MQ security	328
13.3.7 Connecting WebSphere MQ to the Service Integration Bus	329
13.4 Sample application	333
Chapter 14. Java 2 Connector security	335
14.1 What is the J2EE Connector Architecture?	336
14.2 Securing the J2EE Connector	337

14.3 Additional information	341
Chapter 15. Securing the database connection	343
15.1 Database security	344
15.1.1 Securing the connection	344
15.1.2 Securing access to database data	347
15.2 Defining a JDBC data source	348
15.2.1 Defining a JDBC data source using the Administrative Console ..	349
15.2.2 Defining a JDBC data source in EAR	352
15.3 JDBC connection to DB2 from WebSphere Web applications	355
15.3.1 Defining the Resource Reference	355
15.3.2 Obtaining the JDBC Connection	358
15.4 EJB persistence in WebSphere using DB2	360
15.4.1 Bean Managed Persistence Entity Beans	361
15.4.2 Container Managed Persistence Entity beans	362
15.5 Sample application	364
Part 3. Development environment	365
Chapter 16. Development environment security	367
16.1 Rational Application Developer	368
16.1.1 Securing the workspace	368
16.2 WebSphere Test Environment	370
16.2.1 Creating a new test server	371
16.2.2 Enabling security for the WebSphere Test Server V6	374
16.3 Administering and configuring the WebSphere test servers	376
16.4 Enterprise application security	377
16.4.1 Configuring enterprise application security during the development phase	377
16.4.2 JAAS authentication entries in the deployment descriptor	379
16.5 Creating a new profile for the WebSphere Test Server	381
Part 4. Appendixes	385
Appendix A. Additional configurations	387
Sample application for client security	388
Sample application for testing JACC	391
Testing	392
Configuring Embedded Messaging	393
Sample application for messaging	402
Configure the application server	403
Configure MQ	407
Install the Application	408
Test the application	409

Sample application for database connections	411
Creating the sample database	412
Defining the J2C Authentication Alias.....	414
Defining the JDBC data source.....	416
Importing and starting the sample application in Rational Application Developer	419
Install the sample application into WebSphere Application Server.....	419
Testing the sample application	421
Appendix B. Additional material	423
Locating the Web material	423
Using the Web material	423
System requirements for downloading the Web material	424
How to use the Web material	424
Abbreviations and acronyms	425
Related publications	427
IBM Redbooks	427
Online resources	427
How to get IBM Redbooks	428
Help from IBM	428
Index	429

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

@server®	Domino®	Rational®
©server®	DB2 Universal Database™	Redbooks™
Redbooks (logo)  ™	DB2®	RACF®
developerWorks®	DRDA®	SLC™
ibm.com®	ETE™	Tivoli®
AIX®	IBM®	WebSphere®
Distributed Relational Database Architecture™	Lotus®	
	Notes®	

The following terms are trademarks of other companies:

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

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

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

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

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

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

Preface

This IBM Redbook is part of the WebSphere V6 series; it focuses on security and security-related topics and provides technical details to design and implement secure solutions with WebSphere. This book provides IT Architects, IT Specialists, application designers, application developers, application assemblers, application deployers and consultants with information necessary to design, develop and deploy secure e-business applications using WebSphere Application Server V6. This redbook not only discusses theory but also provides hands-on exercises and sample applications.

Part 1, “Application server security” on page 1 discusses security for the application server and its components, including enterprise applications. You will find essential information on how to secure Web and EJB applications and how to develop a Java client using security.

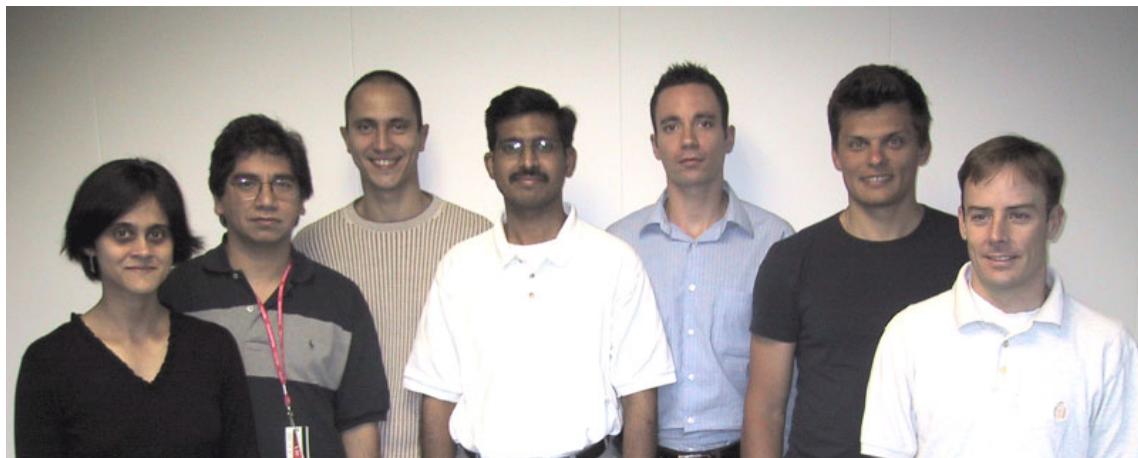
Part 2, “Extending security beyond the application server” on page 185 introduces additional components from the enterprise environment and discusses security beyond the application server. External components include third-party security servers, messaging clients and servers, and database servers.

Part 3, “Development environment” on page 365 is a short introduction to development environment security. Here you can read about guidelines and best practices applicable to a secure development environment.

Finally, Part 4, “Appendices” on page 385 provides additional information related to chapters in the previous parts.

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 redbook team: Irina Singh, Lucky Kartasasmita, Denis Masic, Sudhakar Nagarajan, Emilio Bielsa, Peter Kovari, Matthew Stokes

Peter Kovari is a WebSphere Specialist at the International Technical Support Organization, Raleigh Center. He writes extensively about all areas of WebSphere. His areas of expertise include e-business, e-commerce, security, Internet technologies and mobile computing. Before joining the ITSO, he worked as an IT Specialist for IBM in Hungary.

Emilio Bielsa is an IT Specialist with the Security and Privacy services practice in IBM UK. He has been working for IBM for over four years and has been involved in various customer projects in various areas such as development, technical support and performance testing. For the last two years, he has almost exclusively been working with Tivoli Access Manager. In so doing, he gained experience with LDAP, IT Security, SSO, and various other fields that relate to the product, such as WebSphere integration. He holds a degree in IT from De Montfort University (Leicester, UK) as well as a degree in Telecommunications Engineering from the University of Catalonia (Barcelona, Spain).

Saravana C Chandran is a Senior IT Specialist with the IBM Software Services for WebSphere, helping customers to deploy IBM WebSphere brand products. He has 10 years of experience and holds a Master's degree in Physics and Computer Science.

Lucky Kartasasmita is a WebSphere specialist for IBM Global Services Netherlands. He gathered several years of Java and programming experience before joining the WebSphere infrastructure support group. He works with a number of customers providing technical and programming support in the J2EE and WebSphere areas.

Denis Masic is an Advisory IT Specialist working in IBM Global Services Slovenija as a technical team leader of the EMEA WAP support team. He has been with IBM since 2001, primarily providing support for WebSphere products; his other activities include teaching various WebSphere classes and coaching peer colleagues. He holds a BsC in Computer Science from the University of Ljubljana.

Sudhakar Nagarajan is presently the Globalization team lead for the WebSphere test team in the Software group at 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. He has worked extensively with WebSphere, J2EE and Java programming. He holds a Master's degree in Manufacturing Engineering from REC Tiruchy, India.

Fumiko Satoh is a researcher at the Tokyo Research Laboratory, Japan. She has been with IBM Research for four years, focusing on Web services security. Her experience includes studying metadata for video digest, studying Web services for mobile devices, developing Web services applications with Web services security for customers and designing and developing the Web services security runtime for WebSphere Application Server. Her current research interests include securing Web services and Web services security related specifications. She received her Master's degree in Physics from the Tokyo Institute of Technology.

Irina Singh is a Consulting IT Specialist at IBM. Irina specializes in WebSphere architecture, security and clustering on various platforms. She has managed multiple WebSphere Portal customer implementations in the past three years. Irina has several certifications from Sun and IBM in Java and WebSphere technologies and has also written multiple publications on WebSphere security, LDAP and Linux. Irina has more than eight years of IT experience and holds a Bachelor's degree in Technology in Electrical Engineering from IIT Kanpur.

Matthew Stokes is a Software Engineer at IBM. Matthew has over 10 years of IT and development experience, including eight years of experience with Lotus Domino and five years with IBM Java technologies. His areas of expertise include Linux, DB2, WebSphere Application Server and messaging products, and Lotus Domino. Matthew is a Microsoft Certified System Engineer, Redhat Certified Engineer, IBM Specialist and Lotus Principal CLP. He holds a BS

degree in Computer Science from Brigham Young University and has been with IBM for nine years.

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

Cecilia Bardy

Linda Robinson

Margaret Ticknor

Jeanne Tucker

International Technical Support Organization, Raleigh Center

Peter Birk

Keys Botzum

Ching-Yun Chao

Carlton Mason

Alasdair Nottingham

Ajay Reddy

Kenichiroh Ueno

Become a published author

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

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an email 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

Application server security



Chapter 1

1

Introduction

This chapter serves as a short introduction to the book. It presents the scenarios used in each chapter and gives a quick overview of how the big picture is divided up into scenarios. This chapter also provides a few pointers that will help you find your way around other WebSphere V6 Redbooks.

1.1 Focus on security

The focus in this book is on security, mostly WebSphere Application Server Version 6 security. We will cover not only the application server but other components as well, such as the directory server (for user registry), the reverse proxy security server, etc.

1.2 No introduction to security

If you are familiar with the WebSphere Handbook family and especially the WebSphere V4 and WebSphere V5 Security Redbooks, then you will find this WebSphere V6 redbook to be a bit different. There are two noticeable changes in this book compared to the previous ones.

The first change is that there is no introduction to security. You will not find security fundamentals or basics covered in this book. Leaving these topics aside, the book is shorter and more focused on the current release of WebSphere, version 6.

The security basics and fundamentals are still available for you in a separate Redpaper on the IBM ITSO Web site at:

<http://www.redbooks.ibm.com/abstracts/redp3944.html>

1.3 Scenario-based chapters

Another change in the book is that the individual chapters now focus on application scenarios. Instead of discussing bits and pieces or components, you will find a description of smaller scenarios, for example a scenario illustrating how to secure Web applications.

Figure 1-1 on page 5 shows a diagram you will find at the beginning of each chapter; it shows which major components are covered in that particular chapter. The key components will be emphasized in each diagram, while others will be faded to give a clearer understanding of the topic at hand.

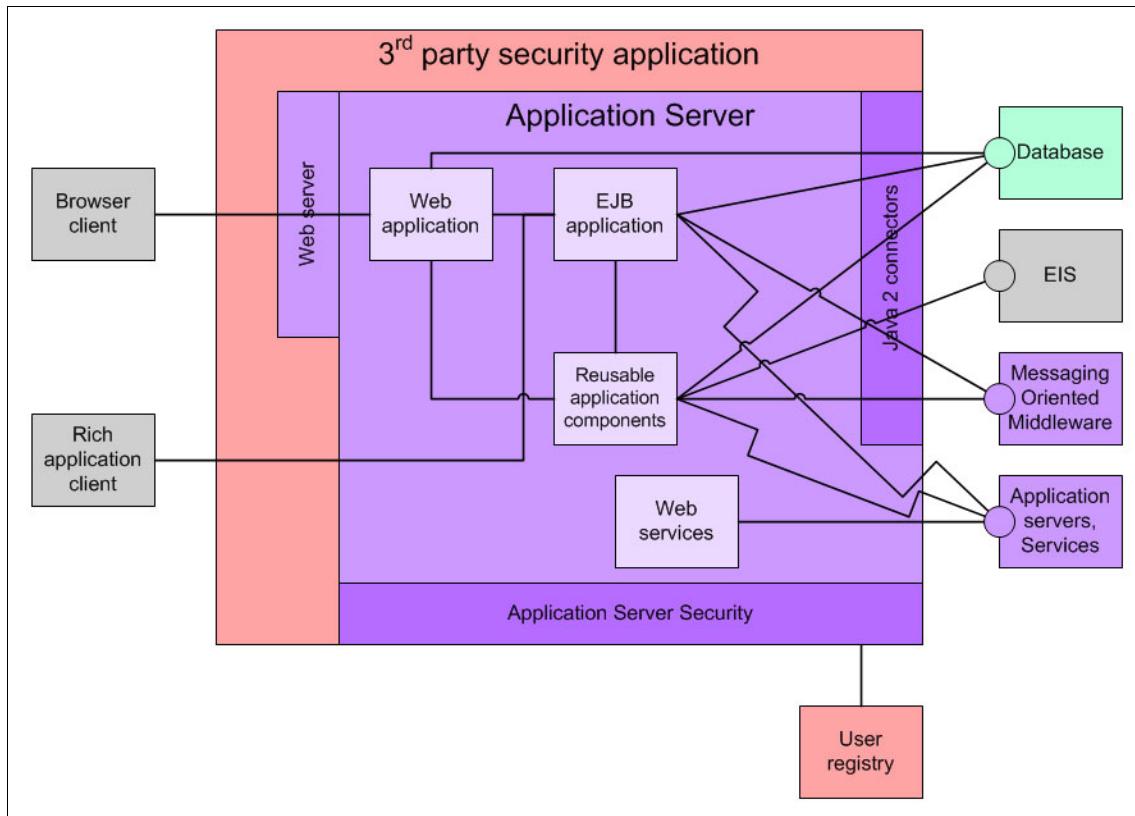


Figure 1-1 The big picture

1.4 Sample applications

Each of the scenarios provides sample configurations and sample applications you can try. The sample applications are available as additional material.

The sample applications in this book are very simple. Their purpose is to show in practice the theory described in the chapters. The samples are more like simple components running in small, self-contained applications.

The samples have changed since the previous Redbooks written about WebSphere security. In this book, there is no one “big” complex sample; the samples in the chapters are not connected or related. This arrangement also make it easier to follow and test each chapter independently.

1.5 WebSphere Information Center

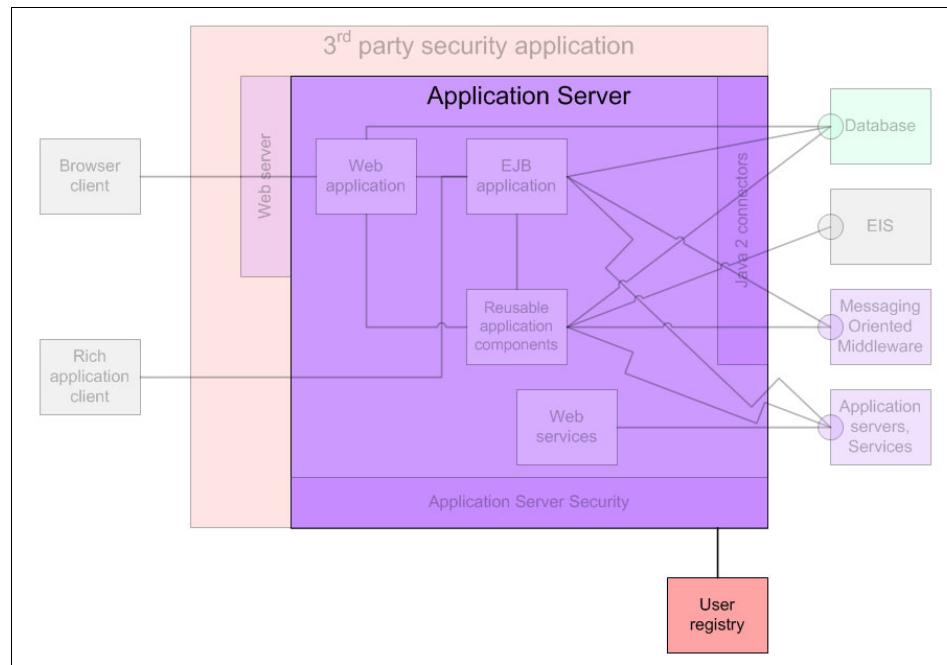
This book is not a replacement for the WebSphere Information Center, which is a great source of information for WebSphere V6. This book and the WebSphere Information Center work as complements to each other. However, please note the following points:

- ▶ This book provides hands-on exercises and follows scenarios to explain the security-related tasks. The WebSphere Information Center is a great reference guide for all the security-related tasks.
- ▶ This redbook follows a linear pattern, even though you can read only parts of the book and move back and forth. In contrast, the WebSphere Information Center contains hypertext documentation; you can easily navigate between topics to find the piece of information you are looking for.

You can find the WebSphere Information Center at:

<http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp>

Configuring the user registry



2.1 User registries

WebSphere Application Server V6 provides a total of three types of user registries:

- ▶ Local OS user registry
- ▶ LDAP user registry
- ▶ Custom user registry

In this chapter, we only look at the user registry configuration; Figure 2-1 shows how the user registry fits in with other Security Authentication components in WebSphere.

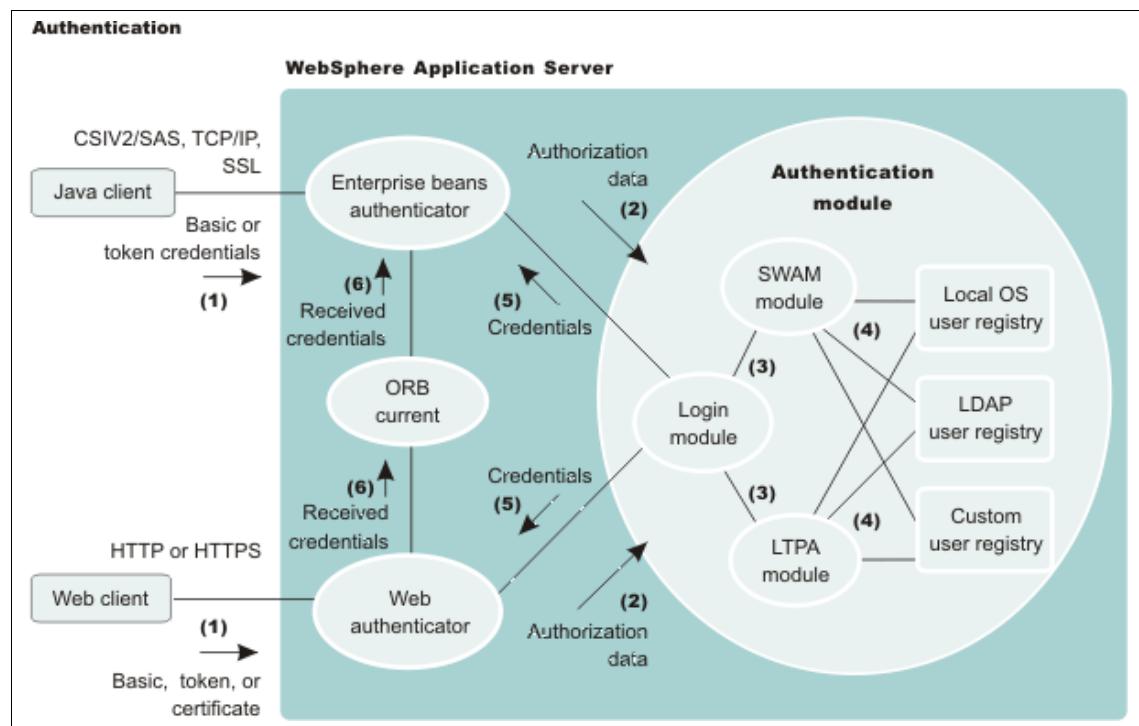


Figure 2-1 WebSphere Application Server Authentication mechanisms

Figure 2-1 demonstrates the steps in the authentication process. Basically, authentication is required for enterprise bean clients and Web clients when they access protected resources. Enterprise bean clients (a servlet or other enterprise beans or a pure client) send the authentication information to a Web application server using one of the following protocols:

- ▶ Common Secure Interoperability version 2 (CSIV2)
- ▶ Secure Authentication Service (SAS)

Web clients use the HTTP or HTTPS protocol to send the authentication information as shown in the previous figure. The authentication information can be Basic Authentication (user ID and password), credential token, or client certificate. The Web authenticator and the EJB authenticator pass the authentication data to the login module (2), which can use Lightweight Third Party Authentication (LTPA) or Simple WebSphere Authentication Mechanism (SWAM).

The authentication module uses the registry that is configured on the system to perform the authentication (4). Three types of registries are supported: LocalOS, Lightweight Directory Access Protocol (LDAP), and a custom registry. External registry implementation following the registry interface specified by IBM can replace either the LocalOS or the LDAP user registry. The login module creates a JAAS subject after authentication and stores the Common Object Request Broker Architecture (CORBA) credential derived from the authentication data in the public credentials list of the subject. The credential is returned to the Web authenticator or EJB authenticator (5).

The Web authenticator and the EJB authenticator store the received credentials in the Object Request Broker (ORB) current for the authorization service to use in performing further access control checks.

2.2 LDAP user registry

In this chapter, we use the IBM Tivoli Directory Server V5.2 that ships with IBM Tivoli Access Manager for e-business V5.1. Figure 2-2 on page 10 shows the directory setup used for this book.

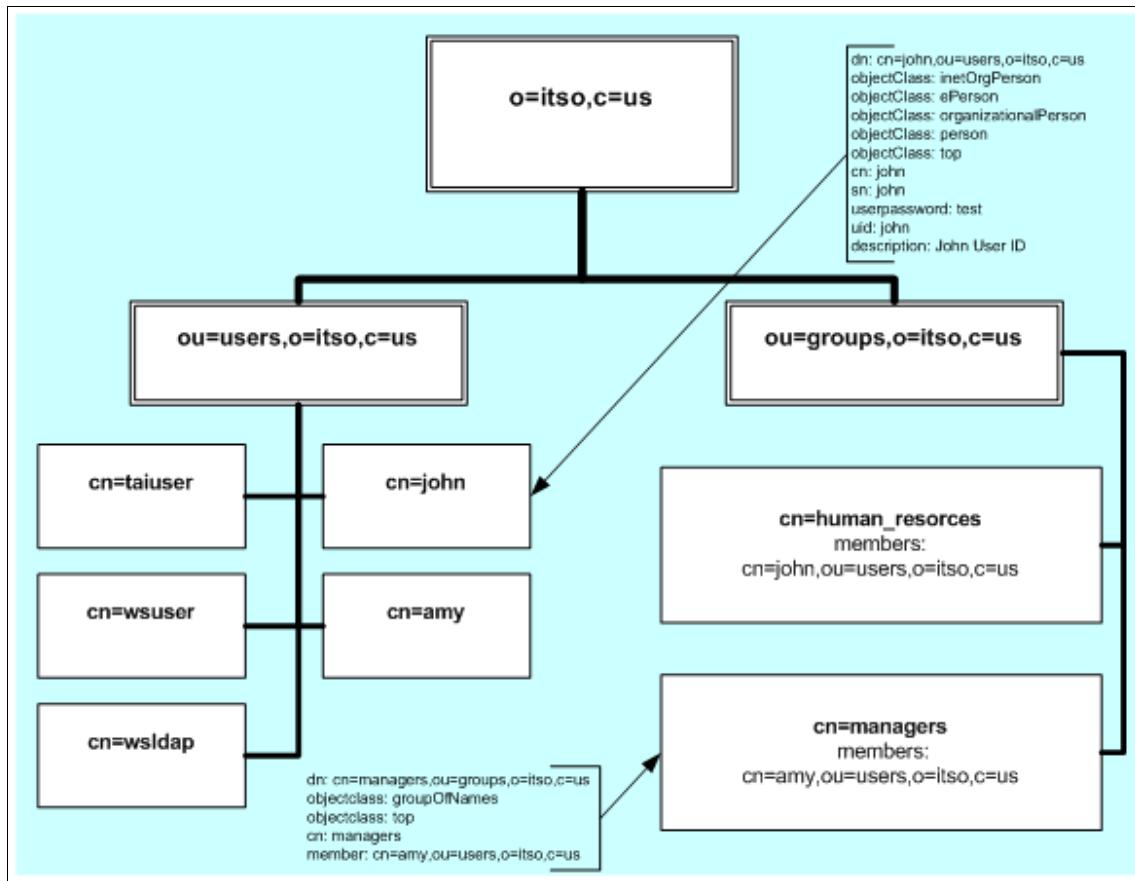


Figure 2-2 LDAP Server structure and data

WebSphere supports several other LDAP servers; please check the latest information about the supported LDAP servers at:

<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>

2.2.1 Basic LDAP configuration for WebSphere Application Server

To configure WebSphere Application Server to use the LDAP registry, perform the following steps:

1. Start your WebSphere Application Server V6 Administrative Console and log in; select **Global Security**, then, in the User Registries section, select **LDAP**.
2. Provide the details for the fields in the Configuration panel as listed below.
 - **Server Id:** this is the WebSphere administrator ID, for example: wsuser.

- **Server Password:** this is the password for the user specified under Server Id.
- **Type:** this is the type of LDAP server; select **IBM Tivoli Directory Server** from the drop-down list provided for the example in this book.
- **Host:** specify the host ID (IP address or DNS name) of the LDAP server.
- **Port:** this is the port number on which the LDAP server is running; by default, it is 389 for the non-secure connection.
- **Base DN:** specify the base DN of your LDAP configuration, for example: o=itso,c=us.
- **Bind DN:** this is the distinguished name for the application server to use to bind to the LDAP server, for example: cn=root.

Tip: In the same way that it is not advisable to run WebSphere Application Server as the root or Administrator user, from a security point of view, it is recommended that, for a production environment, WebSphere be configured with an LDAP ID different from that of cn=root, with only read and search rights in the LDAP server.

- **Bind Password:** this is the password for the application server to use to bind to the LDAP server; provide the password for the user specified under Bind DN.
- **Reuse Connection:** generally, this should be selected. In rare situations, when you use a router to spray the requests to multiple LDAP servers and this router does not support affinity, you should disable this checkbox.

General Properties		Additional Properties
<p>* Server user ID wsuser</p> <p>* Server user password *****</p> <p>Type IBM Tivoli Directory Server</p> <p>* Host ldaphost</p> <p>Port 389</p> <p>Base distinguished name (DN) o=itso,c=us</p> <p>Bind distinguished name (DN) cn=root</p> <p>Bind password *****</p> <p>Search timeout 120 seconds</p>		<ul style="list-style-type: none"> ■ Advanced Lightweight Directory Access Protocol (LDAP) user registry settings ■ Custom properties
<p><input checked="" type="checkbox"/> Reuse connection</p> <p><input checked="" type="checkbox"/> Ignore case for authorization</p> <p><input type="checkbox"/> SSL enabled</p> <p>SSL configuration bc2srv2Node01/DefaultSSLSettings</p>		

Figure 2-3 LDAP settings for WebSphere Application Server

3. Click **Apply**.
4. Save the configuration for WebSphere.
5. We need to define the configuration for Global Security in the Administrative Console. Navigate to **Security** → **Global Security** and from the Active User Registry drop-down list, select **Lightweight Directory Access Protocol (LDAP) user registry** as the active user registry when security is enabled.

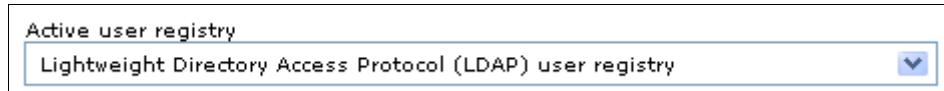


Figure 2-4 LDAP set for Active User Registry

6. Click **Apply**; this will validate the settings.

Note: If the validation fails for any reason, go back to the LDAP configuration panel and check your settings again.

7. Save the configuration for WebSphere, then restart the server.

Testing the LDAP user registry in WebSphere Application Server

To test the connection, please follow the steps in 3.2, “Enabling Global Security” on page 32 to enable Global Security and when the server starts, launch the Administrative Console; it should ask for a user name and password for authentication. This is because Global Security is enabled. Provide the user name as wsuser and password as test if you loaded the directory with the data provided with this book. If you are able to log in successfully, it means your configuration is working properly.

Note: Once you have enabled security, to stop the server you will need to provide the -username and -password parameters for the **stopserver** command script. For example, in a UNIX environment, you stop WebSphere Application Server with the following line:

```
/opt/IBM/WebSphere/AppServer/bin/stopServer.sh server1 -username wsuser  
-password test
```

2.2.2 Advanced LDAP configuration

In order to access the Advanced Lightweight Directory Access Protocol (LDAP) user registry settings, select **Global security** → **LDAP User Registry** → **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**. There might be cases where you want to modify the default settings that WebSphere Application Server uses to search the LDAP server. The default values are based on the type of LDAP server selected on the LDAP Settings panel. Table 2-1 on page 14 shows the default search settings for IBM Tivoli Directory Server.

Table 2-1 Advanced LDAP Settings for Tivoli Directory Server

Property: Default Value	Description
User Filter: (&(uid=%v)(objectclass=ePerson))	Specifies the LDAP user filter used to search the registry for users.
Group Filter: (&(cn=%v)(!(objectclass=groupOfNames) (objectclass=groupOfUniqueNames)))	Specifies the LDAP group filter used to search the registry for users.
User ID map: *:uid	Specifies the LDAP filter that maps the short name of a user to an LDAP entry. This field takes multiple objectclass:property pairs delimited by a semicolon (;).
Group ID Map: *:cn	Specifies the LDAP filter that maps the short name of a group to an LDAP entry. This field takes multiple objectclass:property pairs delimited by a semicolon (;).
Group member ID map: ibm-allGroups:member;ibm-allGroups:uniqueMember	Specifies the LDAP filter that identifies user to group relationships.
Perform nested group search: un-checked	Select this option if the Lightweight Directory Access Protocol (LDAP) server does not support recursive server-side group member searches.
Certificate map mode: EXACT_DN	Specifies whether to map X.509 certificates into an LDAP directory by EXACT_DN or CERTIFICATE_FILTER. Specify CERTIFICATE_FILTER to use the specified certificate filter for the mapping.
Certificate filter:	The filter is used to map attributes in the client certificate to entries in the LDAP registry. The syntax or structure of this filter is: LDAP attribute=\${Client certificate attribute} (for example, uid=\${SubjectCN}).

2.2.3 Configuring LDAP access over SSL

This section allows you to configure the LDAP connection for WebSphere Application Server V6 by following the steps from 2.2, “LDAP user registry” on page 9. The IBM Tivoli Directory Server V5.2 used in this example is already configured to use SSL.

This topic describes how to establish a Secure Sockets Layer (SSL) connection between WebSphere Application Server V6 and a Lightweight Directory Access Protocol (LDAP) server.

WebSphere Application Server key configuration

Before you can configure WebSphere Application Server V6 to use SSL to communicate with the LDAP server, you must extract the LDAP server certificate from the LDAP key store and import it into the application server's key store that is used for LDAP connection. Assuming a default installation of WebSphere Application Server V6, you need to follow these steps:

1. Open the IBM Tivoli Directory Server V5.2 key store with the ikeyman tool (for more information about how to use the tool, please refer to the IBM Redpaper *WebSphere Security Fundamentals*, REDP3944). To find the location of the keystore that IBM Tivoli Directory Server V5.2 uses, look up the *ibm-slapdSslKeyDatabase* parameter in the ibmslapd.conf configuration file; in our case this is /etc/ldap_key.kdb.
Export the key as ldap_key.arm.
2. To load the certificate extracted in step 1 into the keystore used by WebSphere Application Server V6, you must open the key store with the ikeyman tool. We used the default key store for LDAP security, which is located in {Websphere_root}/profiles/default/etc/DummyServerTrustFile.jks.
Import the ldap_key.arm into the key store.

Configuring WebSphere User Registry with SSL

Assuming you have WebSphere working with the LDAP registry from 2.2.1, “Basic LDAP configuration for WebSphere Application Server” on page 10, the following steps will guide you through the configuration of WebSphere Application Server V6 to enable SSL for the Lightweight Directory Access Protocol client.

1. Start the WebSphere Application Server V6 Administrative Console and log in; select **Global Security**, then in the User Registries section click **LDAP**.
2. Update the panel as in Figure 2-5 on page 16. You must check the **SSL** checkbox and change the port to **636**, default SSL LDAP (LDAPS) port, to enable SSL. Click **Apply** and save. The default SSL settings, **Security → SSL**, will not need to be modified as we have chosen to overwrite the default server trust keyring. For more information about the SSL settings please refer to 3.6, “SSL configurations” on page 45.

General Properties		Additional Properties
<p>* Server user ID wsuser</p> <p>* Server user password *****</p> <p>Type IBM Tivoli Directory Server</p> <p>* Host ldaphost</p> <p>Port 636</p> <p>Base distinguished name (DN) o=itso,c=us</p> <p>Bind distinguished name (DN) cn=root</p> <p>Bind password *****</p> <p>Search timeout 120 seconds</p>		<ul style="list-style-type: none"> ■ Advanced Lightweight Directory (LDAP) user registry settings ■ Custom properties
<input checked="" type="checkbox"/> Reuse connection <input checked="" type="checkbox"/> Ignore case for authorization <input checked="" type="checkbox"/> SSL enabled <p>SSL configuration bc2srv1Node01/DefaultSSLSettings</p>		

Figure 2-5 Enabling SSL for the LDAP User Registry

3. Click **Apply**; this will validate the settings.
4. Save the configuration for WebSphere, then restart the application server.

Testing the LDAP SSL connection with WebSphere

As in “Testing the LDAP user registry in WebSphere Application Server” on page 13, to test the connection, please follow the steps in 3.2, “Enabling Global Security” on page 32 to enable Global Security. The assumption here is that the LDAP connection has been tested and found to be working with Global Security Enabled and the purpose of this test is to make sure that WebSphere Application

Server V6 is now communicating with LDAP using SSL on the 636 port. The simplest way to check will be to examine the network connections opened, after restarting is to execute the command **netstat**, on the WebSphere Application Server V6 machine. This command should work in both Windows and UNIX systems.

Example 2-1 netstat command in UNIX

Active Internet connections (w/o servers)					
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	wshost:32965	1daphost:1daps	ESTABLISHED

Example 2-2 netstat command in Windows

Active Connections				
Proto	Local Address	Foreign Address	State	
TCP	1daphost:32965	wshost:1daps	ESTABLISHED	

If netstat reports that the 1daps/636 port is being used, WebSphere Application Server V6 is using SSL to communicate with the LDAP server. When the server starts, launch the Administrative Console; provide the user name and password wsuser, test. If you are able to log in successfully, it means your configuration was successful.

2.3 Local OS User Registry

With the local operating system, or Local OS, user registry implementation, the WebSphere Application Server authentication mechanism can use the user accounts database of the local operating system.

The respective operating system APIs are called by the product processes (servers) for authenticating a user and other security-related tasks (for example, getting user or group information). Access to these APIs are restricted to users who have special privileges. These privileges depend on the operating system and are described below.

Required privileges in Windows

The user running the WebSphere Application Server process requires proper operating system privileges to call the Windows systems application programming interface (API) for authenticating and obtaining user and group information from the Windows operating system. This user logs into the machine,

or if running as a service, is the *Log On As* user. Depending on the machine and whether the machine is a stand-alone machine or a machine that is part of a domain or is the domain controller, the access requirements vary.

- ▶ For a stand-alone machine, the user:
 - Is a member of the *administrative* group.
 - Has the *Act as part of the operating system* privilege.
 - Has the *Log on as a service* privilege, if the server is run as a service.
- ▶ For a machine that is a member of a domain, only a domain user can start the server process; that user:
 - Is a member of the *domain administrative* groups in the domain controller.
 - Has the *Act as part of the operating system* privilege in the Domain security policy on the domain controller.
 - Has the *Act as part of the operating system* privilege in the Local security policy on the local machine.
 - Has the *Log on as a service* privilege on the local machine, if the server is run as a service.
 - The user is a *domain user* and not a local user, which implies that when a machine is part of a domain, only a domain user can start the server.
- ▶ For a domain controller machine, the user:
 - Is a member of the *domain administrative* groups in the domain controller.
 - Has the *Act as part of the operating system* privilege in the Domain security policy on the domain controller.
 - Has the *Log on as a service* privilege on the domain controller, if the server is run as a service.

Note: For more information on how to configure the required users for Windows please refer to the WebSphere WebSphere Information Center (search for the *csec_localos* topic ID) or read the operating system's documentation.

Required privileges with UNIX

The user that is running the process ID that runs the WebSphere Application Server process needs *root* authority to call the local operating system APIs for authentication and for obtaining user or group information. With WebSphere Application Server in UNIX systems, only the local machine registry can be used, the Network Information Service (NIS) (Yellow Pages) is not supported.

Configuring WebSphere Application Server V6

To configure WebSphere to use the local Operating System's registry, open the Administrative Console and select **Security → User Registries → Local OS**.

1. Enter a user name and password, with sufficient OS privileges as explained earlier. This is the identity under which the request is sent to the registry. The LocalOS registry requires an identity for a user that has administrative capabilities, as explained in the previous section.
2. Click **OK**. It should not be necessary to set any custom properties for the Local OS registry.

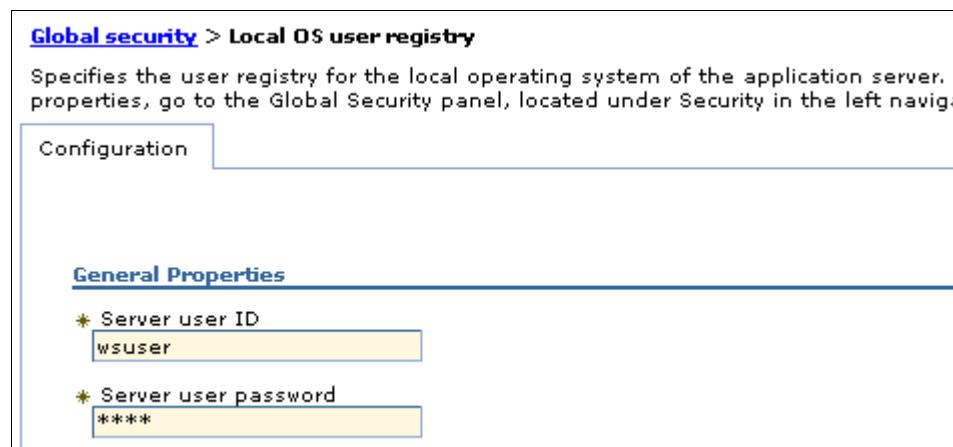


Figure 2-6 LocalOS registry user name and password

3. If there are no errors at this stage, select **Security → Global Security**. Ensure that the Active User Registry option is set to **Local OS** and that Global Security is enabled. If this is not the case, make the necessary changes.

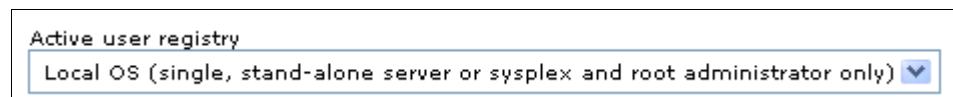


Figure 2-7 LocalOS set for Active User Registry

4. Click **Apply**; this will validate the settings.
5. Save the configuration for WebSphere.
6. Restart your WebSphere Application Server V6.

Testing the Local OS user registry

To test the connection, please follow the steps in 3.2, “Enabling Global Security” on page 32 to enable Global Security. When the server starts, launch the

Administrative Console; it should ask for a user name and password for authentication. If you are able to log in successfully then your configuration was successful.

Tip: If WebSphere fails to start, after enabling Global Security, it might be caused by a problem with the User Registry. If that is the case, you will not be able to login to the Administrative Console, we need another solution to disable security. To disable global security manually for WebSphere, refer to 3.3, “Disabling Global Security” on page 39.

2.4 Custom User Registry

WebSphere can be configured to use a type of user registry other than Local OS or LDAP. This registry is referred to as a Custom Registry. The custom registry feature supports any user registry that is not implemented by WebSphere Application Server. You can use any user registry by implementing the UserRegistry interface.

2.4.1 The UserRegistry interface

The UserRegistry interface is very helpful in situations, for example, where the current user and group information exists in some other format (for example, a database) and cannot be moved to Local OS or LDAP. In such a case, implement the UserRegistry interface so that WebSphere Application Server can use the existing registry for all of the security-related operations. Using a custom registry is a software implementation effort; it is expected that the implementation does not depend on other WebSphere Application Server resources, for example, data sources, for its operation.

To implement the UserRegistry interface it will be necessary to provide a Java class that provides WebSphere with a standard interface in order for WebSphere to communicate with the registry in an appropriate fashion. The provision of this interface ensures that a variety of user registries may be used, such as relational databases, files stored on directly on the file system. A combination of multiple registries may be used, such as LDAP and RACF.

The UserRegistry interface defines a general set of methods to allow the application server to obtain user and group information from the registry, the interface is also implemented by the two other available user registries in WebSphere Application Server V6, LDAP and Local OS. The registry can operate as a process running remotely to the application server and thus it is necessary for each registry to implement the `java.rmi.Remote` interface.

There is one point worth noting in regard to the initialization of a WebSphere Application Server V6 custom registry. With V4, it was possible to use other WebSphere Application Server components to initialize the custom registry. For example, a data source might have been used to connect to a database-based custom registry or one may have made use of a deployed EJB. However, since V5, neither of these examples is possible because, unlike in V4, the security mechanism is initialized before other components such as containers, and therefore, these facilities are not available when the security component is started. Therefore any implementation of the custom registry should not depend on any WebSphere Application Server component such as data sources, enterprise beans, and so on.

The methods in the `UserRegistry` interface operate on the following information for users:

- `userSecurityName`: This is the user name used to log on when prompted by an application.
- `uniqueUserId`: This ID represents a unique identifier for the user; it is equivalent to the `uid` in UNIX or the `dn` in LDAP.
- `userDisplayName`: This name is an optional string that describes a user.
- `groupSecurityName`: Represents the security group.
- `groupUniqueId`: This ID represents a unique identifier for the group.
- `groupDisplayName`: This name is an optional string that describes a group.

The list below includes all the methods defined in the `UserRegistry` interface. Each method must be implemented by the custom registry.

Table 2-2 WebSphere's `UserRegistry` interface

Method signature	Use
<code>void initialize(java.util.Properties props) throws CustomRegistryException, RemoteException</code>	Initializes the registry. This method is called when creating the registry.
<code>String checkPassword(String userSecurityName, String password) throws PasswordCheckFailedException, CustomRegistryException, RemoteException</code>	Checks the password of the user. This method is called to authenticate a user when the user's name and password are given.
<code>String mapCertificate(X509Certificate[] cert) throws CertificateMapNotSupportedException, CertificateMapFailedException, CustomRegistryException, RemoteException</code>	Maps a Certificate (of X509 format) to a valid user in the Registry. This is used to map the name in the certificate supplied by a browser to a valid <code>userSecurityName</code> in the registry.

Method signature	Use
String getRealm() throws CustomRegistryException, RemoteException	The realm is a registry-specific string indicating the <i>realm</i> or <i>domain</i> for which this registry applies. For example, for OS400 or AIX this would be the host name of the system whose user registry this object represents. If null is returned by this method realm defaults to the value of "customRealm".
Result getUsers(String pattern, int limit) throws CustomRegistryException, RemoteException	Gets a list of users that match a <i>pattern</i> in the registry. The maximum number of users returned is defined by the <i>limit</i> argument.
String getUserDisplayName(String userSecurityName) throws EntryNotFoundException, CustomRegistryException, RemoteException	Returns the display name for the user specified by userSecurityName.
String getUniqueUserId(String userSecurityName) throws EntryNotFoundException, CustomRegistryException, RemoteException	Returns the UniqueId for a userSecurityName. This method is called when creating a credential for a user.
String getUserSecurityName(String uniqueUserId) throws EntryNotFoundException, CustomRegistryException, RemoteException	Returns the name for a user given its uniqueId.
boolean isValidUser(String userSecurityName) throws CustomRegistryException, RemoteException	Determines if the <i>userSecurityName</i> exists in the registry.
Result getGroups(String pattern, int limit) throws CustomRegistryException, RemoteException	Gets a list of groups that match a <i>pattern</i> in the registry. The maximum number of groups returned is defined by the <i>limit</i> argument.
String getGroupDisplayName(String groupSecurityName) throws EntryNotFoundException, CustomRegistryException, RemoteException	Returns the display name for the group specified by groupSecurityName.
String getUniqueGroupId(String groupSecurityName) throws EntryNotFoundException, CustomRegistryException, RemoteException	Returns the Unique id for a group.
List getUniqueGroupIds(String uniqueUserId) throws EntryNotFoundException, CustomRegistryException, RemoteException	Returns the UniqueIds for all the groups that contain the UniqueId of a user. Called during creation of a user's credential.
String getGroupSecurityName(String uniqueGroupId) throws EntryNotFoundException, CustomRegistryException, RemoteException	Returns the name for a group given its uniqueId.

Method signature	Use
boolean isValidGroup(String groupSecurityName) throws CustomRegistryException, RemoteException	Determines if the <i>groupSecurityName</i> exists in the registry.
Result getUsersForGroup(String groupSecurityName, int limit) throws NotImplementedException, EntryNotFoundException, CustomRegistryException, RemoteException	Gets a list of users in a group. The maximum number of users returned is defined by the <i>limit</i> argument.
public List getGroupsForUser(String userSecurityName) throws EntryNotFoundException, CustomRegistryException, RemoteException	Gets all the groups the given user is a member of.
Credential createCredential(String userSecurityName) throws NotImplementedException, EntryNotFoundException, CustomRegistryException, RemoteException	Throws the NotImplementedException for this method.

2.4.2 File-based Custom User Registry sample

A sample custom registry implementation is provided with the application server. The custom registry class is called `FileRegistrySample`. The class is installed with WebSphere Application Server V6 and the source code is provided in the WebSphere Information Center for reference purposes. Refer to the WebSphere Information Center for details regarding the format of these files and two sample files. The files must be copied to the directories specified in the initialization properties for the custom registry before the registry can be enabled.

Table 2-3 *FileRegistrySample* initialization properties

Name	Value
usersFile	file location and name, for example: \${USER_INSTALL_ROOT}/customer_sample/users.props
groupsFile	file location and name, for example: \${USER_INSTALL_ROOT}/customer_sample/groups.props

To configure the application server to make use of the file custom registry, follow the steps below:

1. Launch the Administrative Console and login, select **Global Security**, then in the User Registries section, click **Custom**.
2. Enter the user name and password of an identity under which the custom registry will operate. Enter the class name for the custom registry. For example, the flat file-based sample registry is `com.ibm.websphere.security.FileRegistrySample`. Click **Apply**.

Global security > Custom user registry

Specifies a custom user registry that implements the com.ibm.websphere.security.ICompatibility interface. WebSphere Application Server also supports a custom user registry that implements the com.ibm.websphere.security.CustomRegistry interface. When security is enabled and security settings are changed, go to the Global Security panel, located under Security in the left navigation bar, and validate the changes.

Configuration

General Properties

* Server user ID
bob

* Server user password

* Custom registry class name
com.ibm.websphere.security.FileRegistrySample

Ignore case for authorization

Figure 2-8 FileRegistry Sample Custom Registry user name and password

3. Click **Custom Properties** and add the properties necessary to initialize the registry. These properties will be passed to the initialize method of the custom registry. For the supplied FileRegistrySample code, enter the properties as in Table 2-3 on page 23.

Global security > Custom user registry > Custom properties		
Specifies arbitrary name and value pairs of data. The name is a property key and is used to set internal system configuration properties.		
<input type="checkbox"/> Preferences		
<input type="button" value="New"/>	<input type="button" value="Delete"/>	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Select	Name  	Value  
<input type="checkbox"/>	groupsFile	<code>#{USER_INSTALL_ROOT}/customer_sample/groupsFile</code>
<input type="checkbox"/>	usersFile	<code>#{USER_INSTALL_ROOT}/customer_sample/usersFile</code>
Total 2		

Figure 2-9 File Registry Sample Custom Properties.

4. If there are no errors at this stage, select **Security** → **Global Security**. Ensure that the Active User Registry option is set to **Custom user registry** and that Global Security is enabled. If this is not the case, make the necessary changes.

Active user registry
<input style="width: 150px; height: 20px; border: 1px solid black; padding: 2px; font-size: small; font-weight: bold; background-color: #f0f0f0; color: black; text-decoration: none; margin-left: 10px;" type="button" value="Custom user registry"/>

Figure 2-10 Custom Registry set for Active User Registry

5. Click **Apply**; this will validate the settings.
6. Save the configuration for WebSphere.
7. Restart the application server.

Testing the Custom Registry

To test the connection please follow the steps in 3.2, “Enabling Global Security” on page 32 to enable Global Security. When the server starts, launch the Administrative Console; it should ask for a user name and password for authentication. If you are able to log in successfully then your configuration was successful.

2.4.3 DB2 Custom User Registry Sample

The following section documents the implementation of a DB2 custom registry. The DB2 registry uses JDBC to communicate with the database. Although this registry was tested with DB2, it should be possible to modify it to work with other relational databases. The source code (DB2UserRegistrySample.java) is included in the files associated with this book, along with the database structure which follows that of the LDAP registry.

Open the DB2UserRegistrySample.java source in Rational Application Developer V6 and check the comments in the source code. You will find all the required methods for the UserRegistry interface implemented. Look for the SQL queries in the code and see what each method does with the database,

Although this can be modified, the sample instructions use the DB2 JDBC Universal Driver (Type 4) to successfully run and compile the application the DB2 Type 4 driver libraries (db2jcc.jar, db2jcc_javax.jar, db2jcc_license_cu.jar) and the WebSphere security libraries (sas.jar wssec.jar) will need to be added to the “Java Build Path” as can be seen in Figure 2-11.

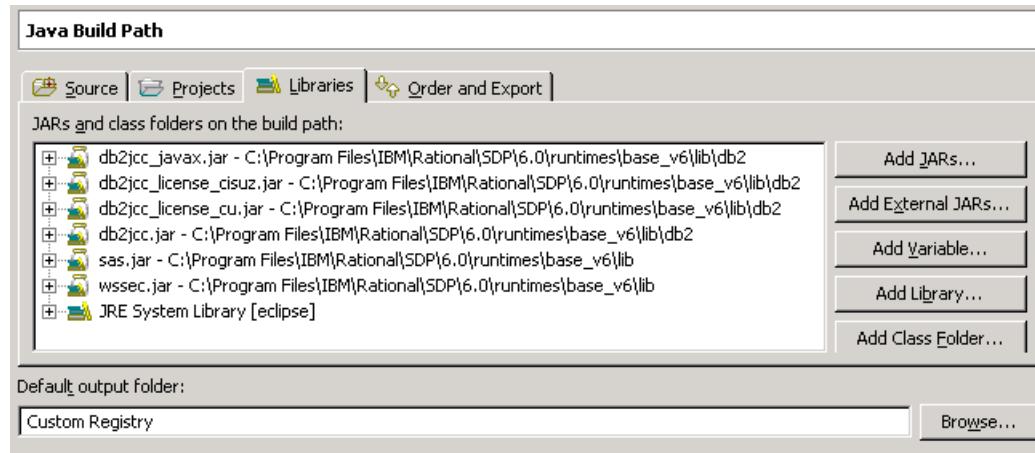


Figure 2-11 Rational Application Developer libraries required

The libraries (db2.jar files) above and the compiled DB2UserRegistrySample.class file must be present in a directory accessible by the application server, that is, a directory that is in the application server's classpath (for example, <WebSphere_root>/lib/ext). Alternatively, update the application server's classpath to refer to the directory that contains the class file and .jar files.

A simple custom registry test class *DB2UserRegistrySampleTest*, that runs from the command line or from Rational Application Developer is included and can be used to test whether the custom registry is working as required. The tool allows the developer to be sure that the custom registry is functioning before configuring the application server to use it.

Example 2-3 DB2UserRegistrySampleTest output

```
Initialized DB2UserRegistrySample
Enter a user name. wsuser
Enter a UID. 1
Enter a group name. admngrp
Enter a GID. 1
Enter a password. test
X509 certificate file.
Testing registry...
checkPassword: wsuser
getGroupDisplayName: group for administrators
getGroups: com.ibm.websphere.security.Result@1bb97283
getGroupSecurityName: admngrp
getRealm: customRealm
getUniqueGroupId: 1
getUniqueGroupIds: [1]
getGroupsForUser: [admingrp]
getUniqueUserId: 1
getUserDisplayName: WebSphere administrator
getUsers: com.ibm.websphere.security.Result@1eeab283
getUserSecurityName: wsuser
isValidGroup: true
isValidUser: true
mapCertificate: null
Test completed.
```

To run the *DB2UserRegistrySampleTest* tool, you must provide two arguments, the Custom Registry class, *DB2UserRegistrySample*, and the Custom Registry property file filename; the property file contains the information as shown in Table 2-4 on page 28. The tool will ask for some user and group information and use this information to query the custom registry. It will also ask for a X.509 certificate file, although the response can be empty (just press **Enter**). In this case, the certificate check will not be performed. The compiled classes are provided with the book as part of the additional material and the DB2 libraries are available together with the DB2 product.

Table 2-4 DB2RegistrySample initialization properties

Name	Value
DBDRIVER	com.ibm.db2.jcc.DB2Driver
DBURL	jdbc:db2://9.42.171.75:50000/userreg
DBUSERNAME	webas
DBPASSWORD	test
DBSCHEMA	userreg

Configure the application server to make use of the custom registry, follow the steps below:

1. Launch the Administrative Console and log in, then select **Global Security**, then in the User Registries section, select **Custom**.
2. Enter the user name and password of an identity under which the custom registry will operate. Enter the class name for the custom registry. For example, the supplied DB2 custom registry is DB2UserRegistrySample. Click **Apply**.

Global security > Custom user registry

Specifies a custom user registry that implements the com.ibm.websphere.security.IWebSphere Application Server also supports a custom user registry that implements interface. When security is enabled and any of the properties on this panel are changed, click Apply or OK to validate the changes.

Configuration	
General Properties	
* Server user ID	wsuser
* Server user password	*****
* Custom registry class name	DB2UserRegistrySample
<input type="checkbox"/> Ignore case for authorization	

Figure 2-12 DB2 Registry Sample Custom Registry user name and password

- Click **Custom Properties** and add the properties necessary to initialize the registry. These properties will be passed to the initialize method of the custom registry; see Figure 2-13 for details.

Global security > Custom user registry > Custom properties

Specifies arbitrary name and value pairs of data. The name is a property key and the value is a string value that can be used in system configuration properties.

[Preferences]

Select	Name	Value	Description
<input type="checkbox"/>	DBDRIVER	com.ibm.db2.jcc.DB2Driver	
<input type="checkbox"/>	DBPASSWORD	test	
<input type="checkbox"/>	DBSCHEMA	userreg	
<input type="checkbox"/>	DBURL	jdbc:db2://9.42.171.75:50000/userreg	
<input type="checkbox"/>	DBUSERNAME	webas	

Total 5

Figure 2-13 DB2 Registry Sample Custom Properties

- If there are no errors at this stage, select **Security → Global Security**. Ensure that the Active User Registry option is set to Custom user registry and that Global Security is enabled. If this is not the case, make the necessary changes.



Figure 2-14 Custom Registry set for Active User Registry

- Click **Apply**; this will validate the settings.
- Save the configuration for WebSphere.
- Restart the application server.

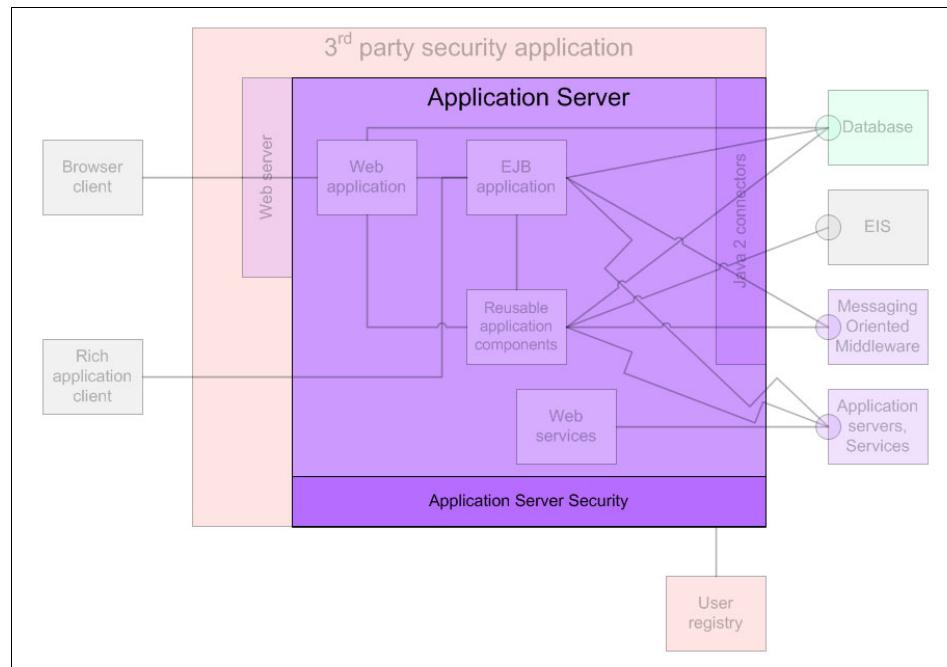
Testing the Custom Registry

To test the connection, please follow the steps in 3.2, “Enabling Global Security” on page 32 to enable Global Security. When the server starts, launch the Administrative Console; it should ask for a user name and password for authentication. If you are able to log in successfully then your configuration was successful.



Chapter 3

Global Security



3.1 Introduction

The term *Global Security* represents the security configuration that is effective for the entire *security domain*. The basic requirement for a security domain is that the access ID returned by the registry from one server be the same access ID as that returned from the registry on any other servers within the same security domain.

Global security applies to all applications running in the environment. It determines whether security is used at all, the type of registry against which authentication takes place, the type of authentication mechanism and some other security values.

Global security has to be enabled in case the *declarative security* is used by any application deployed in the application server. However, if your application relies only on the *programmatic security*, for example using the `HttpServletRequest` interface method `getRemoteUser()`, where authentication is already done on the HTTP server side, you do not necessarily have to enable Global Security.

3.2 Enabling Global Security

Once the user registry has been set, the WebSphere Global Security can be enabled via the Administrative Console by clicking **Security** → **Global security**.

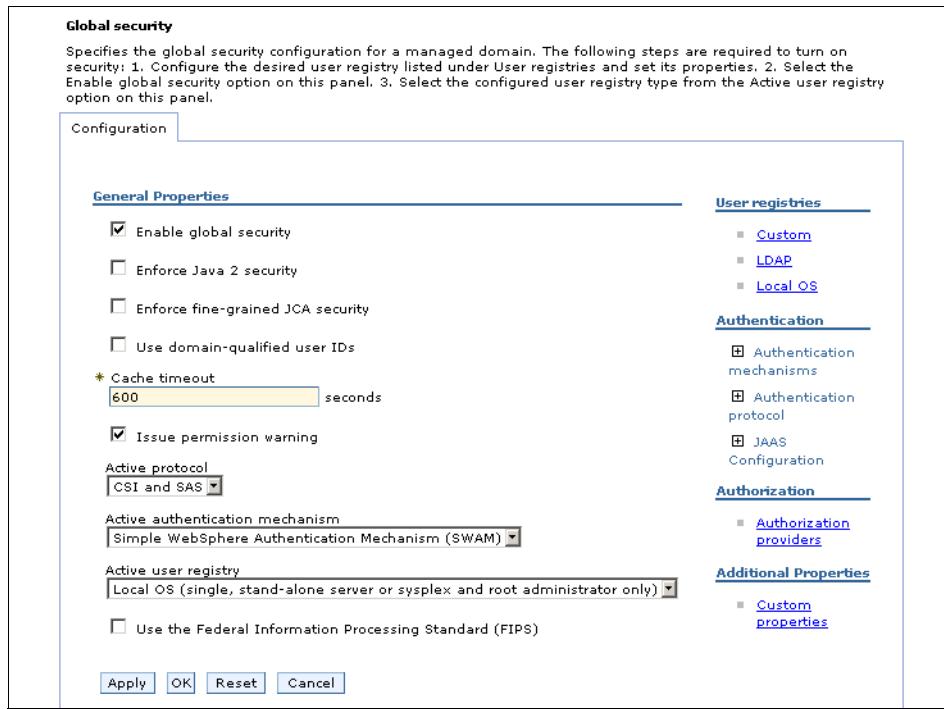


Figure 3-1 Global security configuration page

3.2.1 Main configuration parameters for Global Security

There are three minimum criteria to be filled to enable Global Security:

- ▶ Active protocol(s)

This is the active authentication protocol for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI IIOP) requests, used when security is enabled. WebSphere Application Server can be configured to support both Common Secure Interoperability version 2 (CSIV2) and IBM's Secure Authentication Service (IBM's SAS). IBM's SAS is the authentication protocol used by all releases of WebSphere Application Server prior to version 5. The CSIV2 has been defined by the Object Management Group (OMG) as a standard authentication protocol so that vendors can interoperate securely. The implemented CSIV2 in WebSphere Application Server has more features than the IBM's SAS. If all the servers within the security domain are WebSphere version 5 or 6 servers, specifying the CSIV2 protocol should be good enough.

► Active authentication mechanism

The WebSphere Application Server supports two authentication mechanisms: Simple WebSphere Authentication Mechanisms (SWAM) and Lightweight Third Party Authentication (LTPA). SWAM does not support forwardable credentials, so it is only suitable for simple, non-distributed, single application server environments. Because SWAM uses session ID for identification, the use of SSL is also strongly recommended if SWAM is chosen for the authentication mechanism.

On the other hand, LTPA is intended for distributed application server environments. It supports forwardable credentials and therefore supports Single Sign-On (SSO). If you choose LTPA, further configuration is needed and is explained in “Configuring TAI in WebSphere Application Server” on page 254.

► Active user registry

There are three types of user registries which can be used: Local OS, Lightweight Directory Access Protocol (LDAP) or the custom registry. Details regarding user registry can be seen in Chapter 2., “Configuring the user registry” on page 7. Make sure that the server user ID and password fields are correctly filled.

Whenever Local OS is chosen for the user registry, only special user(s) will be able to enable Global Security, and later on to start the secure WebSphere (see also 2.3, “Local OS User Registry” on page 17):

- For Unix-based platforms, the WebSphere Application Server process must be *owned* by a user with a root authority
- For Windows-based platforms, WebSphere has to be *started* by a user who has the “Act as part of the operating system” rights. Make sure that the system is rebooted if you have to change the rights, otherwise WebSphere might not pick up the changes.

The following table summarizes the authentication mechanism capabilities and user registries.

Table 3-1 Authentication mechanism capabilities

	Forwardable Credentials	SSO	LocalOS User Registry	LDAP User Registry	Custom User Registry
SWAM	No	No	Yes	Yes	Yes
LTPA	Yes	Yes	Yes	Yes	Yes

Once those three criteria are set, Global Security can be enabled.

Important: Restart the Application Server to activate the changes made within Global Security.

Once the server has been restarted, to gain access to the Administrative Console you can use the server user ID and password defined in the user registry. Adding access to the Administrative console for other users or groups will be discussed in 3.4, “Administrative roles” on page 40.

Verifying and testing Global Security

Once your server has been restarted in secure mode, it is wise to test that security is properly enabled. There are several basic tests that can be performed.

1. Verify the form login. When using the Web-based Administrative Console, the Web-based form login page shown forces us to fill in a user ID and password. Only a user ID with administrative roles should be able to log in.
2. Verify the Basic Authentication login. If *DefaultApplication* is installed, test the Web-based Basic Authentication by accessing the URL:
`http://<hostname>:<port>/snoop` (the default <port> is 9080). A challenge login window should appear and you can type *any* user ID and password defined in the custom registry.
3. Verify that the Java Client Basic Authentication works fine by executing:

```
<WebSphere_home>\bin\dumpNameSpace.bat
```

A challenge login window should appear. Although you might be able to just click **Cancel**, you should type *any* correct user ID and password defined in the custom registry to test the security.

Be aware that the login panel for the above Java client will only appear if the property `com.ibm.CORBA.loginSource` is set to prompt in the file `sas.client.props`. Clicking **Cancel** will work only if the CosNaming security (see 3.5, “Naming service security: CosNaming roles” on page 43) allows *read access* to everyone. Those values are the default values when you installed WebSphere.

Successfully running the above basic tests indicates that Global Security is working correctly.

Application considerations

One of the most common problems that occurs when Global Security is enabled, is that the `getRemoteUser()` or `getUserPrincipal()` methods of the `HttpServletRequest` interface returns a null value. This can happen if, for example, authentication is done in the HTTP Server container before reaching the WebSphere Application Server. Whenever Global Security is enabled,

WebSphere will only pass the authentication token to *secure* resources within its container. To secure those resources, you need to add a security constraint within the application descriptor file, web.xml, as shown in Example 3-1.

Example 3-1 Securing the resource /securedhello URI

```
<security-constraint>
    <display-name>Authenticated</display-name>
    <web-resource-collection>
        <web-resource-name>Authenticated Resources</web-resource-name>
        <url-pattern>/securedhello</url-pattern>
        <http-method>PUT</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <description>Authorized guest roles</description>
        <role-name>ServletGuest</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>INTEGRAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>

<security-role>
    <description>Authenticated guest for servlet</description>
    <role-name>ServletGuest</role-name>
</security-role>
```

Do not forget to define a correct security role mapping for the role that you have added when deploying your application (in Example 3-1, the name of the role is **ServletGuest**). If the application has already been installed, you can edit the security role mapping by selecting **Applications** → **Enterprise Application** → **<your_application>** → **Map security roles to users/groups**.

The screenshot shows the 'Mapping users to roles' interface. At the top, there are buttons for 'Look up users' and 'Look up groups'. Below is a table with columns: Select, Role, Everyone?, All authenticated?, Mapped users, and Mapped groups. Two rows are present: one for 'Anonymous' (selected) and one for 'ServletGuest' (selected). The 'Everyone?' checkbox is checked for both rows, while 'All authenticated?' is unchecked. There are icons for adding and deleting rows at the top of the table.

Select	Role	Everyone?	All authenticated?	Mapped users	Mapped groups
<input type="checkbox"/>	Anonymous	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
<input type="checkbox"/>	ServletGuest	<input type="checkbox"/>	<input checked="" type="checkbox"/>		

Figure 3-2 Security role mapping

More detailed information can be found in Chapter 6, “Securing a Web application” on page 65. Please consult also the J2EE Servlet specification for further information about this subject.

Important: If Global Security is enabled, the methods `getRemoteUsers()` and `getUserPrincipal()` return a null value even if the user is logged in, unless the servlet of the JSP itself is secured within the application server.

3.2.2 Other properties in the Global Security page

There are several other properties that can be set in the Global Security page (see Figure 3-1 on page 33). Some of them are used only if Global Security is enabled, such as User registries, Authentication and Authorization. Others, for example Enforce Java 2 Security, are not related to the enablement of Global Security. This means that those properties can be activated *and used* even if Global Security is not enabled.

- ▶ **Enforce Java 2 Security.** Specifies whether to enable or disable Java 2 security permission checking.

When the Enforce Java 2 security option is enabled and if an application requires more Java 2 security permissions than are granted in the default policy, the application might fail to run properly until the required permissions are granted in either the `app.policy` file or the `was.policy` file of the application. For applications that were not developed with the Java 2 security in mind, the simplest way is to grant full permission to all resources within the application by putting the following entry in the `was.policy` file:

```
grant codeBase "file:${application}" {  
    permission java.security.AllPermission;  
}
```

- ▶ **Enforce fine-grained JCA security.** Enable this option to restrict application access to sensitive Java Connector Architecture (JCA) mapping authentication data. Consider doing this when both of the following conditions are true:
 - a. Java 2 security is enabled.
 - b. The application code is granted the `accessRuntimeClasses` `WebSphereRuntimePermission` in the `was.policy` file found within the application enterprise archive (EAR) file.
- ▶ **Use domain-qualified user IDs.** If this option is enabled, user names will appear with their fully-qualified domain attribute when retrieved programmatically.
- ▶ **Issue permission warning.** The `filter.policy` file contains a list of permissions that an application should *not* have according to the J2EE 1.3 Specification. If

an application is installed with a permission specified in this policy file and this option is enabled, a warning will be issued.

- ▶ **Use the Federal Information Processing Standard (FIPS).** If this option is enabled, the LTPA implementation will support the FIPS-approved cryptographic algorithms for DES, Triple DES and AES.

Note: Some of the properties in the Global Security page, for example *Enforce Java 2 Security*, can be enabled even if WebSphere Global Security is not enabled.

3.2.3 Stopping the application server

While the command to start the application server is still the same when Global Security is enabled, stopping the server requires extra information. You have to specify a user ID with Administrator role rights, or the server user ID specified in the user registry and its password, in the **stopServer** command:

```
<WebSphere_home>\bin\stopServer.bat <server_name> -username <userID> -password  
<password>
```

For WebSphere Application Server running under Unix OS, the above command (the UNIX equivalent) carries a security problem. Anybody who uses the command **ps -ef** while the stopServer process is running will be able to see the user ID and the password.

To avoid the above problem, you can do the following.

1. If you are using the SOAP connection type (default) to stop the server, edit the file

`<WebSphere_home>\profiles\<profilePath>\properties\soap.client.props` and change the values of these properties:

```
com.ibm.SOAP.securityEnabled=true  
com.ibm.SOAP.loginUserId=<user ID>  
com.ibm.SOAP.loginPassword=<password>
```

Again, the user ID `<user ID>`, with its password `<password>`, is the user ID with Administrator role rights or the server user ID defined in the user registry.

2. It is recommended that you encode the `com.ibm.SOAP.loginPassword` property value using:

```
<WebSphere_home>\bin\PropFilePasswordEncoder.bat soap.client.props  
com.ibm.SOAP.loginPassword
```

Examine the result and remove the backup file created by the command above.

3. Make sure that proper file access rights for sensitive WebSphere Application Server files, for example properties files, executable files, are set.

Whether Global Security is enabled or disabled, we can now stop the WebSphere Application Server by using:

```
<WebSphere_home>\bin\stopServer.bat <server_name>
```

3.3 Disabling Global Security

There are several ways to disable Global Security. The easiest is to use the Administrative Console under **Security** → **Global security**. However, this means that the application server should already have been started. If, for some reason, the application server cannot be started, for example because the password of the server user ID in the user registry is expired, then you can disable Global Security using the command line.

1. Type:

```
<WebSphere_home>\bin\wsadmin.bat -conntype NONE
```

2. When the system command prompt redisplays, type:

```
securityoff
```

3. When done, type quit and restart the application server.

The above procedure should work without any problem. However, in case it fails, you could try to disable Global Security by directly editing the file `<WebSphere_home>\profiles\<profilePath>\config\cells\<cell_name>\security.xml`, and changing the security attribute `enabled="true"` to `enabled="false"`. Some other properties (which can be seen in Figure 3-1 on page 33), like enforcing Java 2 security, can also be found in this file. However, care should be taken when modifying this file directly.

Example 3-2 Content snippet of the file security.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<security:Security xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" ...
    xmi:id="Security_1" useLocalSecurityServer="true"
    useDomainQualifiedUserNames="false" enabled="true" cacheTimeout="600"
    issuePermissionWarning="true" activeProtocol="BOTH"
    enforceJava2Security="true" enforceFineGrainedJCASecurity="false"
    activeAuthMechanism="SWAMAuthentication_1"
    activeUserRegistry="LocalOSUserRegistry" defaultSSLSettings="SSLConfig_1">
<authMechanisms ....
    ...
    ...
</security:Security>
```

3.4 Administrative roles

As in WebSphere Application Server V5, the Administrative Console of WebSphere Application Server V6 uses the J2EE role-based authorization concept. Four roles are defined for performing administrative tasks.

Table 3-2 WebSphere Administrative roles

Role	Description
monitor	Least privileged; allows a user to view the WebSphere configuration and current state.
configurator	Monitor privilege plus the ability to change the WebSphere configuration.
operator	Monitor privilege plus the ability to change runtime state, such as starting or stopping services.
administrator	Operator and configurator privilege, plus additional privileges granted solely to the administrator rule, such as : <ul style="list-style-type: none">▶ - Modifying the server user ID and password▶ - Mapping users and groups to the administrator role

Note: The Administrative roles are effective only when Global Security is enabled.

The server user ID specified when enabling Global Security is automatically mapped to the Administrator role. Therefore, it is not necessary to manually add this identity to the administrator role.

Users and groups, as defined by the user registry, may be mapped to administrative roles. To enable a new mapping, it is necessary to save the changes to the master configuration and restart the server. For this reason, it is advisable to map groups to administrative roles so that users may be added to the groups appropriately (hence, the users are mapped to administrative roles) without the need to restart the WebSphere server.

Mapping a user to an administrative role

In order for a user to perform an administrative action, its identity must be mapped to an administrative role.

1. From the Administrative Console, select **System administration** → **Console settings** → **Console Users**.
2. Click **Add**.
3. Enter a user name in the User text box. This user must be defined in the user registry that will be active when Global Security is enabled.
4. Select the appropriate administrative role; more than one role may be selected.
5. Click **OK**. If the user cannot be found in the registry, then an error will occur.
6. Ensure the new mapping is in the Console Users list.
7. Click **Save** to save the change to the master configuration.

Note: The recently added user will be active only after the server is restarted.

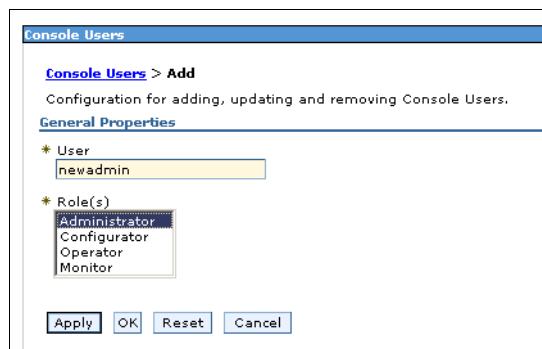


Figure 3-3 Mapping a user to an Administrative role

Mapping a group to an administrative role

As mentioned earlier, it is advisable to map groups to roles rather than users. Mapping a group is fairly similar to mapping a user.

1. From the Admin Console, click **System administration** → **Console settings** → **Console Groups**
2. Click **Add**.
3. Either a specific group or a special subject may be mapped.

To map a specific group, enter the group name in the **Specify group** text box. This group must be defined in the user registry that will be active when Global Security is enabled.

To map a special subject, select the **Special subject** option and the appropriate subject from the drop-down list. A special subject is a generalization of a particular class of users. The All Authenticated special subject means that the access check of the administrator role ensures that the user making the request has at least been authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action, as if no security were enabled.
4. Select the appropriate administrative role; more than one role may be selected.
5. Click **OK**. If the group cannot be found in the registry, then an error will occur.
6. Ensure the new mapping is in the Console Groups list.
7. Save the change to the master configuration, using the link provided at the top of the window, and restart the server.

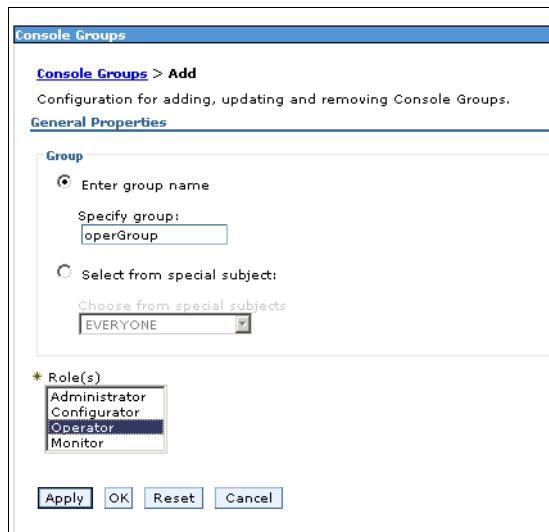


Figure 3-4 Mapping a group to an Administrative role

3.5 Naming service security: CosNaming roles

The J2EE role-based authorization concept has been extended to protect the WebSphere CORBA naming service (CosNaming) to increase the granularity of its security control. In doing so, WebSphere will be able to get a better control for client program accessing the content of the WebSphere Name space. There are generally two ways in which client programs will make a CosNaming call:

- ▶ Through the JNDI interfaces
- ▶ If CORBA clients invoking CosNaming methods directly.

In Chapter 8, “Client security” on page 153, several examples of J2EE and thin Java application clients that use the JNDI or CosNaming method call are explained. In order for some of these clients to work, at least a *CosNaming read role* to the CosNaming service should be granted to everyone (this is the default setup for WebSphere). In Table 3-3, all the four CosNaming roles are shown.

Table 3-3 CosNaming roles

Role	Description
Cos Naming Read	Users will be allowed to perform queries of the WebSphere Name Space, such as through the JNDI lookup method. The special subject Everyone is the default policy for this role.
Cos Naming Write	Users will be allowed to perform write operations such as JNDI bind, rebind, or unbind, plus CosNamingRead operations. The special subject, AllAuthenticated, is the default policy for this role.
Cos Naming Create	Users will be allowed to create new objects in the Name Space through such operations as JNDI createSubcontext, and perform CosNamingWrite operations. The special subject AllAuthenticated is the default policy for this role.
Cos Naming Delete	Users will be able to destroy objects in the Name Space, for example using the JNDI destroySubcontext method, as well as perform CosNamingCreate operations. The special subject AllAuthenticated is the default policy for this role.

Note: The CosNaming roles are effective only when Global Security is enabled.

Mapping a user or a group to a CosNaming role

The process of mapping a user/group to a CosNaming role is the same as for mapping user or group to an administrative role. You can do that by clicking **Environment** → **Naming** → **CORBA Naming Service Users** for a user's mapping and **Environment** → **Naming** → **CORBA Naming Service Groups** for a group's mapping.

Applying CosNaming security: an example

We will show a simple practical example of the use of CosNaming security. WebSphere Application Server provides a Java application client <WebSphere_home>\bin\dumpNameSpace.bat which is useful to find out all the CORBA naming services available in the server. When running the dumpNameSpace.bat Java client in a secure WebSphere environment, you will get a window similar to the following.



Figure 3-5 A window prompted by the dumpNameSpace.bat Java application client

This window is brought up when the property com.ibm.CORBA.loginSource is set to "prompt" in the CORBA client configuration file sas.client.props. You can either fill in *any* correct user ID and password defined in your user registry and click **OK**, or you can just simply click **Cancel**. With a default setup of the WebSphere Application Server, both actions should run without problems because the CosNaming read rights role is valid for everyone (see Figure 3-6).

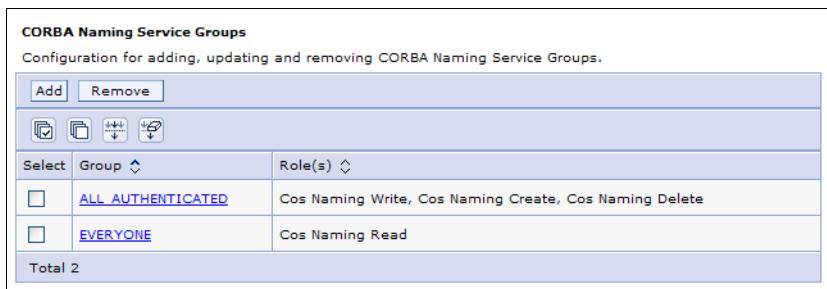


Figure 3-6 Default CosNaming security for WebSphere Application Server

Below, we will show a simple example of how to restrict the access to CORBA naming service by allowing read access only to authenticated users.

1. From the Administrative Console, click **Environment** → **Naming** → **CORBA Naming Service Groups**.
2. Remove the CosNaming read rights for the special role group EVERYONE.
3. Add the CosNaming read rights for the existing special group ALL_AUTHENTICATED.
4. Save the setup and restart the WebSphere Application Server.

The final setup for the CosNaming security should be as shown in Figure 3-7.

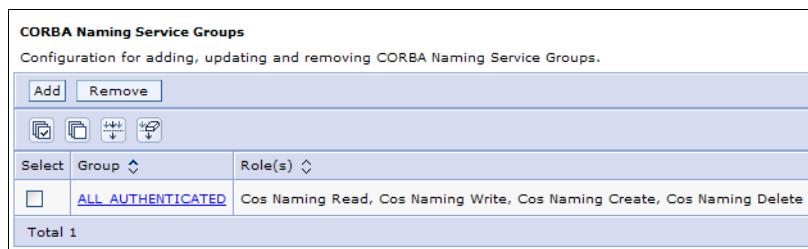


Figure 3-7 Customized CosNaming security

Once the WebSphere has been started, running the dumpNameSpace.bat Java application client will only give good results if you enter a correct user ID and password during the authentication process. Otherwise, the WebSphere Application Server will throw an exception: org.omg.CORBA.NO_PERMISSION.

Important: Unless it is needed, set the CosNaming security back to the default value as in Figure 3-6 on page 44. Failure to do so might give unexpected results for some applications that use the CORBA naming service.

3.6 SSL configurations

Transport layer secure communication uses SSL in WebSphere. SSL has to be configured for each secured transport. WebSphere can hold a set of SSL configurations and these configurations can be mapped to individual transports in the application server.

WebSphere Application Server V6 has a default SSL configuration that is brought up with the installation; it is called *DefaultSSLSettings*.

SSL entries can be managed in the Administrative Console. Select **Security** → **SSL** on the left-hand side, and the SSL configuration repertoires page is brought up on the right side.

3.6.1 Creating a new SSL entry

It is recommended that you create separate SSL configurations for the different transports in your application server. The following transports can have an SSL configuration:

- ▶ HTTP
- ▶ LDAP
- ▶ IIOP

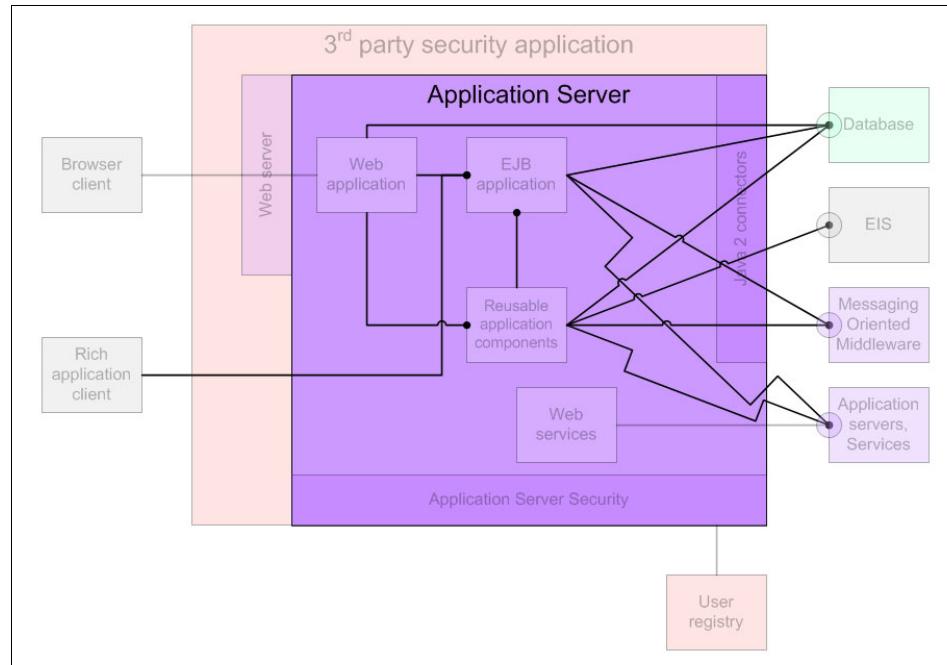
1. To create a new SSL entry, click **New JSSE Repertoire** on the SSL configuration repertoires page.
 - a. Set the Alias for the new configuration, for example: WebSSL
 - b. You can enable Client authentication to enable mutual authentication between the peers.
 - c. Set the security level, which determines the key length used for securing the message during the transport.
 - d. The Cipher suite specifies the ciphers that are accepted by the server. Make sure that there are matching cyphers listed both on the server and client sides, otherwise the communication will not work.
 - e. The cryptographic token enables and disables hardware acceleration.
 - f. Under the *provider*, you can configure the JSSE provider; this is the code that actually performs the cypher and decypher tasks.
 - g. *Protocol* defines the protocol you wish to use for securing the transport; the options are: SSL, SSLv3, SSLv2, TLS, TLSV1.
 - h. Key file and Trust file specifies the key definitions used for securing the transport. For more information about the key and trust stores, refer to *WebSphere Security Fundamentals*, REDP3944.
2. After setting the basic configurations, click **Apply**.
3. After applying the changes, you can perform further configurations.
 - a. Cryptographic token, configuration details for the hardware accelerator.
 - b. Custom properties are a set of properties for the JSSE provider. Twenty-two of these properties are preset after creating a new SSL configuration.
4. Click **OK**.
5. Save the configuration for WebSphere.

You can configure as many SSL entries as you need. Once the configurations are available, you can use them in other configurations.



Chapter 4

JAAS for authentication and authorization in WebSphere



4.1 Introduction

This chapter provides a short overview of JAAS (Java Authentication and Authorization Services). JAAS is an integral part of Java 2 security and WebSphere itself exploits both authentication and authorization services. It implements a Java version of the standard Pluggable Authentication Module (PAM) framework, and extends the access control architecture of the Java 2 platform in a compatible fashion to support user-based authorization or principal-based authorization.

4.1.1 JAAS in WebSphere

WebSphere extensions to JAAS include the following items:

- ▶ JAAS configuration can be performed from the Administrative Console. JAAS configuration is originally written in plain text files (refer to JAAS documentation). Although the plain file configuration is still available and supported in WebSphere, it is recommended that you use the Administrative Console for configuration for the following reasons:
 - Easy administration using the GUI interface.
 - Central management of configuration.
 - Distribution of configuration in Network Deployment environment.
- ▶ WSSubject (`com.ibm.websphere.security.auth.WSSubject`) is an extension to the original Subject. The WSSubject implementation can return the subject in the running thread using the `getSubject()` method inside a `doAs()` method. This is not the case with the original JAAS V1.0 implementation.
- ▶ Proxy LoginModule is responsible for loading the actual LoginModule. The reason for a proxy loader is to resolve class visibility. The proxy is an internal component, it is not going to effect application developers or administrators.

4.2 Custom JAAS login in WebSphere

JAAS provides a pluggable authentication and authorization framework in Java and for the application server as well.

This section will introduce the various components you can plug in for JAAS and use in WebSphere Application Server.

4.2.1 Callback handler

Callback handlers are responsible in JAAS for collecting the necessary information in the application to perform the authentication. The callback handler,

as its name suggests, uses the callback programming model to collect information. Together with the callback handler, there are numerous different types of callbacks defined that can be invoked. These callbacks, or just one, are invoked one after the other, for example: user ID callback to retrieve the user name, password callback to retrieve the user's password. Callbacks can be implemented in different ways; they can be interactive or non-interactive (in other words, programmatic). The interactive callbacks can interact with the user (or device) in numerous ways, for example, asking for a user ID typed in from the system console. A non-interactive, programmatic callback collects information without prompting the user, for example by reading the user ID from a properties file.

For more information about the available callback handlers in WebSphere V6, refer to "Built-in CallbackHandler in WebSphere" on page 174.

Custom callback handler

When writing your own callback handler, you can implement the different callbacks to collect information. For more information about writing your own callback handler, please refer to "Custom CallbackHandler" on page 178.

4.2.2 Login module

Login modules are responsible for the actual login, including making the authentication check and creating the principal that is stored in the subject later.

WebSphere Application Server V6 comes with a few login modules implemented for various login situations. They come registered and configured for the application server; see 4.2.4, "Configuration" on page 56.

You can find the details about how JAAS login modules work in WebSphere in 8.6.1, "JAAS login module in WebSphere" on page 171. 8.6.2, "Login process, programmatically" on page 173 shows the interaction diagram for the whole login mechanism using JAAS.

Custom login module

You can also write your own login module for WebSphere Application Server. You need to implement the LoginModule interface and code the following methods:

- ▶ public void initialize (Subject subject, CallbackHandler callbackHandler, Map sharedState, Map options) - this method is responsible for initializing the login module after loading.

- ▶ public boolean login() throws LoginException - this method performs the actual login. This is the part where you can code the authentication check using callbacks.
- ▶ public boolean commit() throws LoginException - after a successful login, you need to commit the login. This is the part where you can insert the new subject into the security context.
- ▶ public boolean abort() throws LoginException - in case of any problem, this method aborts the login process.
- ▶ public boolean logout() throws LoginException - after a successful login and by the end of the session, the application needs to log out and remove the subject from the security context.

A very simple custom login module is shown in Example 4-1.

Example 4-1 Custom login module: CustomLoginModule.java

```
package redbook.sg246316.loginmodule;

import java.util.*;
import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;
import javax.security.auth.spi.*;
import redbook.sg246316.loginmodule.SamplePrincipal;

public class CustomLoginModule implements LoginModule {

    // initial state
    private Subject subject;
    private CallbackHandler callbackHandler;
    private Map sharedState;
    private Map options;
    // the authentication status
    private boolean succeeded = false;
    private boolean commitSucceeded = false;
    // username and password
    private String username;
    private String password;
    // testUser's SamplePrincipal
    private SamplePrincipal userPrincipal;

    public void initialize(Subject subject, CallbackHandler callbackHandler,
        Map sharedState, Map options) {
        this.subject = subject;
        this.callbackHandler = callbackHandler;
        this.sharedState = sharedState;
        this.options = options;
    }

    public boolean login() throws LoginException {
        // do the authentication here
        // ...
        return true;
    }

    public boolean commit() throws LoginException {
        // do the commit here
        // ...
        return true;
    }

    public boolean abort() throws LoginException {
        // do the abort here
        // ...
        return true;
    }

    public boolean logout() throws LoginException {
        // do the logout here
        // ...
        return true;
    }
}
```

```

    }

    public boolean login() throws LoginException {
        // prompt for a user name and password
        if (callbackHandler == null) throw new LoginException("Error: no
CallbackHandler available!");

        Callback[] callbacks = new Callback[2];
        callbacks[0] = new NameCallback("user name: ");
        callbacks[1] = new PasswordCallback("password: ", false);

        try {
            callbackHandler.handle(callbacks);
            username = ((NameCallback) callbacks[0]).getName();
            password = new String(((PasswordCallback)
callbacks[1]).getPassword());
            ((PasswordCallback) callbacks[1]).clearPassword();

        } catch (java.io.IOException ioe) {
            throw new LoginException(ioe.toString());
        } catch (UnsupportedCallbackException uce) {
            throw new LoginException("Error: " + uce.getCallback().toString());
        }

        // verify the username/password
        // this code is using hard-coded user name and password for the sake of
simplicity
        boolean usernameCorrect = false;
        boolean passwordCorrect = false;
        if (username.equals("testUser")) usernameCorrect = true;
        if (usernameCorrect && password.equals("testPassword")) {
            // authentication succeeded!!!
            passwordCorrect = true;
            succeeded = true;
            return true;
        } else {
            // authentication failed
            succeeded = false;
            username = null;
            password = null;
            throw new FailedLoginException("Authentication failed!");
        }
    }

    public boolean commit() throws LoginException {
        if (succeeded == false) {
            return false;
        } else {
            // add a Principal (authenticated identity) to the Subject

```

```

        // this is a custom principal: SamplePrincipal
        // in WebSphere you may want to use the WSPrincipalImpl class
        userPrincipal = new SamplePrincipal(username);
        if (!subject.getPrincipals().contains(userPrincipal))
            subject.getPrincipals().add(userPrincipal);
        username = null;
        password = null;
        commitSucceeded = true;
        return true;
    }
}

public boolean abort() throws LoginException {
    if (succeeded == false) {
        return false;
    } else if (succeeded == true && commitSucceeded == false) {
        // login succeeded but overall authentication failed
        succeeded = false;
        username = null;
        password = null;
        userPrincipal = null;
    } else {
        // overall authentication succeeded and commit succeeded, but someone
        else's commit failed
        logout();
    }
    return true;
}

public boolean logout() throws LoginException {
    subject.getPrincipals().remove(userPrincipal);
    succeeded = false;
    succeeded = commitSucceeded;
    username = null;
    password = null;
    userPrincipal = null;
    return true;
}

```

You will find that the custom login module uses a custom principal (SamplePrincipal). For more information about the principals, refer to the next section.

The login module basically performs the following steps:

1. Initializes the login module, instantiates the necessary objects.
2. Sets up the callback handler and the callback methods.

3. Walks through the callbacks one after the other.
4. Authenticates the user using the information returned from the callbacks.
 - a. If authentication is successful, it creates a principal based on the authentication data and inserts it into the subject setup during initialization.
 - b. If authentication fails, the module destroys the objects and returns with a failed flag.
5. In the meantime, the login process can be aborted.
6. After a successful login, the application can also log out. The logout method should take care of removing the principal from the subject and destroying the objects in the login module.

4.2.3 Principal

Principals in JAAS are objects storing user credentials. Principals can then be added (stored) in subjects, which is another object in JAAS to store multiple (or just one) principals. The subject then is propagated with the security context, and is available for the application server to check the logged-in principals.

WebSphere itself has its own implementation of a principal, `WSPrincipal`. This principal is used internally with the security context.

Custom principal

Principals can be customized and new ones can be implemented based on the `java.security.Principal` interface. Example 4-2 shows an implementation, `SamplePrincipal`, of a principal. You can customize the principals to store extra information about the user, other than just the user name.

Example 4-2 Custom principal: SamplePrincipal.java

```
package redbook.sg246316.loginmodule;

import java.security.Principal;

public class SamplePrincipal implements Principal, java.io.Serializable {
    private String name;

    public SamplePrincipal(String name) {
        if (name == null) throw new NullPointerException("illegal null input");
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

```

public String toString() {
    return ("SamplePrincipal: " + name);
}

public boolean equals(Object o) {
    if (o == null)
        return false;
    if (this == o)
        return true;
    if (!(o instanceof SamplePrincipal))
        return false;

    SamplePrincipal that = (SamplePrincipal) o;
    if (this.getName().equals(that.getName()))
        return true;
    return false;
}

public int hashCode() {
    return name.hashCode();
}
}

```

As you can see, the principal is a simple Java object with attributes, a collection of set and getter methods, and a few other supporting methods.

4.2.4 Configuration

Take the compiled code for the custom login module and the dependent classes and package them in a JAR file. You can also deploy the code unbundled, in directories and files, but packaging it into a JAR file is recommended.

You can place the custom login module code in the following places:

- ▶ Within an .ear file for a specific enterprise application; then, it is only accessible to the specific application.
- ▶ In the WebSphere Application Server shared library.
- ▶ In the Java extensions directory (<WebSphere_root>\jre\lib\ext), it is available to all applications.

In the WebSphere Administrative Console, you can configure JAAS login modules under the following link: **Security** → **Global Security** → **Authentication** → **JAAS Configuration**. Here you will find three items; the first two are related to LoginModule configuration.

Application logins

Application logins are the ones that your enterprise applications can use. After installation, you will find three items already defined.

- ▶ ClientContainer
com.ibm.ws.security.common.auth.module.WSClientLoginModuleImpl
- ▶ DefaultPrincipalMapping
com.ibm.ws.security.auth.j2c.WSPrincipalMappingLoginModule
- ▶ WSLogin com.ibm.ws.security.common.auth.module.WSLoginModuleImpl

You should not remove or modify these definitions; some applications may use them and those applications may break if you change any of them.

You can add a new application JAAS login module configuration to the list.

1. Under Application login configuration, click **New**.
2. Provide an alias name, for example: MyLoginModule.
3. Click **Apply**. Do not click **OK** yet, we are going to define the login module first before we save the configuration.
4. Click **JAAS login modules**.
5. Click **New** in the new window.
6. Provide the fully qualified name (including package name) for your custom LoginModule implementation in the Module class name field, for example: com.ibm.itso.MyLoginModuleImpl.

Check the **Use login module proxy** check box, to ensure the class visibility for applications. For more information about the login module proxy, refer to the WebSphere InfoCenter.

Select the authentication strategy, set as REQUIRED for now. The options include: REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL. For more information about the different strategies, refer to the WebSphere InfoCenter.

7. Click **OK**.
8. Save the configuration for WebSphere.

System logins

System login definitions are very similar to the application login definitions, except that they are related to the application server itself, not the applications. These definitions are used for internal login purposes, for example: LTPA, SWAM, RMI, Web.

You can write your own login modules and use them internally, but it is not recommended that you remove or change the existing ones under the System login configuration.

4.2.5 Programming authentication

You can customize the whole authentication process in WebSphere for various situations. You can customize and plug in any or all of the components of the JAAS login. WebSphere comes with pre-defined and pre-configured components for every parts of the login mechanism. If you need custom behavior in your application, then you need to plug in your own implementation.

You can find the details of programmatic login using JAAS in 8.6.2, “Login process, programmatically” on page 173.

4.2.6 Programming authorization

Authentication is not the only part of JAAS. JAAS also can help with authorization to run (or not to run) specific code under a specific subject.

In the application code, in any component, you can perform a custom login using the login module, as described before.

1. After the login, you can authorize the subject to run a piece of code.
2. Instantiate the class `java.security.PrivilegedAction` and pass a specific method description as a parameter. The method signature is the following:

```
public Object run()
```

The method body executes the application code.

3. After the login and after instantiating the action, you can call the `doAs()` method on the subject returned from the login module. The `doAs()` method requires the subject and the instantiated action class passed as a parameter.

For more details and a sample implementation of authorized code, refer to 8.6.2, “Login process, programmatically” on page 173.

4.3 J2C Authentication data

The J2C Authentication data entries configure the access to external resources, for example: database, messaging oriented middleware and other J2C adapters.

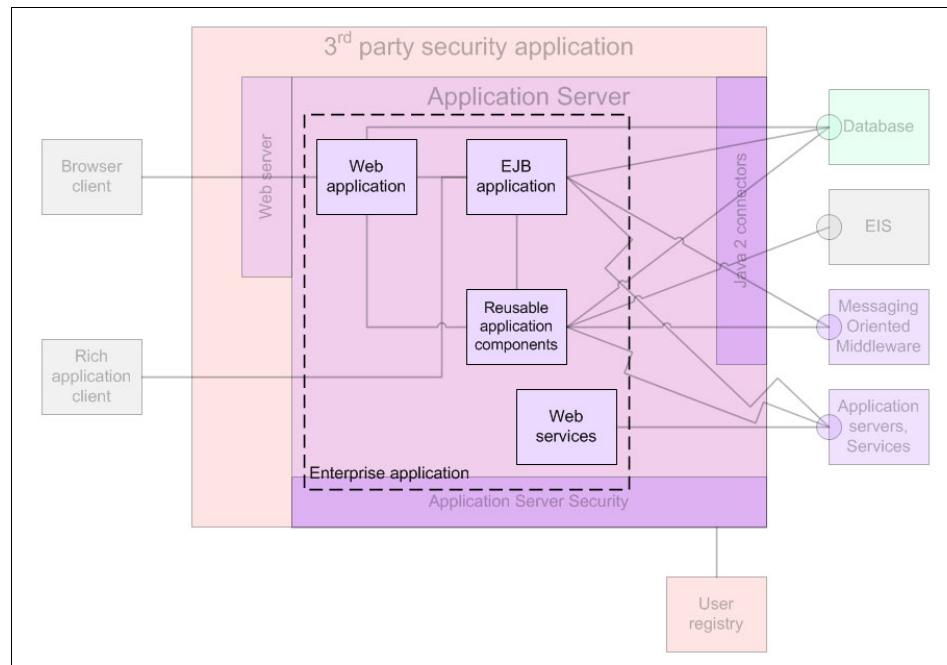
Why do we need the J2C Authentication data entries? How are they used?

When the application accesses an external resource, it happens through a J2C adapter. The application doesn't authenticate itself for the external resources by providing an user ID and password. The user ID and password for the J2C adapter is defined under the J2C Authentication data. When the J2C adapter requests a login, it uses a JAAS login module in WebSphere. The login module simply looks up the J2C Authentication data entry from the WebSphere configuration during the login process and uses the retrieved data with the callback methods.

You will find JAAS authentication entries under the Administrative Console at **Security** → **Global security** → **JAAS Configuration** → **J2C Authentication data**. You can create new entries or remove entries here.



Enterprise application security



5.1 Deploying a secured application

Deploying a secured application is hardly different than deploying any other (non-secured) enterprise application. The only difference is that during deployment, you can perform the role mapping for users and groups, as well as the run-as mapping.

5.1.1 Role mapping during application installation

During the process of running the application installation, you will get to a step with the title: *Map security roles to users/groups*. At this step, you have the option of selecting any of the roles and assign a user or a group from the user registry using one of the lookups. You can also assign one of the special subjects (*Everyone* or *All authenticated*) to the role.

If you have EJBs or EJB methods with Run-As mapping, then you get to the step: *Map RunAs roles to users*. At this step, you can specifically assign a user name and password (an identity) to a Run-As (delegation) definition.

If you have EJB methods without security assignments, then you get to the step: *Ensure all unprotected 2.x methods have the correct level of protection*. At this step, you can assign a role to these methods, on a per EJB basis (not on a per method basis). You can also exclude the methods so they cannot be accessed; or you can uncheck them so they can be accessed by everyone.

The above mentioned three types of mappings might be already defined in the enterprise archive. These can be defined during assembly time, just before deployment, for example in the Rational Application Developer or in the Application Server Toolkit. Even if the mappings have been done previously, you can review and modify them during deployment or even later, as you will find in the next section.

5.1.2 Role mapping after installation

Once the application is installed, you can still change the security settings for the application.

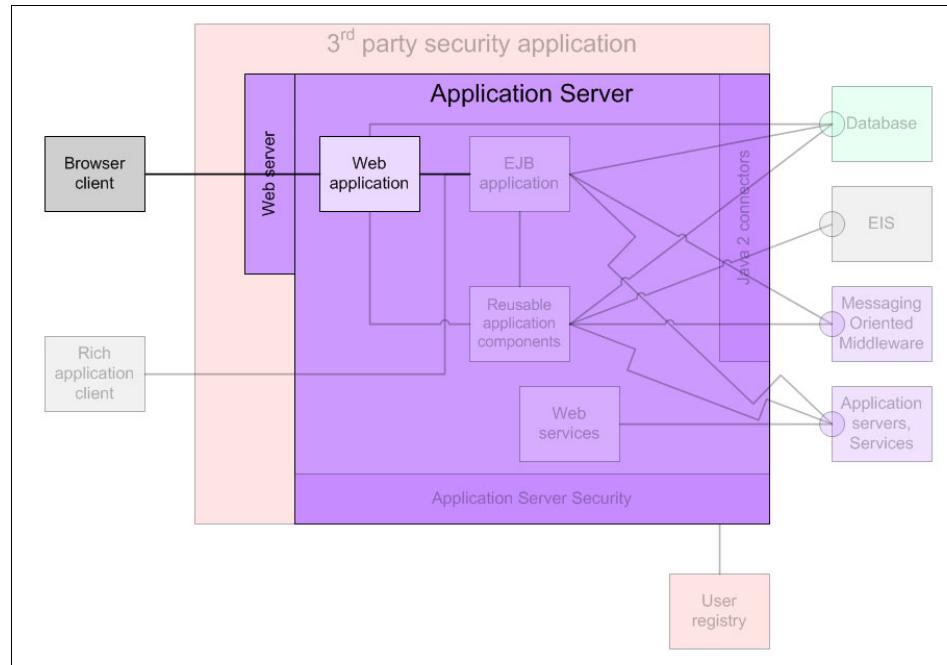
1. Launch Administrative Console and log in.
2. Click **Applications** → **Enterprise Applications** from the menu.
3. Select the application that you want to change.
4. You will find the following items under the Additional Properties section:
 - Map security roles to users/groups
 - Map RunAs roles to users

Note: The *Ensure all unprotected 2.x methods have the correct level of protection* configuration is not available after deployment. Once the methods are defined as unchecked, excluded or mapped to a role, this does not change.

By selecting any of these, you will get to the same configuration page as the one you saw during deployment.

5. Make sure that you save the configuration for WebSphere after the changes.
6. Make sure you restart the Enterprise Application after you made and saved the changes.

Securing a Web application



6.1 Introduction

This chapter will discuss the security aspects involved with securing Web applications. We will discuss how to secure the transport channels between all components and what are the authentication and authorization options available at the each processing component.

A Web application consists of different Web components, such as HTML pages, JSPs and servlets; all these form Web pages on the server side. On the client side, a Web browser is usually used to issue a request for a Web resource. The request goes to the application server. WebSphere Application Server then processes all the Web components which form the requested Web resource, creates a Web page and sends it back as the response. The browser transforms responded Web page in more human readable format and presents it on the screen.

Transport channel

Transport channel refers to the communication channel between Web client and Web application server. The communication can be classified into different layers, each has its own functions and scope. Here we will focus on the topmost layer of communication between Web clients and Web application servers.

So-called protocols define different communication types and HTTP (Hyper Text Transfer Protocol) protocol is used for as application communication protocol between Web clients and Web application servers. Using purely HTTP, the data flow is not encrypted, so anybody who could intercept it would understand the content. For securing the transport channel, we need secured HTTP, which is called HTTPS. Usually, HTTP runs on top of the TCP transport protocol and to secure it, we need SSL (Secure Socket Layer). In summary, for encrypted communication between Web clients (browsers) and Web application servers (WebSphere Application Server) we use HTTPS which runs on top of SSL secured TCP transport channel.

6.2 Securing the static content

Static Web resources are those whose content does not change over time no matter which user accesses them or what the user input data is; for example, this could be a static .html page or a .jpg image file. Although WebSphere Application Server provides a mechanism to serve them, we recommend that those resources be served by the Web server. When the Web server is involved, WebSphere does not have security control over the resourced served by the Web server; thus transport security, authentication and authorization must be configured for the Web server. In this section, we will describe how to secure static content which is served by the Web server only. For securing static content

served by WebSphere, please see “Securing content served by WebSphere Application Server” on page 87.

In 6.2.2, “Authentication with Web server” on page 69, we provide an example of how to configure IBM HTTP Server to secure static content with HTTP basic authentication when the user registry is set to an LDAP directory. In 6.2.3, “Authorization with the Web server” on page 72, we explain how access to this static content can be managed using the .htaccess configuration files.

Describing all the possible options for managing security in IBM HTTP Server is not within the scope of this book. For detailed information, see the product documentation for the appropriate release.

Additional products may also be used to provide the end-to-end security infrastructure. For information about how Tivoli Access Manager fits into this scenario, see Chapter 10, “Securing a WebSphere application using Tivoli Access Manager” on page 211.

6.2.1 Securing transport channel between Web browser and Web server

The Web browser and Web server communicate with each other over the HTTP protocol. By default, HTTP is not secured at all. To assure the data integrity, we must use the SSL protocol with the HTTP protocol to secure the transport.

IBM HTTP Server uses IBM’s proprietary SSL module and SSL configuration described in this section. If you use another Web server, please refer to the product documentation to see how to set up an SSL transport channel.

As a starter for this section, we created a key store of CMS format that contains a self-signed digital certificate to secure the HTTP transport channel between the Web browser and the IBM HTTP Server. For more information about how to create a key database file which stores necessary certificates, refer to the IBM Redpaper *Security Basics for WebSphere*, REDP-3944.

Configuring IBM HTTP Server for SSL

1. Open the httpd.conf file, which is the configuration file for IBM HTTP Server; you will find the file under the <IHS_root>\conf directory.
2. Add the *ibm_ssl_module* definition to the end of the *LoadModule* list as in Example 6-1 on page 68.

Example 6-1 Adding ibm_ssl_module definition to httpd.conf

```
LoadModule alias_module modules/mod_alias.so
#LoadModule rewrite_module modules/mod_rewrite.so
#LoadModule deflate_module modules/mod_deflate.so
LoadModule ibm_ssl_module modules/mod_ibm_ssl.so
```

3. Create a virtual host, then enable and configure SSL just for this virtual host. On a global level, the IBM HTTP server will still not use SSL. Add the directives to the httpd.conf as in Example 6-2.

Example 6-2 Adding a virtual host definition and configuring it for SSL

```
SSLDisable
Listen 443

<VirtualHost webserver01.redbook.net:443>
SSLEnable
KeyFile "C:/IBMHTTPServer/conf/keys/IHS6Certificates.kdb"
</VirtualHost>
```

This is the most basic SSL setup, but there are other SSL directives that you can use to set the SSL configuration more specifically to your needs. Further explanation of those is not within the scope of this redbook; please refer to the IBM HTTP Server documentation. The directives that we used for configuration are explained here:

- The directive *Listen 443*, which is placed into global definition scope, makes our IBM HTTP Server listen on port 443 as well.
 - The directive *VirtualHost* starts our virtual host stanza. Make sure you specify a TCP resolvable hostname or IP there.
 - The directive *SSLEnable* enables SSL for this virtual host only.
 - The directive *KeyFile* defines where the key database file is located.
4. Save the httpd.conf configuration file and restart IBM HTTP Server.

Testing SSL between Web browser and Web server

Open a Web browser and connect to the Web server using
`https://<virtualhostname>` where `<virtualhostname>` is the TCP domain resolvable name that you used with your virtual host definition. In our case, we enter the following into the browser's address bar:

`https://webserver01.redbook.net/`

Since we did not specify any port with the request, the request will go to the default HTTPS server listening port, which is 443. The Web server will recognize the request and, since it comes to port 443, bind it to the configured virtual host.

Because SSL is enabled only for the virtual host, we can still access the Web server unsecured by HTTP on port 80 which is defined on the global scope. If you want an SSL-only configuration then specify SSL directives on the global scope without creating a virtual host, like in Example 6-3.

Example 6-3 Configuring SSL on global configuration scope

```
Listen 443
SSLEnable
KeyFile "C:/IBMHTTPServer/conf/keys/IHS6Certificates.kdb"
```

Also delete the default *Listen 80* directive just to limit port 443 to be the only listening port when using an SSL-only configuration.

6.2.2 Authentication with Web server

Most Web servers are able to secure the files that they serve. For example, IBM HTTP Server can protect its own resources, in the following ways:

- ▶ **HTTP basic authentication**

With IBM HTTP Server, different user registry modules are provided for using for authentication, including: simple text file, LDAP directory or database.

In 6.5.2, “Configuring LDAP authentication with IBM HTTP Server” on page 98 you can find how to set IBM HTTP Server for using Basic authentication with the LDAP directory as the user registry.

In “Configuring Basic Authentication for the Web server” on page 69 you can find how to set IBM HTTP Server to use Basic authentication with the text file user registry.

For more details about HTTP Basic Authentication, please see the protocol definition document at <http://www.ietf.org/rfc/rfc2617.txt>.

- ▶ **HTTP digest authentication**
- ▶ **Digital client certificate authentication using SSL**

Configuring Basic Authentication for the Web server

In this section, we will present a simple scenario of how to implement basic authentication for the Web server when user registry is stored in a simple text file.

For this example scenario, we will enable security for all the static Web components in the C:\IBMHttpServer\htdocs\en_us directory.

Creating the user registry text file

First we need to create a simple text file which stores the user registry information. For this purpose, we need to use the htpasswd utility that comes with IBM HTTP Server.

In Example 6-4, see how to run the htpasswd utility. In our case, we created a new user registry file named users and added the ITS0user user.

Example 6-4 Creating user registry text file

```
C:\IBM HTTP Server\conf>..\bin\htpasswd -c users ITS0user
Automatically using MD5 format.
New password: *****
Re-type new password: *****
Adding password for user ITS0user

C:\IBM HTTP Server\conf>
```

Configuring the Web Server to use Basic authentication

1. Open the httpd.conf file, which is the configuration file for IBM HTTP Server.
2. Make sure that *auth_module* module definition in the *LoadModule* list is uncommented, like in Example 6-5.

Example 6-5 auth_digest module definition

```
#LoadModule auth_module modules/mod_auth.so
#LoadModule auth_anon_module modules/mod_auth_anon.so
#LoadModule auth_dbm_module modules/mod_auth_dbm.so
LoadModule auth_digest_module modules/mod_auth_digest.so
```

3. In previous sections, we created a virtual host definition and defined SSL transport just for that scope. Now we are going to enable SSL on the global level and add Basic authentication.

Add the *Directory* directive to protect the C:\IBMHttpServer\htdocs\en_us directory. This directory is set as a global Web server root so when we call the Web server just using hostname, the Web server is going to search for the index.html file under the Web server root.

Within *Directory* we will specify additional security directives that will be effective just on that scope; see Example 6-6.

Example 6-6 Configuring HTTP basic authentication with text file user registry

```
#Listen 80
Listen 443
SSLEnable
KeyFile "C:/IBM HTTP Server/conf/keys/IHS6Certificates.kdb"
```

```
<Directory "C:/IBM HTTP Server/htdocs/en_US">
AuthType Basic
AuthName "Restricted Directory"
AuthUserFile "C:/IBM HTTP Server/conf/users"
Require valid-user
Options None
AllowOverride None
</Directory>
```

-
4. Save httpd.conf and restart the Web server.

Testing the Basic authentication configuration with the Web server

1. Open a new browser window on the Web server machine.
2. In the address bar, enter https://localhost. If you are using a browser on a separate machine, provide the proper server name in the URL.
3. First SSL connection will be established and then you will get an authentication pop-up window. Enter ITS0user for the User Name and the corresponding password; see Figure 6-1.
4. Click **OK** and you should get the content (index.html) served.

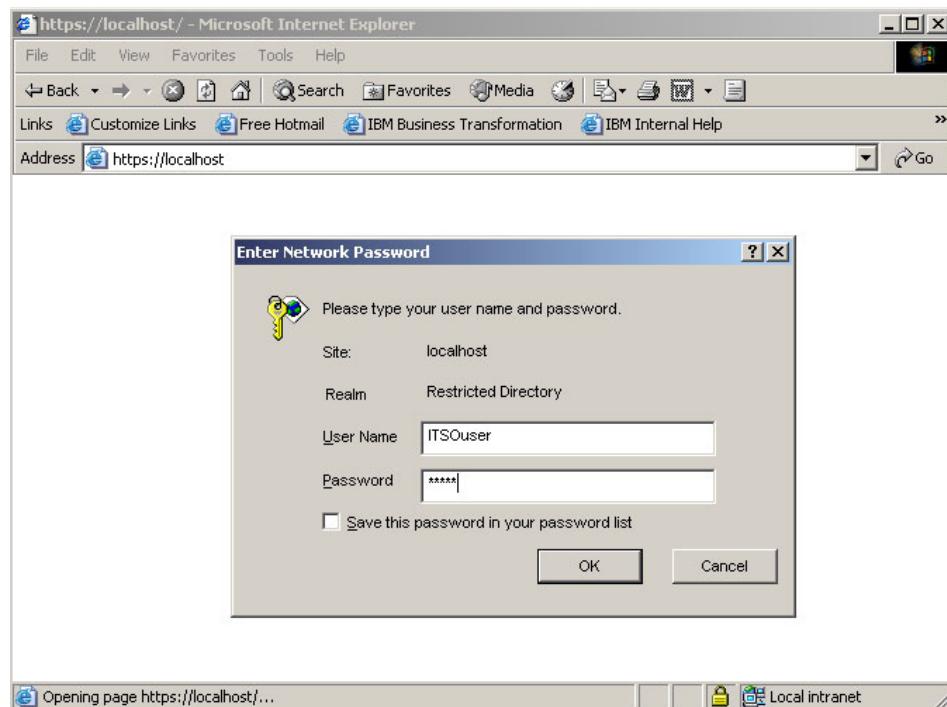


Figure 6-1 Testing HTTP Basic authentication with enabled SSL

6.2.3 Authorization with the Web server

By default, the Web server configuration and access control directives are handled by the Web server administrator by modifying the httpd.conf file. The appropriate section of the file enforces these settings.

Example 6-7 Enforcing access control management by settings in httpd.conf file

```
<Directory "C:/IBMHttpServer/htdocs/en_US">
    AllowOverride None
    Options None
</Directory>
```

The directive *AllowOverride None* tells the Web server not to look for any other access control definition files within the given directory scope. In a default httpd.conf configuration file shipped with IBM HTTP Server, this directive is included in every <Directory> container.

However, in many cases this is a limiting factor and may require an administrator's intervention in case of simple changes to the file. Secondly, you might want to give to an individual user or group of people the possibility to configure their own area of the Web site. This is not possible with the default httpd.conf settings.

If there is a need to set an access control on a per-directory basis, overriding the settings in httpd.conf file, IBM HTTP Server uses *.htaccess* files for every directory over which the user wants to have such control. Changes done to any *.htaccess* file do not require restarting the Web server or any other administrator intervention since the file is read every time a resource is fetched from the directory.

A *.htaccess* file placed in one directory applies to all its subdirectories. If there is more than one access file in a directory tree, the directives set in a file for the subdirectory take precedence over the directives in the parent directory.

The drawback of using *.htaccess* files is a negative impact on the performance of the Web server. The other problem with the *.htaccess* files is the system management. It is difficult to maintain, especially in a centralized security infrastructure.

For more information about how to use *.htaccess*, see the Apache tutorial at:
<http://apache-server.com/tutorials/ATusing-htaccess.html>

6.3 Securing the Web server plugin for WebSphere

This section focuses on securing the WebSphere Application Server HTTP plugin. Although it runs as a part of Web server's process, detached from WebSphere, it is an integral part of the application server and security.

6.3.1 Securing the transport channel between the Web server and WebSphere

This section documents the configuration necessary to instantiate a secure connection between the Web server plug-in and the embedded HTTP server in the WebSphere Web container. By default, this connection is not secure, even when Global Security is enabled. The documentation will cover the configuration for IBM HTTP Server, but the Web server related configuration in this situation is not specific to any Web server.

Setting the authentication mechanism as client-certificate

The following steps are mandatory for generating the certificates for SSL communication between the two peers.

1. Create a self-signed certificate for the Web server HTTP plug-in.
2. Create a self-signed certificate for the WebSphere Web Container.
3. Exchange the public keys between the two peers.
4. Modify the Web server plugin-cfg.xml file to use SSL/HTTPS
5. Create a new SSL repertoire (or modify an existing one)
6. Modify the WebSphere embedded HTTP Server (Web Container) to use SSL/HTTPS.

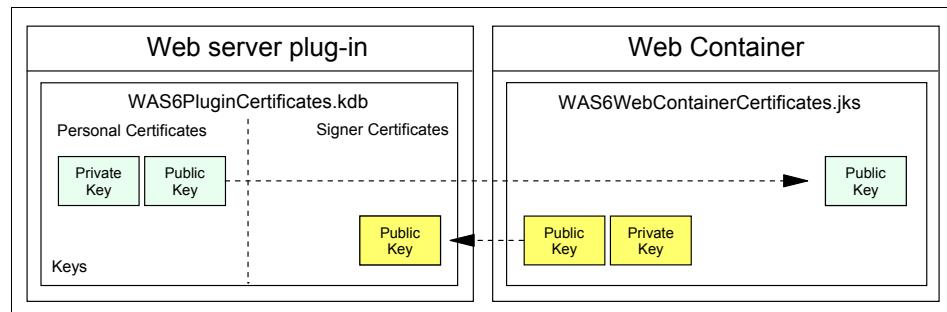


Figure 6-2 Certificates

Figure 6-2 on page 73 illustrates the exchange of the public certificate keys associated with each peer participating in the secure SSL communication.

Creating a self-signed certificate for the Web server HTTP plug-in

Create a CMS type key store, for example: WAS6PluginCertificates.kdb under the C:\IBM Http Server\conf\keys\ directory. Create a self-signed certificate.

Tip: For more details about creating and extracting a self-signed certificate, see the IBM Redpaper *Security Basics for WebSphere*, REDP3944.

Creating a self-signed certificate for the WebSphere Web Container

Create a JKS type key store, for example: WAS6WebContainerCertificates.jks under the C:\WebSphere\Appserver\etc directory. Create a self-signed certificate.

Tip: For more details about creating and extracting a self-signed certificate, see the IBM Redpaper *Security Basics for WebSphere*, REDP3944.

Exchanging the public keys between the two peers

Exchange the public certificates from the self-signed certificated between the two key stores: WAS6PluginCertificates.kdb, WAS6WebContainerCertificates.jks.

Tip: For more details about exchanging certificates, see the IBM Redpaper *Security Basics for WebSphere*, REDP3944.

Creating a new SSL entry

Within the WebSphere configuration, an SSL entry represents a set of SSL properties which can be used with different WebSphere resources. For this section, we created a new SSL entry.

1. Start the WebSphere Administration Console, then after login, select **Security → SSL**.
2. Click **New JSSE Repertoire** to create a SSL repertoire. Provide the following values to fill out the form:
 - Alias: Web Container SSL
 - Key File Name:
C:\WebSphere\Appserver\etc\WAS6WebContainerCertificates.jks

- Key File Password: passw0rd
- Key File Format: JKS
- Trust File Name:
c:\WebSphere\Appserver\etc\WAS6WebContainerCertificates.jks
- Trust File Password: passw0rd
- Trust File Format: JKS
- Client Authentication: not selected
- Security Level: HIGH

Leave other options as default and click **OK** when you have finished.

The client authentication option is to set whether the Web container also expects to get a client certificate either trusted by a well-known CA, or self-signed with the imported public key. If we enable this option for an SSL entry which will be used for HTTP plugin to Web container transport protection, then we also need a client certificate in the HTTP plugin key database and the Web container must be able to recognize that certificate.

Per above, we left this option as unchecked; if you need client authentication within this scope, refer to 6.5.3, “Configuring SSL certificate based client authentication method for IBM HTTP Server” on page 103 to see how to set a client certificate onto a CMS key database needed for the HTTP plugin and also refer to 6.5.4, “Configuring SSL certificate based client authentication method for WebSphere Application Server” on page 107 to see how to set the SSL entry and Web container configuration for client authentication.

Note: If you want mutual SSL between the two parties, select the **Client Authentication** check-box.

3. Save the configuration for WebSphere.

Modifying the Web Container configuration to support SSL

To complete the configuration between Web server HTTP plug-in and Web Container, the WebSphere Web Container must be modified to use the previously created self-signed certificates.

The following steps document the required Web Container modifications.

1. Select **Servers** → **Application Servers**, then click the server you want to work with, in this case: **server1**.
2. In the Configuration tab, go to **Container Settings** section and click **Web container transport chain**.

3. In Figure 6-3 you can see that a default secured transport chain called *WCInboundDefaultSecure* is already defined. Web Container listens on the TCP port 9443 for this chain. We will modify this chain to use it in this section. We could also create another transport chain and configure it as follows.

Select	Name	Enabled	Host	Port	SSL Enabled
<input type="checkbox"/>	WCInboundAdmin	Enabled	*	9060	Disabled
<input type="checkbox"/>	WCInboundAdminSecure	Enabled	*	9043	Enabled
<input type="checkbox"/>	WCInboundDefault	Enabled	*	9080	Disabled
<input type="checkbox"/>	WCInboundDefaultSecure	Enabled	*	9443	Enabled

Total 4

Figure 6-3 Default transport chains for server1

4. Click **WCInboundDefaultSecure** (or on the new transport channel that you might have been created). Make sure that the **Enabled** check box is selected.
5. Click **SSL Inbound Channel (SSL 2)** then go to the SSL repertoire section and select **Web Container SSL** from the list. Thus we specify a previously created SSL repertoire to be used with this transport channel, see Figure 6-4.
6. Click **OK**, then save the configuration for WebSphere.
7. Regenerate and propagate the Web server plug-in.

General Properties		Additional Properties
* Transport Channel Name <input type="text" value="SSL_2"/>		■ Custom Properties
Discrimination weight <input type="text" value="1"/>		■ Related Items ■ SSL configuration repertoire - cell level
SSL repertoire <input type="text" value="itsOnode01/Web Container SSL"/>		
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>		

Figure 6-4 SSL Inbound Channel properties

Note: The application server must be restarted in order for the Web Container configuration changes to become effective.

Modifying the Web server plug-in file

The plug-in config file must be modified to reference the plug-in keyring and the password stash file. This allows the transport protocol to be changed from HTTP to HTTPS, using the certificates stored in the keyring.

With WebSphere Application Server V6, more than one Web Server definition is possible and each has its own set of HTTP plug-in properties. Also by default, there is a separate installation directory of HTTP plug-in binaries and within this directory there are separate plug-in configuration directories for each Web Server/plug-in definition. For this section we will use the Web Server definition, named webserver1; in our case its configuration is under C:\Websphere\Plugins\config\webserver1.

Tip: If you are unsure which HTTP plugin configuration file is the right one, open the Web server's configuration file and check which file name is used when defining HTTP plugin module configuration.

Follow the steps below.

1. Open the HTTP plugin configuration file, plugin-cfg.xml, for editing. A standard non-secure HTTP connection in the configuration looks like this:

```
<Transport Hostname="w2ksrvvm01.itso.ibm.com" Port="9080" Protocol="http" />
```

By default, there is also secured connection entry, which looks like this:

```
<Transport Hostname="w2ksrvvm01.itso.ibm.com" Port="9443" Protocol="https">
  <Property name="keyring"
    value="C:\WebSphere\Plugins\etc\WASplugin.kdb"/>
  <Property name="stashfile"
    value="C:\WebSphere\Plugins\etc\WASplugin.sth"/>
</Transport>
```

Note that the w2ksrvvm01.itso.ibm.com is the hostname for the application server.

2. Comment out or delete the non-secure HTTP part as follows:

```
<!-- <Transport Hostname=w2ksrvvm01.itso.ibm.com" Port="9080"
Protocol="http" /> -->
```

3. Modify the default secure sections and specify your key database instead of the default:

```
<Transport Hostname="w2ksrvvm01.itso.ibm.com" Port="9443" Protocol="https">
  <Property name="keyring"
    value="C:\WebSphere\Plugins\etc\WAS6PluginCertificates.kdb"/>
  <Property name="stashfile"
    value="C:\WebSphere\Plugins\etc\WAS6PluginCertificates.sth"/>
</Transport>
```

4. Set the log level to Trace.

```
<Log LogLevel="Trace"
  Name="C:\WebSphere\Plugins\logs\webserver1/http_plugin.log"/>
```

5. Save the plugin-cfg.xml file and restart the Web server.

Testing the secure connection

For this section, we assume that your Web server is not SSL-enabled, and that the connection between the browser and Web server is plain HTTP. However, the behavior should not be different if you use an SSL-enabled Web server. After setting the HTTP plugin and restarting the Web server, take the following steps:

1. Open a new browser window. Make sure that the Web server, WebSphere and the default application are running.
2. Enter the following URL to access the Snoop servlet:
`http://webserver01.redbook.net/snoop`, where webserver01.redbook.net is the hostname for the Web server.
3. Since security is enabled in WebSphere, you will get a Basic authentication pop-up window. Enter a valid user name and password and you will get the Snoop servlet output.

Because we commented out the HTTP transport in the plugin configuration, leaving only the HTTPS option available, we left no choice to the plugin, so it must use HTTPS transport to connect to WebSphere.

We called Snoop via the Web server using plain HTTP; the Web server passed the request to the HTTP plugin and the plugin contacted WebSphere through HTTPS.

Open the plugin trace file, `http_plugin.log`, and you will find something similar to what is shown in Example 6-8. The SSL connection has been used to connect to WebSphere. Furthermore, it shows that WebSphere issued the basic authentication challenge before serving the Snoop servlet response.

Example 6-8 HTTP Plugin trace showing SSL connection

```
ws_common: websphereFindTransport: Finding the transport
ws_common: websphereFindTransport: Setting the transport(case 4):
w2ksrvvm01.itso.ibm.com on port 9443
ws_common: websphereExecute: Executing the transaction with the app server
```

```
ws_common: websphereGetStream: Getting the stream to the app server
ws_transport: transportStreamDequeue: Checking for existing stream from the
queue
ws_common: websphereGetStream: socket 7792 connected to
w2ksrvvm01.itso.ibm.com:9443
lib_stream: openStream: Opening the stream
lib_stream: openStream: Stream is SSL
ws_common: websphereGetStream: Created a new stream; queue was empty, socket =
7792
lib_htrequest: htrequestWrite: Writing the request:
    GET /snoop HTTP/1.1
    Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, /*
    Accept-Language: en-us
    Accept-Encoding: gzip, deflate
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR
1.1.4322)
    Host: webserver01.redbook.net
    Connection: Keep-Alive
    $WSIS: false
    $WSSC: http
    $WSPR: HTTP/1.1
    $WSRA: 127.0.0.1
    $WSRH: 127.0.0.1
    $WSSN: localhost
    $WSSP: 80
    Surrogate-Capability: WS-ESI="ESI/1.0+"
lib_htrequest: htrequestWrite: Writing the request content
ws_common: websphereExecute: Wrote the request; reading the response
lib_htresponse: htresponseRead: Reading the response: c2976c
    HTTP/1.1 401 Unauthorized
    WWW-Authenticate: Basic realm="Default Realm"
Content-Language: en-US
Content-Length: 0
lib_htresponse: htresponseSetContentLength: Setting the content length |0|
```

6.4 Securing the application server Web container

This section discusses the authentication and authorization matters related to the application server Web container.

6.4.1 Securing the transport channel

We secured the transport channel for the Web container already in 6.3.1, “Securing the transport channel between the Web server and WebSphere” on page 73. For internal communication between Web and EJB containers,

WebSphere uses IIOP, and the internal communication is protected by default when you enable WebSphere Global Security. Because a WebSphere cell is often located behind a second firewall, it is not necessary to protect internal IIOP transport channels, but if you need to do that, refer to 7.5.3, “RMI/IIOP transport channel protection” on page 150.

6.4.2 Authentication with the Web container

The authentication method defines how the user will be authenticated by the Web application. Before any authorization constraint is applied, the user needs to pass the authentication process using a configured mechanism.

For Web container authentication, WebSphere provides full compliance of the J2EE specification, which defines the following types of authentication methods:

- ▶ Basic authentication
- ▶ Form-based authentication
- ▶ Client certificate authentication

For more details about authentication mechanis, m refer to IBM Redpaper *Security Basics for WebSphere*, REDP-3944.

If a security constraint has been set but no authentication method for a Web module has been configured, the default is to use basic authentication. For any type of authentication methods to work, at least one security constraint should be defined for the requested Web resources and Global Security must be enabled for the application server.

For instructions on how to define security constraints for Web resources, see “J2EE Web module security fundamentals” on page 84. For instructions on how to enable Global Security on the server, please refer to Chapter 3, “Global Security” on page 31.

This chapter presents basic scenarios of how to set up authentication for the ITSObank application.

Configuring form-based authentication

One of the login challenges defined in J2EE Specification is the *form-based login*. It enables the application developer to customize the login process and present an application-specific form by making use of the Form Login Authentication mechanism.

Form login works in the following manner:

1. An unauthenticated user requests a resource protected by the Form Login authentication type.

2. The application server redirects the request to the Login Form defined previously in the Web deployment descriptor.
3. On the HTML login form, the user enters the user ID and password and submits the form.
4. The action, triggered by the form submission, refers to a special servlet *j_security_check*. The Web container, after receiving a request for the *j_security_check* servlet, dispatches the information to the application server's security mechanism to perform the authentication.
5. If the servlet authenticates the user successfully, the originally requested resource is displayed.

If you select **LTPA** as the authentication mechanism under Global Security settings and use form login in any Web application, you must also enable Single Sign-On (SSO). If SSO is not enabled, authentication during form login fails with a configuration error. SSO is required because it generates an HTTP cookie that contains information representing the identity of the user to the Web browser. This information is needed to authorize protected resources when a form login is used.

Form login configuration

The following steps show how to configure form-based login using the Rational Application Developer.

1. Load your Web application module into the Rational Application Developer, in our example: **itsobank.ear**.
2. Within J2EE perspective, click **Dynamic Web Projects** → **itsobank** to expand the tree.
3. Double-click the Deployment Descriptor of **itsobankWeb** Module. The Web Deployment descriptor page will open.
4. Select the **Pages** tab and scroll down.
5. In the Login section, select the **FORM** authentication method.
6. For the Login Page, specify */login/login.html* Web page and for the Error Page specify */login/loginerror.html* Web page.
7. Save the changes.

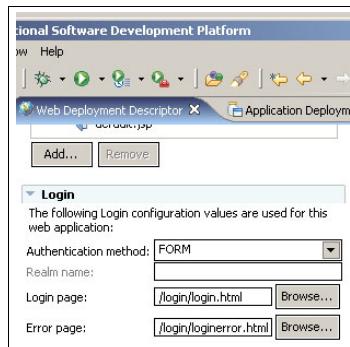


Figure 6-5 Form login configuration

Setting the Authentication Method for the application Web module creates a <login-config> section in a Web deployment descriptor XML file, as shown in the following example.

Example 6-9 Login-config section of the Web deployment descriptor

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>ITSO Bank</realm-name>
    <form-login-config>
        <form-login-page>/login/login.html</form-login-page>
        <form-error-page>/login/loginerror.html</form-error-page>
    </form-login-config>
</login-config>
```

Simple form-based login does not require any extra code development on the server side. Servlet `j_security_check` used by WebSphere Application Server enforces only the name of the input fields that the developer should put in the custom Login Form.

These fields are as follows:

- ▶ **j_username** should be the input field in which a user will type the user name.
- ▶ **j_password** should be the input field into which user will type the password.

The action required for the HTTP POST method is `j_security_check`. A simple HTML code for the custom Login Form is given in Example 6-10.

Example 6-10 Sample custom login form from the ITSOBank application

```
<!-- ..... -->
<form method="post" action="/itsobank/j_security_check">
User name:<input type="text" name="j_username">
```

```
 Password:<input type="password" name="j_password">
<input type="submit" name="action" value="Login">
</form>
<!-- ..... -->
```

Note: The `j_security_check` servlet will not work when Global Security is disabled; the application server will return a Page Not Found error.

This is also true for the `ibm_security_logout` servlet.

Form-based logout

One of the IBM's extensions to the J2EE Specification is the form-based logout. After logging out, the user is required to re-authenticate to have access to protected resources again. This logout form can be on any page with calling a POST action on the `ibm_security_logout` servlet. This form must exist within the same Web application to which the user gets redirected after logging out.

Example 6-11 Sample logout form from the ITSOBank application

```
<form method="post" action="ibm_security_logout" name="logout">
<input type="submit" name="logout" value="Logout">
<input type="hidden" name="logoutExitPage" value="/login/login.html">
</form>
```

Today's e-business Web applications require strict and well-designed security; providing the logout function is one of the important functions. Obviously, closing the browser and destroying the session is always an option for the user, but it is not the most appropriate solution to finish a session with an application.

Combining the logout function with programmatic security, one can implement step up re-authentication, where the user can change credentials and can get higher authority in the application.

Note: The previously introduced logout only works together with form-based login. When the application is configured to use Basic Authentication, the credentials are stored in the client's browser and the browser will send the user name and password to the server together with every request. The only way to log out is to break the session by closing the browser.

6.4.3 Authorization with Web container

For Web container authorization, WebSphere provides full compliance on J2EE specification. In this section, we are going to talk about *declarative security*, that means that the application is not security aware, it does not contain any security

related code. The protection on J2EE level is configured through deployment descriptors and enforced by WebSphere.

The case when the application is security-aware we describe as *programmatic security*. Refer to 6.4.4, “Programmatic security” on page 92 for more information.

J2EE Web module security fundamentals

In a J2EE application architecture, the Web module of the enterprise application is comprised of one or more related servlets, Java Server Pages (JSP files), XML and HTML files that can be managed as a one integrated unit. The files in the Web module are related in the sense that they perform a common business logic function.

The Web modules of the enterprise application run within the Web container of the application server. The Web container, as a runtime environment for the Web application, is responsible for handling requests for servlets, JSP files and other Web components running on the server-side or served from the server side. The Web container creates servlet instances, loads and unloads servlets, creates and manages requests and response objects and performs other servlet management tasks.

This section describes how to configure security for the Web module of enterprise application.

Security roles

WebSphere implements the roles-based security from the J2EE Specification. The security role is a logical grouping of principals. Access to a specific part of the application is granted based on the role, which is then mapped during the development or deployment phase to specific user registry entries. It gives a certain level of transparency to the application development process. The developer need not bother about the different user privileges that can be defined for the application.

The following steps will describe how to define a role for the Web module with Rational Application Developer.

1. Within J2EE perspective, click **Dynamic Web Projects** → **itsobank** to expand the tree.
2. Double-click the Deployment Descriptor of the itsobankWeb Module. The Web Deployment descriptor page will open.
3. Select the **Security** tab.
4. In the Security Roles section, click **Add** to add a new security role.

5. A pop-up window named Add Security Role will open. Enter User as the value for the Name field and click **Finish**.
6. Repeat steps 4 on page 84 and 5, this time creating a Manager security role.
7. Save and close the file.

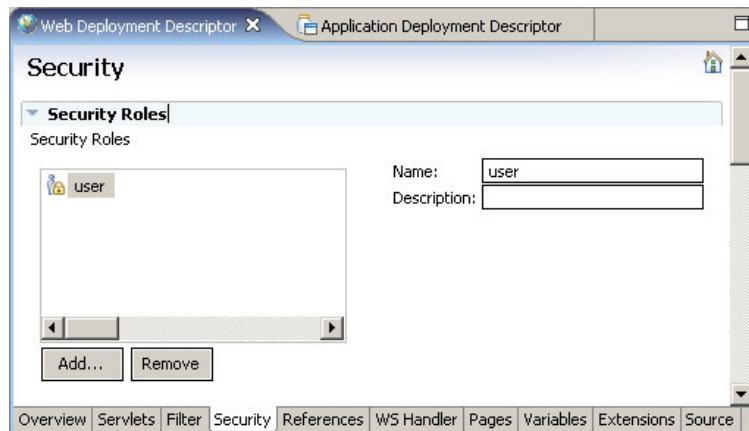


Figure 6-6 Create a new security role for the Web Module

Security constraints

Providing an authentication mechanism for global application security and creating security roles as we did in the previous two sections does not provide the mechanisms to control access to the Web resources.

Security constraints declare how the content of the application is protected. For a given security constraint, three things should be defined:

- ▶ One or more Web resources that define actual application components that are to be protected by the security constraint. A Web resource is a set of URL patterns and HTTP methods in those resources. All requests that will be matched with the pattern defined for a given Web resource will be subject to a security constraint.
- ▶ An authorization constraint that defines roles which will be provided access to the Web resources existing within the security constraint. An authorization constraint is a set of roles that the user must be granted in order to have access to a Web resource collection existing within a security constraint. In order to have access to the Web resource, the user should be granted at least one of the roles that are defined within the Authorization constraint.
- ▶ The User Data Constraint indicates the transport layer setting for client/server communication in order to satisfy given security constraint. This setting should guarantee either content integrity (preventing tampering in transit) or

confidentiality (preventing reading data during transfer). User Data Constraint may override standard security settings for the application. For example, access to some functions of the application may require just basic login using a user ID and password, and at the same time some functions may require a higher level of protection. User Data Constraint allows an application deployer to introduce such protection.

If Global Security is enabled, and a security constraint is set for a particular resource, then the resource is secured.

J2EE Security role reference

During the development phase of the application, the actual role names for security constraints may not be known to the groups of developers. On the other hand, the actual role names in a deployed runtime environment may not be known until the Web application and EJB modules are ready and assembled into the .ear file. Therefore, the role names used during development are considered to be "logical roles." These logical roles are then mapped by the application deployer into the actual runtime roles during the application assembly and deployment phase.

Security role references provide a level of indirection to isolate roles used during development and actual runtime roles. They link the names of the roles used in the module to the corresponding name of the role in the encompassing application.

The definition of the "logical" roles and the mapping to the actual runtime environment roles are specified in the <security-role-ref> element of both the Web application and the EJB jar file deployment descriptors, web.xml and ejb-jar.xml, respectively.

The following steps will show how to define Security Role references for the Web module in Rational Application Developer.

1. Load your Web application module into the Rational Application Developer, in our example: itsobank.ear.
2. Within J2EE perspective, click **Dynamic Web Projects** → **itsobank** to expand the tree.
3. Open the Web Deployment Descriptor for the **itsobankWeb** Module. Select the **Servlets** tab.
4. Select **TransferServlet** in the Servlets and JSPs section.
5. In the Security Role References select User at the Role Link drop-down list. Enter **The User** role for the Role reference name. The User role was already created when we defined security constraints. See Figure 6-7 on page 87.
6. Save and close the file.



Figure 6-7 Setting security role reference for TransferServlet

Securing content served by WebSphere Application Server

On the J2EE security level, WebSphere Application Server can secure Web resources using role-based declarative security mechanisms. This means that the logical application security structure is defined independently from the application itself. The logical security structure is stored in deployment descriptors of the application.

We can divide Web resources into two categories:

- ▶ **Static resources**

By this we mean all the resources that do not change the response output over time. Static resources are for example static HTML pages, different image files and similar.

Static resources of the enterprise application can be secured only if they are served by WebSphere. WebSphere cannot manage access to the static content that resides on the Web server. All the static content that needs to be protected by WebSphere Application Server must be packaged into the Web module (.war, Web Archive file). Static HTML pages can be served by the servlet that implements file serving behavior

- ▶ **Dynamic resources**

By this we mean all the resources that change the response output over time, depending on different input parameters. Dynamic resources are, for example servlets, JSPs.

For all the Web resources, WebSphere Application Server allows you to protect them on the HTTP method level. For example, the POST method of a servlet can be part of a different security constraint than the GET method. The full list of predefined methods that can be secured is as follows:

- ▶ GET
- ▶ POST
- ▶ PUT
- ▶ DELETE
- ▶ HEAD
- ▶ OPTION
- ▶ TRACE

Using method level security constraints for resources, you may want to separate the content that all the users can view from the administrative functions that only privileged users will be allowed to access. In WebSphere Application Server, this is done using different security constraints for the different methods.

Configuring security constraints

The following instructions show how to set up constraint to protect content for the Web application module:

1. Load your Web application module into the Rational Application Developer, in our example: itsobank.ear.
2. Within J2EE perspective, click **Dynamic Web Projects** → **itsobank** to expand the tree.
3. Open the Web Deployment Descriptor for the **itsobankWeb** Module.
4. Select the **Security** tab.
5. In the **Security Constraints** section click **Add** to add a new security constraint.
6. A pop-up window named *Add Security Constraints* will open. Enter ITSO Bank security constraint as the value for the Name field and click **Next**.
7. In this step, you will specify Resource name, HTTP methods that are allowed to be issued on the resource and URL patterns on which this security constraint applies. Fill in the values as in Figure 6-8 on page 89.

Important: Make sure that URL patterns correctly match your resource names or URL bindings. There is no value syntax checking, so if this is entered wrongly, you will end up with an unprotected resource and a security constraint for a non-existent resource.

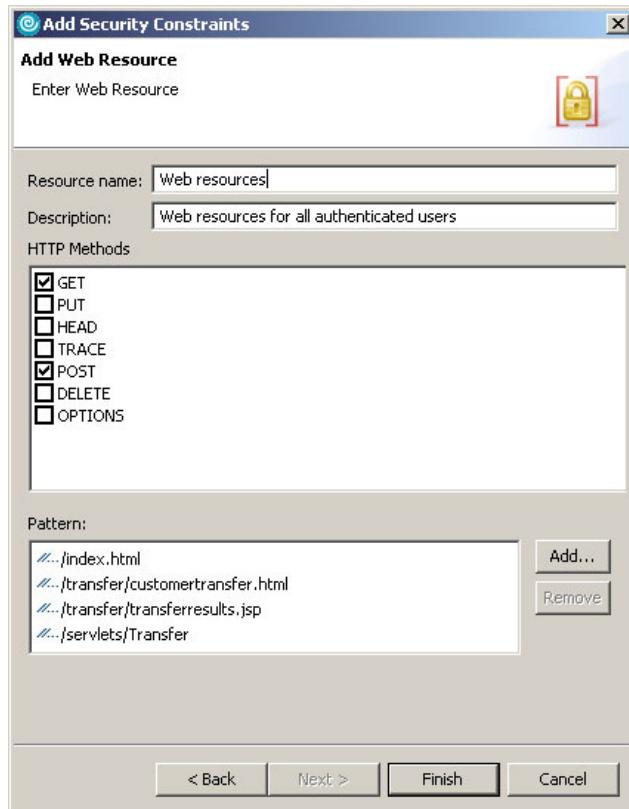


Figure 6-8 Configuring security constraint for Web resources

8. Click **Finish**.

Configuring authorization constraints

So far, a security constraint for static resources has been created. It defines what HTTP methods are allowed on certain URL definitions which represents our Web resources. More specifically, within our security constraint we defined that only GET and POST methods can be run on our Web resources. But by now we have not specified who is allowed to run define security constraint, specifically who is allowed at all to run GET and POST over our Web resources. Take the next steps to define authorization constraint for just created security constraint:

1. We are still on the Deployment Descriptor Security tab of the itsobankWeb Module. Select the **ITSO Bank** security constraint, then go to the Authorized Roles section and click **Add**.
2. A pop-up window named Define Authorization Constraint will open. Enter **Web resources auth constraint** as the value for the Description.

3. On the available Role Names list, make sure that the **User** and **Manager** roles are selected.
4. Click **Finish** and save the changes. See the overview of defined security properties in Figure 6-9.

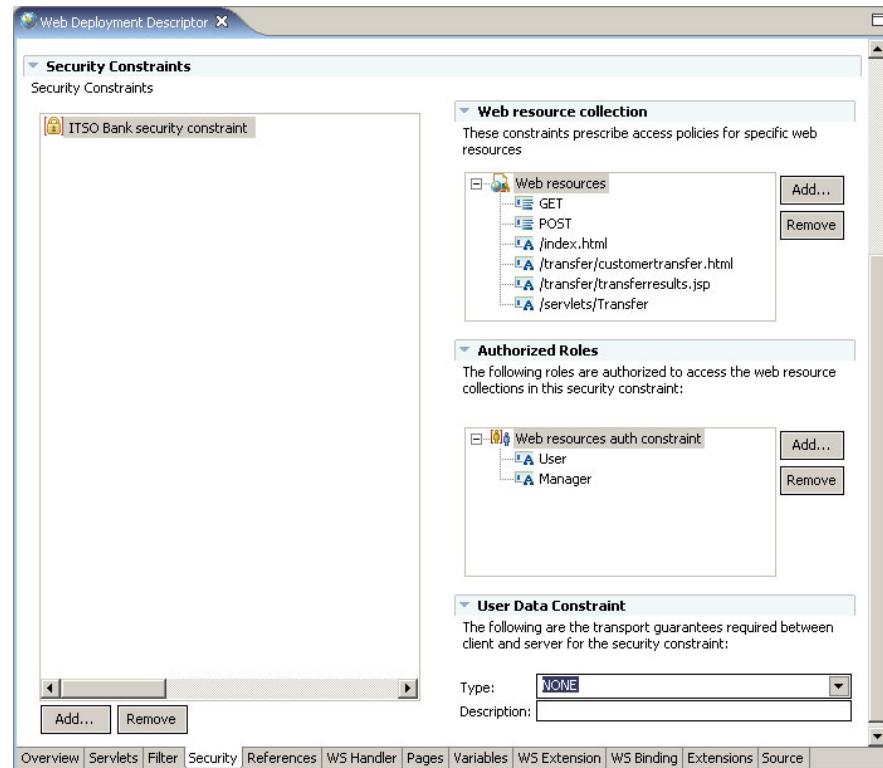


Figure 6-9 Overview of configured security constraints for Web resources

Note: User Data Constraint in this window allows you to choose a *Transport guarantee*, which defines how the communication between the client and the server is to be protected. There are three options to choose from:

► **None**

No constraint indicates that the application does not require any transport guarantee.

► **Integral**

This ensures that data cannot be changed in transit. In practice, this means that a request must be transmitted over an SSL encrypted channel.

► **Confidential**

This ensures that data cannot be viewed in transit. In practice, this means that the request must be transmitted over an SSL encrypted channel.

5. Click **Finish** and save the changes.

Adding security and authorization constraint for the AccountsView page

So far, we have created a security constraint for the majority of our sample application Web resources and then given authorization on this constraint for previously created User and Manager security roles. Whatever users will be mapped to this roles during application installation, they will all have access to these resources. However, our application contains a special static HTML page for which we do not want that every user has access. The page is named the Accounts View page and we only want that the Manager security role has access to this page. Do the following:

1. Repeat steps 1 through 8 in “Configuring security constraints” on page 88 add to another security constraint. Use the values in Figure 6-10 on page 92.

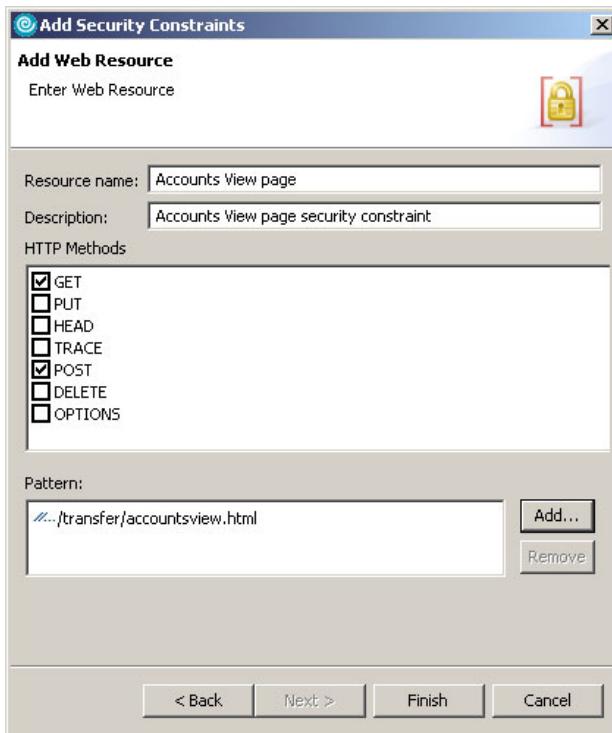


Figure 6-10 Creating security constraint for the Accounts View page

2. Repeat steps 1 through 4 in “Configuring authorization constraints” on page 89 to create another authorization constraint for the just created Accounts View security constraint. This time, make sure that only the **Manager** security role is selected.

6.4.4 Programmatic security

In this section, we will talk about *programmatic security*; this means that the application is security aware: it contains the code which handles security providing any authorization and authentication capabilities that are beyond J2EE security. The opposite case when the protection is configured on the J2EE level and through application deployment descriptors we describe as declarative security.

Refer to 6.4.3, “Authorization with Web container” on page 83 for more details about that subject.

Roughly, we can divide programmatic security into:

- ▶ Java Authentication and Authorization Service (JAAS) where we make use of mechanisms available in JAAS API.
- ▶ J2EE programmatic security where we make use of a few extra Java methods available as part of Java Servlet 2.x specification.

This section will focus only on the latter. If you need more details about JAAS security refer to Chapter 4, “JAAS for authentication and authorization in WebSphere” on page 49.

J2EE servlet security methods

The Servlet 2.4 specification defines three methods that allow programmatic access to the caller’s security information of HttpServletRequest interface.

Important: The methods `getRemoteUser()` and `getUserPrincipal()` return `null` as a result even if the user is logged in, unless the servlet or the JSP itself is secured.

- ▶ **String getRemoteUser()**

The `getRemoteUser` method returns the user name that the client has used to log in.

```
String user = request.getRemoteUser();
```

- ▶ **Boolean isUserInRole(String roleName)**

The `isUserInRole` method allows the developer to perform additional checks on the authorization rights of a user which are not possible, or more difficult, to perform through the deployment descriptor of the servlet.

```
if (request.isUserInRole("Manager")) {
    // the user is in the manager role
    // ...
}
```

- ▶ **java.security.Principal getUserPrincipal()**

The `getUserPrincipal` method allows the developer to get the name of the current caller. To do this, you need to call `getName()` on the `java.security.Principal` object returned.

```
Principal principal=request.getUserPrincipal();
String username=principal.getName();
```

Sample usage of security methods

The following example is a modified code snippet from the ITSOBank sample application. You can find similar code in the TransferServlet.java in the `doPost()` method. For more details, check the comments in the source below or in the sample application.

Example 6-12 Sample code using the servlet security methods

```
// getting the environment variables for restricted role  
// and for maximum transferable amount  
restrictedRole=(String)environment.lookup("RestrictedRole");  
maxWebTransferAmount=(Integer)environment.lookup("MaximumWebTransferAmount");  
// checking if the user is restricted to a certain amount of transfer  
if(request.isUserInRole(restrictedRole) &&  
transferAmount>maxWebTransferAmount.intValue()) {  
    // create an error message  
    // the user cannot transfer the requested amount  
    // forward the request to the response page with the message  
}  
// get the principal from the request  
Principal principal=req.getUserPrincipal();  
// print out the user information about the servlet invocation  
System.out.println("Transfer Servlet was invoked by user:  
"+req.getRemoteUser()+" , principal: "+principal.getName());
```

With the security methods, the servlet will not let the user in a restricted role submit a transfer greater than the maximum transferable amount.

You can see that also two environment variables are used in the code for this purpose: RestrictedRole which defines which role is restricted and MaximumWebTransferAmount which defines the upper limit of allowed transferable amount. In Example 6-13 you can see how these two environment variables are defined in Web application deployment descriptor, the web.xml file.

Example 6-13 Environment variables needed for programmatic Web security sample code

```
<env-entry>  
    <env-entry-name>MaximumWebTransferAmount</env-entry-name>  
    <env-entry-type>java.lang.Integer</env-entry-type>  
    <env-entry-value>5000</env-entry-value>  
</env-entry>  
<env-entry>  
    <env-entry-name>RestrictedRole</env-entry-name>  
    <env-entry-type>java.lang.String</env-entry-type>  
    <env-entry-value>User</env-entry-value>  
</env-entry>
```

You can see that in our case RestrictedRole variable is mapped to the User security role and that MaximumWebTransferAmount variable is set to 5000.

Testing the programmatic security sample

In order to test our programmatic security code in the ITSOBank application, the following prerequisites must be met:

- ▶ The ITSOBank sample application must be installed into a WebSphere Application server and security must be enabled on that server.
- ▶ There are two security roles defined for the application; User and Manager. Make sure that you map these two roles to different users.

To test the programmatic security code, do the following:

1. Make sure that ITSOBank application is started. Open a new browser window and enter the ITSOBank index page URL; in our case:

`http://localhost:9080/itsobank/index.html`

2. Because this is a protected URL and you are not authenticated yet, you will get the login page. First, perform a login with a user mapped to the User security role. In our case WebSphere used an LDAP for the user registry. And during application installation we mapped user john to the User security role, so we enter the following in the login page:

Userid: john

Password: test

Click the **Login** button.

3. On the Welcome page, click **Customer Transfer**. Customer transfer page will open. Enter 10000 as value for the Transferred amount field; you can enter any number higher than 5000. Values for the other fields are not important.
4. Click the **Transfer** button. You will get the response similar to that shown in Figure 6-11 on page 96.

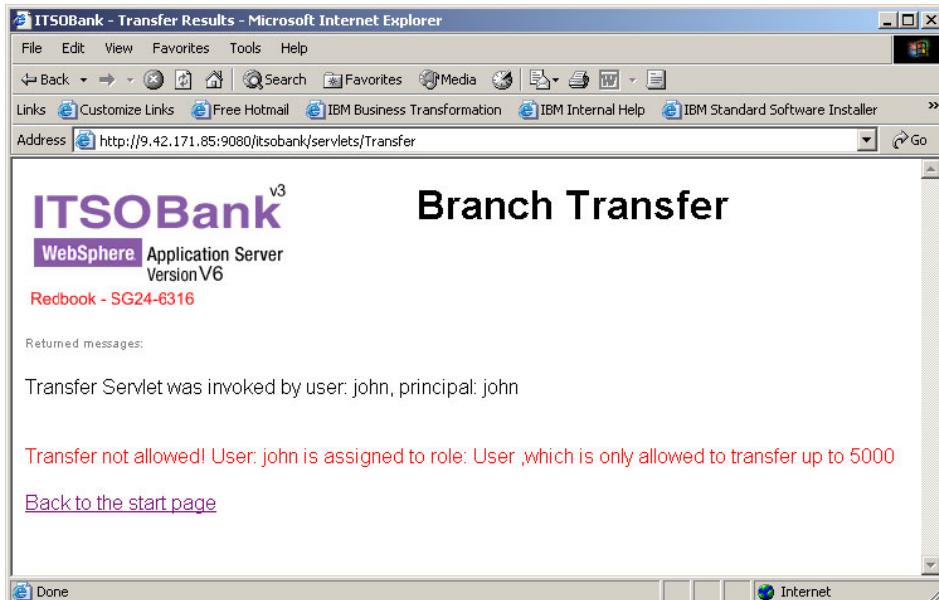


Figure 6-11 Programmatic login sample response show that transfer was not allowed

The first line shows which user under which principal called the TransferServlet servlet. Here is the code snippet responsible for this line, you can see which servlet programmatic methods are used:

Example 6-14 The use of getRemoteUser() and getUserPrincipal() methods

```
// check the transfer amount for restricted roles
    Principal principal=req.getUserPrincipal();
    if(principal!=null) {
        messagePrincipal="Transfer Servlet was invoked by user:
"+req.getRemoteUser()+" , principal: "+principal.getName();
```

The second line in the Branch Transfer result page shows negative response - the transfer was not allowed. In the code we are checking if the user is mapped to the User role and if wanted transfer amount exceeds the specified limit (5000) then transfer is not allowed. Here is the code snippet:

Example 6-15 The use of isUserInRole() methods

```
if(message==null && req.isUserInRole(restrictedRole) &&
transferAmount>maxWebTransferAmount.intValue())
    message="Transfer not allowed! User: "+req.getRemoteUser()+" is
assigned to role: " + restrictedRole +" ,which is only allowed to transfer up
to "+ maxWebTransferAmount;
```

```
else
    message="Transfer initiated between customer:"+customerID+" and
branch:"+branchID+", the amount of:"+transferAmount;
```

5. Now, click **Back to the start page** link. Click the **Log out** button to logout
6. Log in once again, this time with another user which must not be mapped to the User security role.
7. Repeat the Steps 3 and 4, this time you will get the response shown in Figure 6-12.

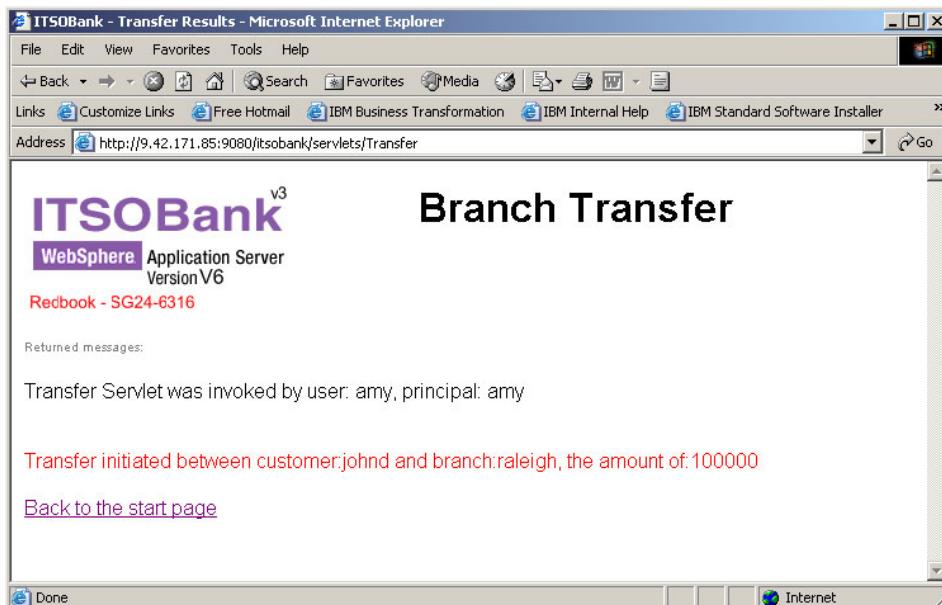


Figure 6-12 Programmatic login sample response show that transfer was fine

6.5 Options

In the following section, many additional transport security, authentication and authorization options for Web servers and WebSphere are described.

6.5.1 Configuring te HTTP Digest Authentication method

In this section, we will present a simple scenario of how to implement digest HTTP authentication for the Web server.

For more details about HTTP Digest Authentication, please see the protocol definition document:

<http://www.ietf.org/rfc/rfc2617.txt>

Configuring your Web Server to use Digest authentication

The following steps will describe how to configure the IBM HTTP Server to use Digest authentication.

1. Open httpd.conf, which is the configuration file for IBM HTTP Server.
2. Make sure that auth_digest_module definition in the LoadModule list is uncommented, as in Example 6-16.

Example 6-16 auth_digest module definition

```
#LoadModule auth_anon_module modules/mod_auth_anon.so  
#LoadModule auth_dbm_module modules/mod_auth_dbm.so  
LoadModule auth_digest_module modules/mod_auth_digest.so
```

3. Add the **Directory** directive to protect C:\IBMHttpServer\htdocs\en_us directory. This directory is set as a global Web server root so when we call the Web server just using hostname, the Web server is going to search for the index.html file under the Web server root.

Within Directory, we will specify additional security directives that will be effective just on that scope. See Example 6-17

Example 6-17 Configuring HTTP digest authentication

```
<Directory "C:/IBM HTTP Server/htdocs/en_US">  
  
AuthType Digest  
AuthName "ITSO"  
AuthDigestDomain /  
AuthDigestFile "C:/IBM HTTP Server/conf/usersDig"  
Require valid-user  
  
Options None  
AllowOverride None  
</Directory>
```

4. Save httpd.conf.

6.5.2 Configuring LDAP authentication with IBM HTTP Server

In this section, we will present a simple scenario of how to implement basic HTTP authentication for the Web server when user registry is stored in LDAP directory.

To test the scenario, we used IBM Tivoli Directory Server version 5.2. The following instructions assume that all the software is installed and you already have an LDAP server populated with users, see Figure 2-2 on page 10 for details about LDAP data structure.

If the LDAP server is on a different machines, client access software for LDAP server will be necessary. The LDAP client software can be downloaded from:

<http://www-4.ibm.com/software/network/directory/downloads>

We will enable security for all the static Web components in the C:\IBMHttpServer\htdocs\en_us directory.

The following steps will show you how to enable basic authentication with LDAP for IBM HTTP Server.

Preparing the necessary configuration files

The following steps will show you which files need to be defined for the Web server and also how to use those files. The *ldap.prop* file is an LDAP configuration file for the Web server. It is stored in the conf directory of the server (in our case it is C:\IBMHTTPServer\conf). A sample LDAP configuration file with explanation of each directive is supplied with Web server software. For basic authentication, the following entries are included.

Example 6-18 LDAP configuration for IBM HTTP Server

```
1dap.realm=LDAP Realm
1dap.URL=ldap://m23x2673.itso.ral.ibm.com/o=itso,c=us
1dap.transport=TCP
1dap.application.authType=Basic
1dap.application.DN=cn=root
1dap.application.password.stashFile=C:\IBMHTTP\bin\ldap.sth
1dap.user.authType=Basic
1dap.user.name.filter=(&(objectclass=inetorgperson)(cn=%v1))
1dap.group.name.filter=(&(cn=%v1)(|(objectclass=groupofnames)(objectclass=group
ofuniqueNames)))
1dap.group.memberAttributes=member
uniqueMember
```

Tip: If you are using Windows, make sure that you use short file name types when specifying a fully qualified path name for the configuration files in *ldap.prop*. See the *ldap.application.password.stashFile* directive above.

1. *1dap.URL* is of the form *1dap://<hostName>/<BaseDN>*
2. *1dap.application.DN* is the DN by which the Web server authenticates itself to the LDAP Server.

3. *ldap.sth* is a stash file containing an encrypted password for the Web server to authenticate with LDAP. If you are using Windows, make sure you specify the fully qualified path name with short Windows file name. Using quotes does not work.

You need to decide with which user name and password the Web server will connect to LDAP. To create the stash file, enter at the command prompt:

```
C:\IBMHTTPServer\bin\ldapstash <password> C:\IBMHTTPServer\bin\ldap.sth
```

Configuring your Web Server to use LDAP for authentication

The following steps will describe how to configure the IBM HTTP Server to use LDAP for authentication.

1. Open httpd.conf, which is the configuration file for IBM HTTP Server.
2. Add `ibm_ldap_module` definition to the end of the LoadModule list.

Note: There is a difference whether specifying LDAP module for UNIX or Windows. See Example 6-19.

Example 6-19 Adding `ibm_ldap_module` definition to `httpd.conf`

```
#LoadModule deflate_module modules/mod_deflate.so
LoadModule ibm_ssl_module modules/mod_ibm_ssl.so

#for Windows only
LoadModule ibm_ssl_module modules/IBMModuleLDAP.dll

#for UNIX only
LoadModule ibm_ssl_module modules/mod_ibm_ldap.so
```

3. Add the Directory directive to protect C:\IBMHttpServer\htdocs\en_us directory. This directory is set as a global Web server root so when we call the Web Server just using hostname, the Web server is going to search for the index.html file under the Web server root.

Within Directory, we will specify additional security directives that will be effective just on that scope. See Example 6-20.

Example 6-20 Configuring HTTP basic authentication with LDAP module

```
<Directory "C:/IBM HTTP Server/htdocs/en_US">

    LdapConfigFile "C:/IBM HTTP Server/conf/ldap.prop"
    AuthName "LDAP Realm"
    AuthType basic
    Require valid-user
    Options None
```

```
AllowOverride None
```

```
</Directory>
```

4. Save httpd.conf.

Testing LDAP authentication with Web server

Now, let's test how LDAP authentication works:

1. Open a new browser window on the machine where the Web server is running.
2. In the address bar, enter `http://localhost` or the address for the Web server.
3. You will get an authentication pop-up window. Depending on your LDAP user registry scheme, you need to enter a valid user. According to our user registry scheme that is used throughout this redbook, we entered the following (see Figure 6-13):

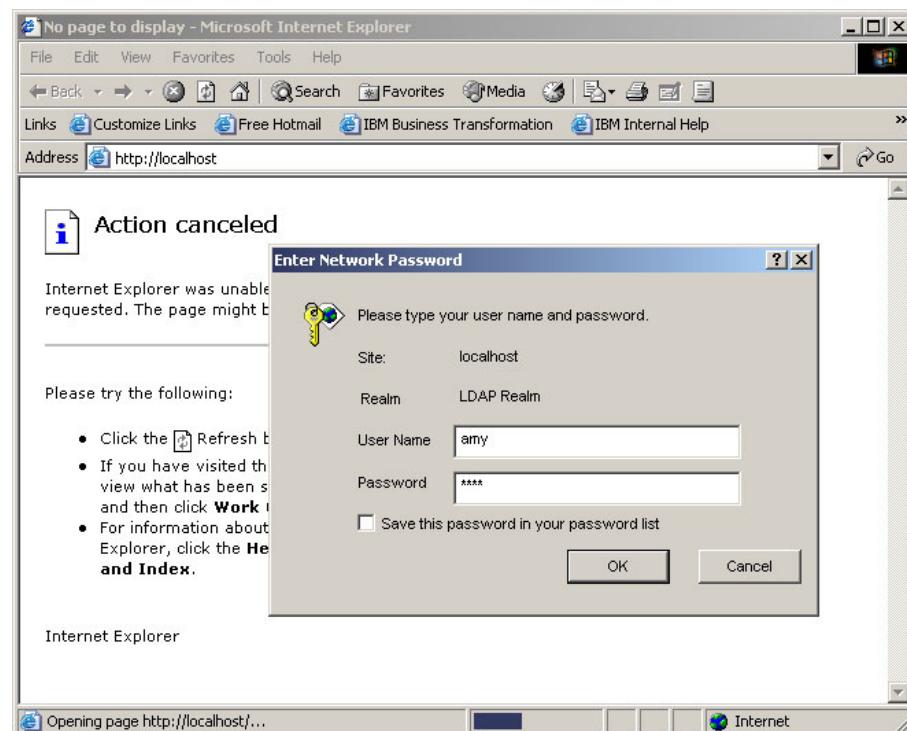


Figure 6-13 Testing the LDAP authentication module for IBM HTTP Server

4. Click **OK** and if the authentication process went smoothly, you will get the standard IBM HTTP index page output.
5. Before starting IBM HTTP Server, you may also enable traces for the LDAP module. To do that, the following variables must be exported into the system:


```
LDAP_DEBUG=65535
LDAP_TRACE_FILE=C:\ldaptrace.txt
```
6. After successful authentication in the LDAP module trace, you will get output similar to the following:

Example 6-21 The LDAP module trace output

```
ldap_authenticate(): entered
LDAP_obtain_session()
LDAP_authenticate_user()
auth_type (BASIC)
calling LDAP_authenticate_user_using_basic
LDAP_authenticate_user_using_basic(): user_name (amy)
LDAP_user2DN(): user_name (amy)
LDAP_user2filter(): user_name (amy)
LDAP_prepare_filter(): template ((&(objectclass=inetorgperson)(cn=%v1)))
the resulting filter: ((&(objectclass=inetorgperson)(cn=amy)))
LDAP_prepare_filter(): returning 0
LDAP_user2filter(): returning 0
LDAP_perform_search(): base (o=itso,c=us), filter
((&(objectclass=inetorgperson)(cn=amy)))
looking in the cache for the base, scope and filter
LDAP_cache_find_ele(): entered
looking for base [o=itso,c=us], filter
[(&(objectclass=inetorgperson)(cn=amy))], scope [2]
no cached answer
reusing 'LDAP Realm' application connection.
Search start: 1100719467, end: 1100719467
converted '(&(objectclass=inetorgperson)(cn=amy))' to DN
'cn=amy,ou=users,o=itso,c=us' for realm 'LDAP Realm'
adding DN (cn=amy,ou=users,o=itso,c=us) to cache
LDAP_cache_find_ele(): entered
looking for base [o=itso,c=us], filter
[(&(objectclass=inetorgperson)(cn=amy))], scope [2]
LDAP_perform_search(): returning 0
LDAP_user2DN(): returning 0
using DN (cn=amy,ou=users,o=itso,c=us)
calling LDAP_obtain_connection
: LDAP_open_connection(): using LDAP V3 API, cp->Version (3)
: connecting to [9.42.171.77, 389]
: cp->Version (3); cp->Transport (TCP)
: LDAP_init(9.42.171.77, 389)
: connected
```

```
: setting deferrals
: setting timeout
: not an application connection
: opened new user connection for 'LDAP Realm'; expiration: 1100720067
calling LDAP_simple_bind_s() with DN (cn=amy,ou=users,o=itso,c=us)
successful authentication
updating the password cache
LDAP_cache_find_ele(): entered
looking for base [o=itso,c=us], filter
[(&(objectclass=inetorgperson)(cn=amy))], scope [2]
cache: [o=itso,c=us], [(&(objectclass=inetorgperson)(cn=amy))], [2]
setting correct password for 'cn=amy,ou=users,o=itso,c=us' cache
LDAP_authenticate_user_using_basic(): returning 0
LDAP_authenticate_user(): returning 0
LDAP_release_session()
```

The output shows a sucessful connection to the LDAP server and a successful authentication query.

6.5.3 Configuring SSL certificate based client authentication method for IBM HTTP Server

The Web client may also provide a digital certificate in order to provide an identity during an SSL initialization.

This section discusses how to use client side certificates with your Web server and with your WebSphere Application Server. It will also show how to configure your servers to support client-side certificates and use them as a base for user authentication.

Creating a personal certificate

Typically, the creation of a client-side certificate involves a Certificate Authority, but for purpose of this section we will create a self signed personal certificate which we will import into Web Server's and WebSphere's key store databases.

The process for requesting and installing a personal client-side certificate on Windows is documented in this section.

1. Use ikeyman tool to create a new key store and a self signed personal certificate.

Note: When creating a new key store use PKCS12 database format.

2. Use values as in Figure 6-14 on page 104.

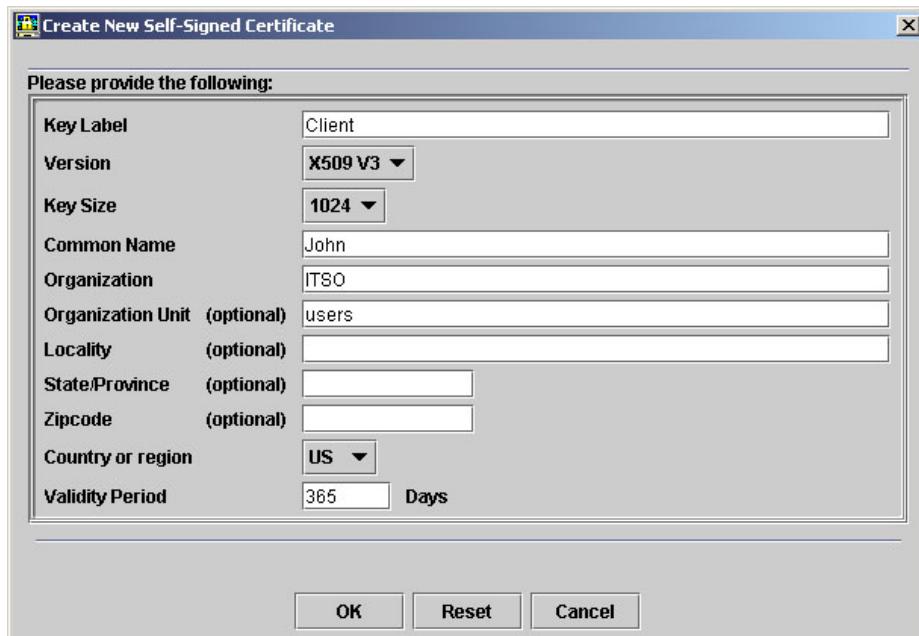


Figure 6-14 Create a new self-signed certificate to be used with a Web browser

3. After the certificate is successfully created, extract it to an .arm file and then import it to Web Server's CMS type key datastore.

For more details about creating self signed certificates, see the IBM Redpaper *Security Basics for WebSphere*, REDP-3944-00, section 3.9, “Security Tools” on page 36.

Importing the certificate into your Web browser

Now, we must import this certificate into our Web Browser and this case we are using Internet Explorer. The procedure is similar with other types of Web Browser:

1. From the Internet Explorer menu bar, select **Tools** → **Internet Options** → **Content** → **Certificates**.
2. Select **Client Authentication** from the Intended purpose drop-down list.
3. Click the **Personal** tab and then click **Import**.
4. The Certificate Import Wizard will open, click **Next** to get to the second step.
5. Browse for the p12 file - the PKCS12 key datastore that has been created for this purpose. Click **Next**.

6. Provide the password to open the p12 file. Click **Next**. Make sure you checked the **Enable strong private key protection** option. You will get a prompt pop-up window every time the certificate is accessed.
7. Click **Next** once more and then click **Finish**. A pop-up window named *Creating a new private exchange key!* will open. Set the security level to medium and click **OK**.
8. In the Personal tab, you will see the certificate that has just been imported. Select the certificate and click **Advanced...** button. In the **Advanced Options** make sure that **Client Authentication** check mark is checked.

Modifying the Web server to support client certificates

You must ensure that the selected Web server is configured to request client side certificates. In the following example, we will use the IBM HTTP Server to show how to configure SSL for the Web server to force the clients to send their certificates.

1. The Web server requires SSL to be enabled and configured in order to use client side certificates. Follow the steps in 6.2.1, “Securing transport channel between Web browser and Web server” on page 67 to enable SSL for your IBM HTTP Server if you have not done so yet.
2. Open the httpd.conf file.
3. Within other SSL configuration directives, add the **SSLClientAuth** directive as in Example 6-22.

Example 6-22 Adding client authorization directive to SSL definition

```
SSLDisable
Listen 443

<VirtualHost w2ksrvvm01:443>
SSLEnable
KeyFile "C:/IBM HTTP Server/conf/keys/IHS6Certificates.kdb"
SSLClientAuth required
</VirtualHost>
```

This is the most basic SSL setup for certificate client authentication, but there are other SSL directives that you can use to set the SSL configuration more specifically to your needs. Further explanation of those is not within the scope of this redbook, so please refer to IBM HTTP Server documentation.

4. Save the httpd.conf configuration file and restart IBM HTTP Server.

Testing the client side certificate with IBM HTTP Server

To test client certificate authentication, take the following steps:

1. Open a new Internet Explorer window and open the `http://localhost` location.
2. Because the certificate is protected, you will get a pop-up window saying that the Internet Explorer application is trying to access your personal certificate; see Figure 6-15. Click **OK** and the default IBM HTTP Server will open.

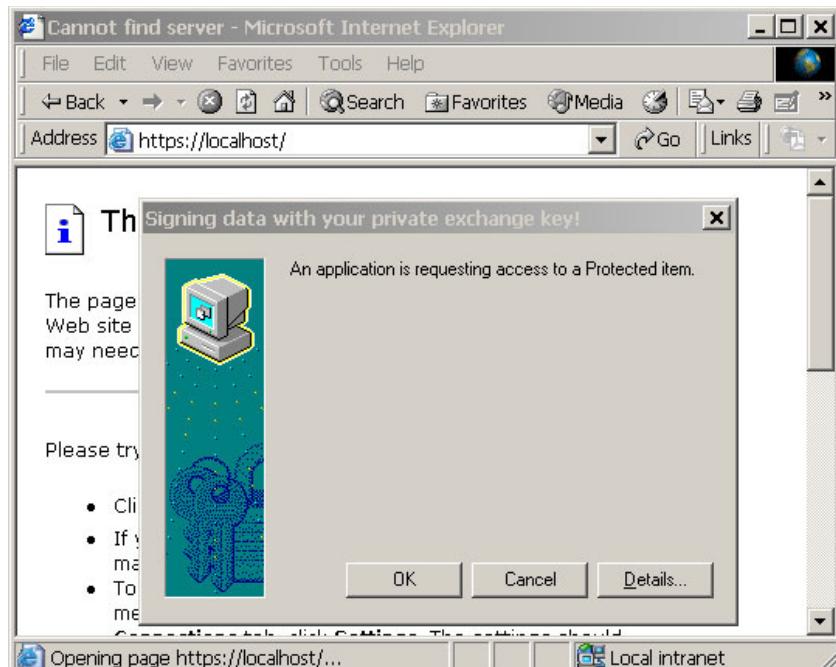


Figure 6-15 Testing client certificate authentication with IBM HTTP Server

3. To see what is happening in IBM HTTP Server, you might have set the log level directive in `httpd.conf` configurational file:

```
LogLevel Debug
```

After successful client authentication, you will get similar entries in the IBM HTTP Server `error.log` file, as follows:

Example 6-23 IBM HTTP Server `error.log` shows SSL client authentication

```
[info] New Session ID: 0
[info] Cert Body Len: 664
[info] Serial Number: 41:89:42:cb
[info] Distinguished name CN=John,OU=users,O=ITSO,C=US
[info] Common Name: John
[info] Country: US
[info] Organization: ITSO
[info] Organization Unit: users
[info] Issuer's Distinguished Name: CN=John,OU=users,O=ITSO,C=US
```

```
[info] Issuer's Common Name: John  
[info] Issuer's Country: US  
[info] Issuer's Organization: ITSO  
[info] Issuer's Organization Unit: users
```

6.5.4 Configuring SSL certificate based client authentication method for WebSphere Application Server

In this section, we will show how to configure client certificate authentication for our applications.

When we are using client certificate authentication with our Web modules, WebSphere security service attempts to map the data from the digital certificate with the user data of selected user registry, which can be either one of the following:

- ▶ LocalOS

Note: In case, we use LocalOS option for the user registry, the certificate Distinguished Name (DN) is parsed and the name between the first equals (=) and comma (,) is used as the mapped name. If the DN does not contain the "=", the complete name is used. If there is no "," in the DN, everything after the "=" is used as the name.

- ▶ LDAP

If we use LDAP server for the user registry, WebSphere provides two way of matching client certificate information to LDAP; mapping by exact distinguished name. Mapping by filtering certificate attributes. Both options will be described in the sections that follows.

If the certificate successfully maps to a user, then the holder of the certificate is regarded as the user in the registry and is authorized as this user.

Configuring J2EE Web application for client certificate authentication

By specification in J2EE application Web module, authentication method can be configured to be one of five available types, including unspecified. This is done in the Web deployment descriptor file.

Take the following steps to configure our ITSObank sample application for client certificate authentication:

1. Load itsobank.ear application into the Rational Application Developer.

2. Within the J2EE perspective, click **Dynamic Web Projects** → **itsobank** to expand the tree.
3. Double-click the Deployment Descriptor of the **itsobankWeb** module. The Web Deployment descriptor page will open.
4. Select the **Pages** tab and scroll down to the Login section.
5. In the Login section, select **CLIENT_CERT** authentication method, as displayed on Figure 6-16. Now, we do not need to log in and log out pages anymore; they can just be deleted.

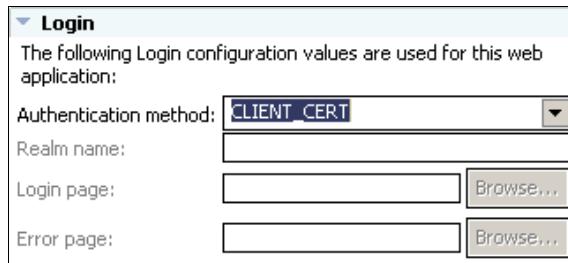


Figure 6-16 Configuring Web module for client certificate authentication

6. Save the changes. The only thing that we actually changed in our application is the Web module deployment descriptor - the Web.xml file. The changes are in within the *login-config* tag, as displayed:

Example 6-24 Changes in the login-cnfig tag

```
<login-config>
    <auth-method>CLIENT-CERT</auth-method>
    <realm-name>ITSO Bank</realm-name>
</login-config>
```

7. Export the application EAR file; for testing purposes, we will install it on a WebSphere Application Server.

Configuring WebSphere for the LDAP Certificate Filter option

This section assumes the following prerequisites are met:

- ▶ You have successfully installed a personal certificate into a client Web browser. For more details, refer to “Creating a personal certificate” on page 103 and “Importing the certificate into your Web browser” on page 104.
- ▶ Also, the WebSphere Web container must be configured to use the SSL repertory which uses our previously configured key store. For more details refer to “Creating a new SSL entry” on page 74 and “Modifying the Web Container configuration to support SSL” on page 75.

It is anticipated that the personal certificate subject Distinguished Name (DN) does not necessarily match, in any way, your LDAP Distinguished Name (DN).

In the following sample, we will use self signed personal certificate as described in “Creating a personal certificate” on page 103

The SubjectDN value of the certificate in our case is:

CN = John
OU = users
O = ITSO
C = US

The next step is to modify WebSphere LDAP filtering rules to map the certificate subjectDN field to the IBM Tivoli Directory Server LDAP uniqueIdentifier (uid) field for a given user. You do not necessarily have to use the uniqueIdentifier field. However, you should ensure that the data type of the field selected is capable of handling the specific value and the certificate attribute selected for authentication is unique between certificates.

Also ensure that WebSphere has the right to search such a field when performing authentication.

The following steps will show you how to configure WebSphere Application Server to use the certificate filter as required.

1. Log in to the WebSphere Administration Console.
2. In the Global Security administration page, go to User registries section.
3. Click the **LDAP** link option to open the LDAP User registry configuration page.
4. In the Additional Properties section, click the **Advanced LDAP user registry settings** link option.
5. The Advanced LDAP user registry settings page will open. As shown in Figure 6-17 on page 110, set the following:
 - Certificate Map Mode: **CERTIFICATE_FILTER**.
 - Certificate Filter: **uid=\${SubjectDN}**

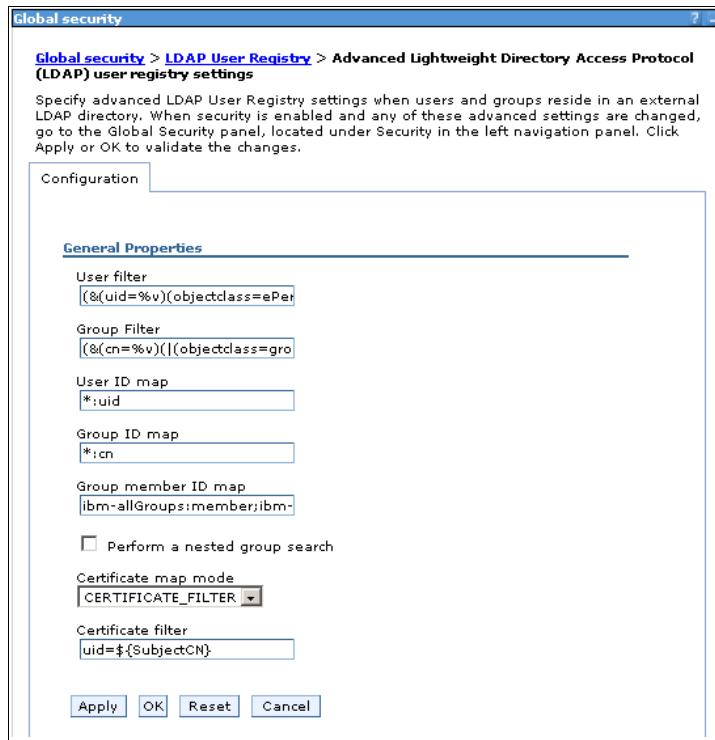


Figure 6-17 Setting the `CERTIFICATE_FILTER` client certificate map mode in WebSphere Application Server

6. Click **OK**, then save the configuration for WebSphere.
7. Restart the application server in order for the changes to become effective.

Testing the client side certificate

First, we are going to test the client certificate authentication using the Default Application that ships with WebSphere and use the snoop servlet by accessing it with your Web browser:

1. Make sure that Default Application is started.
2. Open a new browser window and access the following address from the client: `https://<your_server_name>:9666/snoop`, to determine if your browser is correctly passing a client certificate.

Important: The Default Application is BASIC authentication enabled - that also means that client certificate authentication is not selected (the property in the Web deployment descriptor). Because of that, when accessing snoop, it will still prompt for BASIC authentication login, as in Figure 6-18.

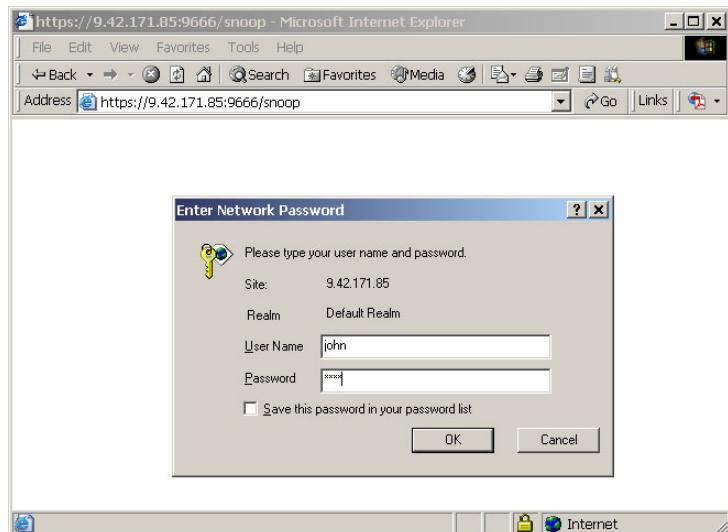


Figure 6-18 The snoop servlet application prompting for basic authentication login

3. Enter login information; in our case it is:

User Name: john
Password: test

4. The snoop servlet displays various request related information. Scroll down to the HTTPS Information section. You can see that our certificate data is present client certificate chain information, as in Figure 6-19 on page 112. In the case when client certificate SSL is not used or if a client fails to pass a certificate, WebSphere will only return the Cipher suite specification as used in the HTTPS connections; there is no client cert chain displayed.

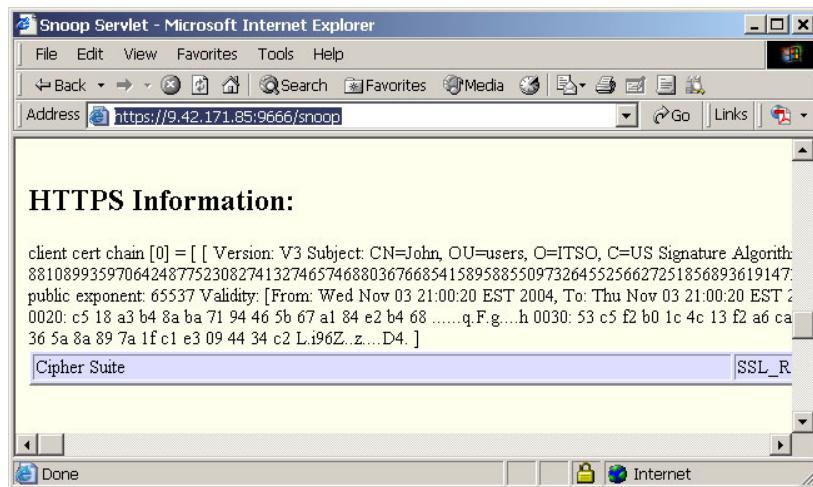


Figure 6-19 Response from the snoop servlet showing the client certificate

Now, let's create a test with our client certificate authentication enabled ITSObank Web sample application. Take the following steps:

1. Make sure that ITSObank application is installed and started.
2. Open a new browser window and access the following address from the client: `https://<your_server_name>:9666/itsobank/index.html`
3. Since client authentication is enabled, the server is going to request a client certificate during SSL handshake and consequently, your browser will open a pop-up window and prompt you to select a client certificate, as in Figure 6-20 on page 113. Select the right client certificate (you might have more than one installed in your browser) and click **OK**.
4. It depends on the browser type and settings, but it is very likely that you will get another pop-up for extra confirmation to receive the server SSL certificate. Click **OK**.

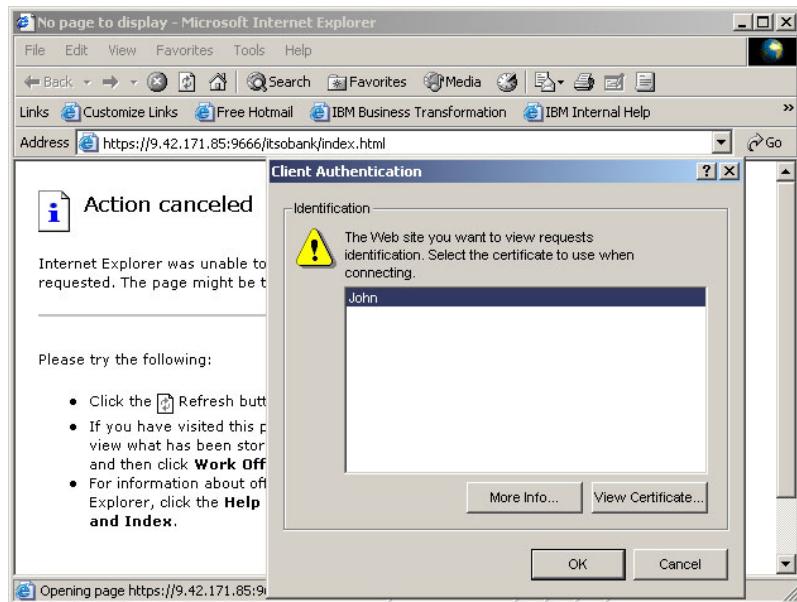


Figure 6-20 The browser prompts to select client certificate

5. If the SSL handshake went smoothly, WebSphere will map the data from client certificate and authenticate the user. If the user defined in your certificate is authorized to have access to ITSObank application, you will get the initial screen, as on Figure 6-21 on page 114.

As you can see, there was no login form screen displayed; WebSphere authenticated the user with the data stored in SSL client certificate. Also, you might still have left login and logout pages in your application. These are dysfunctional now.

Important: The Logout button does not function anymore; if you want to log out, you must close the browser window, otherwise, while it stays open, the browser will send the client certificate automatically when needed to access the application.

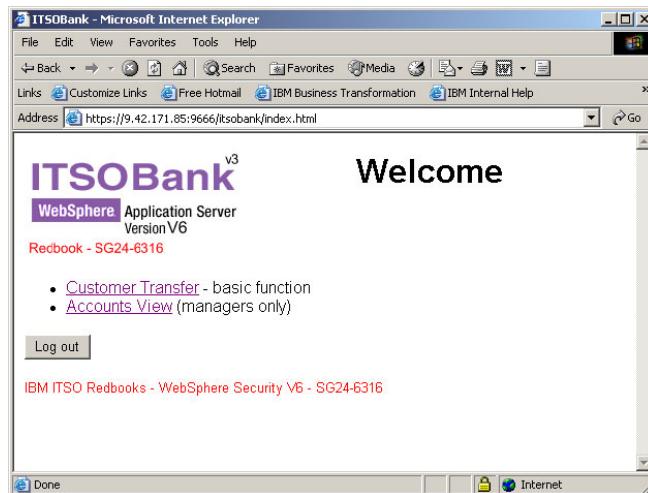


Figure 6-21 Successful client certification login to ITSOBank application

You can follow the operation of the authentication if you have tracing enabled for security. You should be able to find in your trace.log file something similar to the following example:

Example 6-25 trace.log for LDAP authentication test

```
...
WebConstraint > getConstraints: Entry
    /index.html
    GET
...
WebConstraint > getRequiredRoles : /index.html GET Entry
WebConstraint 3 Required roles are
WebConstraint 3 User
WebConstraint 3 Manager
WebConstraint 3 .
WebConstraint < getRequiredRoles Exit
WebAuthentica > authenticate Entry
WebAuthentica > handleSSO Entry
WebAuthentica < handleSSO: no cookies present in the request. Exit
WebAuthentica > handleCertificates Entry
WebAuthentica 3 Challenge type used is CERT.
WebAuthentica 3 Map credential for this certificate.
LdapRegistryI > mapCertificate Entry
LdapRegistryI > search Entry
LdapRegistryI 3 DN: o=itso,c=us
LdapRegistryI 3 Search scope: 2
LdapRegistryI 3 Filter: uid=John
LdapRegistryI 3 Time limit: 2
LdapRegistryI 3 Attr[0]: 1.1
LdapRegistryI > getDirContext Entry
LdapRegistryI < getDirContext Exit
LdapRegistryI < getDirContext Exit
LdapRegistryI 3 enterJNDI:Default : 1
LdapRegistryI 3 exitJNDI:Default : 1
LdapRegistryI 3 Time elapsed: 3
LdapRegistryI < search Exit
LdapRegistryI < mapCertificate Exit
LdapRegistryI > createCredential Entry
    cn=john,ou=users,o=itso,c=us
LdapRegistryI > search Entry
LdapRegistryI 3 DN: cn=john,ou=users,o=itso,c=us
LdapRegistryI 3 Search scope: 0
LdapRegistryI 3 Filter: (objectclass=*)
LdapRegistryI 3 Time limit: 0
LdapRegistryI 3 Attr[0]: uid
LdapRegistryI 3 Attr[1]: ibm-allgroups
LdapRegistryI > getDirContext Entry
LdapRegistryI < getDirContext Exit
LdapRegistryI < getDirContext Exit
LdapRegistryI 3 enterJNDI:Default : 1
LdapRegistryI 3 exitJNDI:Default : 1
LdapRegistryI 3 Time elapsed: 4
```

```

LdapRegistryI < search Exit
LdapRegistryI 3  uniqueUserId = cn=john,ou=users,o=itso,c=us
LdapRegistryI 3  displayName = john
LdapRegistryI 3  groups name = CN=HUMAN_RESOURCES,OU=GROUPS,O=ITSO,C=US
LdapRegistryI < createCredential Exit
    cn=john,ou=users,o=itso,c=us
WebAuthentica 3  Storing certificates in the credential
...
WebConstraint > getConstraints: Entry
    /transfer/accountsview.html
    GET
...
WebConstraint > getRequiredRoles : /transfer/accountsview.html GET Entry
WebConstraint 3  Required roles are
WebConstraint 3  Manager
WebConstraint 3  .
WebConstraint < getRequiredRoles Exit
WebAuthentica > authenticate Entry
WebAuthentica > handleSSO Entry
WebAuthentica 3  Looking for cookie name LtpaToken2
WebAuthentica > getCookieValues Entry
    LtpaToken2
...
WebAuthentica 3  The LTPA token was valid.
...
WebCollaborat A  SECJ0129E: Authorization failed for john while invoking GET
on default_host:itsobank/transfer/accountsview.html, Authorization failed, Not
granted any of the required roles: Manager
...

```

To get the traces, we used the following trace string for WebSphere:

```

com.ibm.ws.security.web.WebAuthenticator=all:com.ibm.ws.security.web.WebConstra
intsTable=all:com.ibm.ws.security.web.WebAccessContext=all:com.ibm.ws.security.
registry.ldap.LdapRegistryImpl=all:com.ibm.ws.security.registry.ldap.LdapRegist
ryImpl$LdapMonitor=all

```

You can see that we first requested the index.html page for which WebSphere checked what security roles are authorized to get the page. Furthermore, it extracted the user information from the SSL certificate matched it with the data in LDAP and checked if the user is in the required security role. Afterwards, we also tried to get another resource, /transfer/accountsview.html, but in the end could not get the authorization since the user was not in the required role.

Configuring WebSphere to use the exact DN mapping option

Using the Distinguished Name (DN) from the certificate to look up the user means that the directory structure where the user can be found has to match the DN exactly. For example, if the user DN is cn=john,ou=users,o=ITSO,c=US, then John has to be under the user's organizational unit (ou), ITSO, US country (c) in this order.

For this section, the prerequisite is that WebSphere is already configured to use the LDAP user registry which also contains some directory structure. Refer to Figure 2-2 on page 10 in 2.1, “User registries” on page 8 to see what directory structure is used in our case.

The following steps will show you how to configure WebSphere Application Server to use Exact Distinguished Name (DN) mapping.

1. Log in to the WebSphere Administration Console.
2. In the Global Security administration page, go to User registries section.
3. Click the **LDAP** link option to open the LDAP User registry configuration page.
4. In the Additional Properties section, click the **Advanced LDAP user registry settings** link option.
5. The Advanced LDAP user registry settings page will open. As shown in Figure 6-22 on page 118, set the following:
Certificate Map Mode: EXACT_DN.

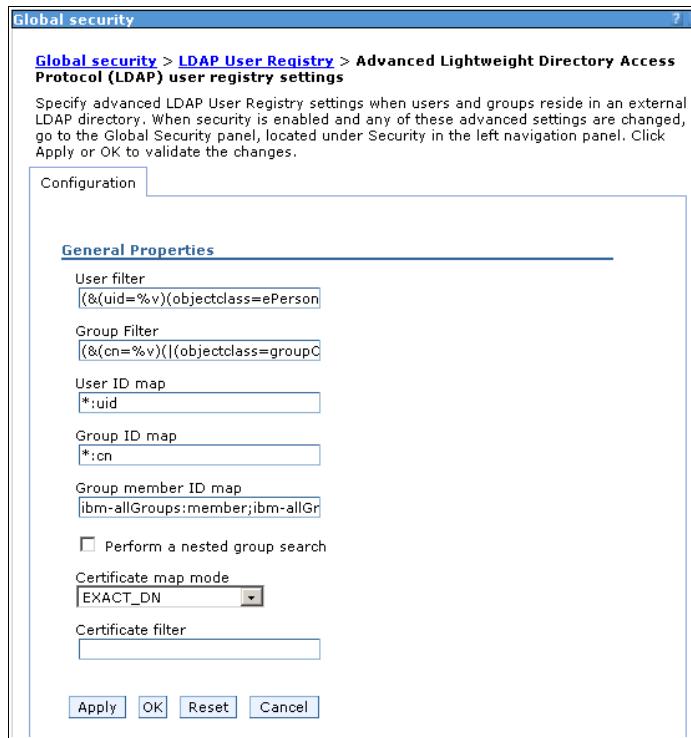


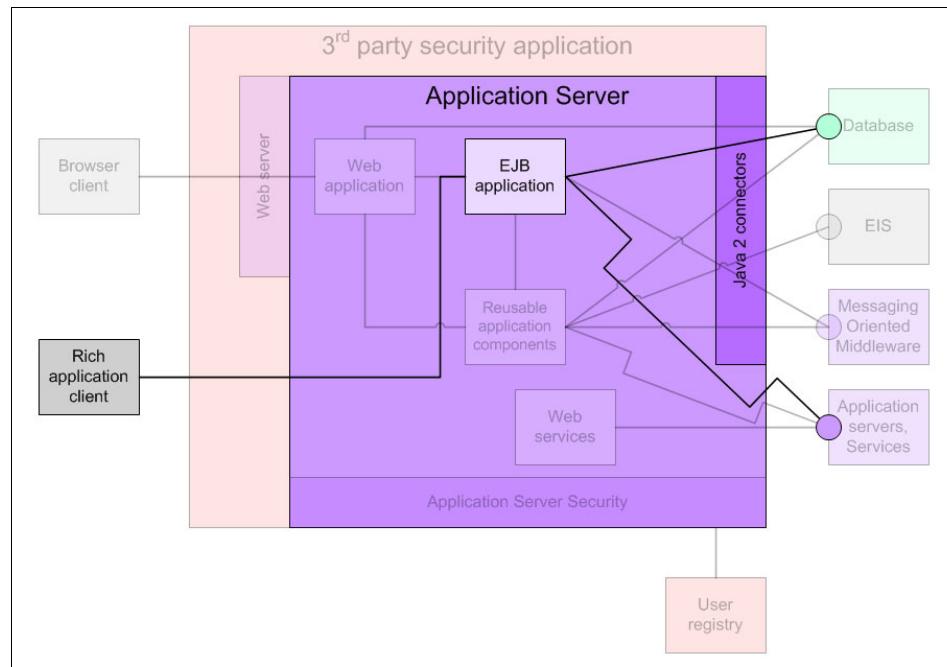
Figure 6-22 Setting the EXACT_DN client certificate map mode in WebSphere Application Server

6. Make sure that the Certificate Filter field is empty.
7. Click **OK**, then save the configuration for WebSphere.
8. Restart the application server in order for the changes to become effective.

For testing, use the same steps described previously with the certificate filter option in “Testing the client side certificate” on page 110.

You can follow the operation of the authentication if you have tracing enabled for security. Use the same trace string as in the previous section.

Securing an EJB application



7.1 Introduction

This chapter will discuss the security aspects involved within the EJB part of the Enterprise applications.

Throughout this chapter, we will see which processing components are usually involved when using EJBs that are running in WebSphere Application Server. The EJB container in WebSphere Application Server hosts the EJBs. We will discuss how to secure the transport channels to the EJB container, what are the authentication and authorization options available and how to configure EJB security on the J2EE level.

EJBs, or *Enterprise Java Beans*, are J2EE components that implement the business logic of an application. They typically have access to sensitive data, and it is very important to understand how security is applied to these resources.

There are three types of EJBs:

1. Session Beans, which represent clients inside the J2EE server. Clients call session bean methods to access an application.
2. Entity Beans, which represent persistent business objects in an application's relational database. Typically, each entity bean has an underlying table in the database, and each instance of the bean corresponds to a row in that table.
3. Message-driven Beans, which allow J2EE applications to process messages asynchronously. Message-driven beans' methods are invoked by the application server runtime as part of message queue processing.

Important: Since queued messages generally do not have any authentication information associated with them, authentication information is unavailable to message-driven beans' methods. As a result, securing message-driven beans from unauthorized access is really a matter of securing the message queue.

Security can be applied to EJBs in the following ways:

- ▶ Access control can be applied to individual session and entity bean methods so that only callers who are members of particular security roles can call those methods.
- ▶ Session and entity bean methods which need to be aware of the role or identity of the caller can programmatically call the J2EE API methods `isCallerInRole()` and `getCallerPrincipal()` to determine a caller's role and

principal, respectively. When using `isCallerInRole()`, the *security role references* are used, and these are later mapped to security roles.

Note: If WebSphere security is not enabled, or if the EJB is not a protected resource, `isCallerInRole` will return `false` and `getCallerPrincipal()` will return `UNKNOWN`.

- ▶ Session, entity, and message-driven bean methods can be delegated to execute under the identity of either the caller (default), the EJB server, or a specific security role. This is referred to as the *Delegation Policy* or *Run-As Mode Mapping*.

In the next sections, each of these methods of applying security to EJBs will be discussed in detail.

Throughout this section, we will use a simple EJB example (see Figure 7-1). It consists of two stateless session EJBs, Hello and SecuredHello; each has just one simple remote method which returns a hello message when invoked. Additionally, there is a servlet just to make the invoking of EJBs easier. We call the servlet from a browser.

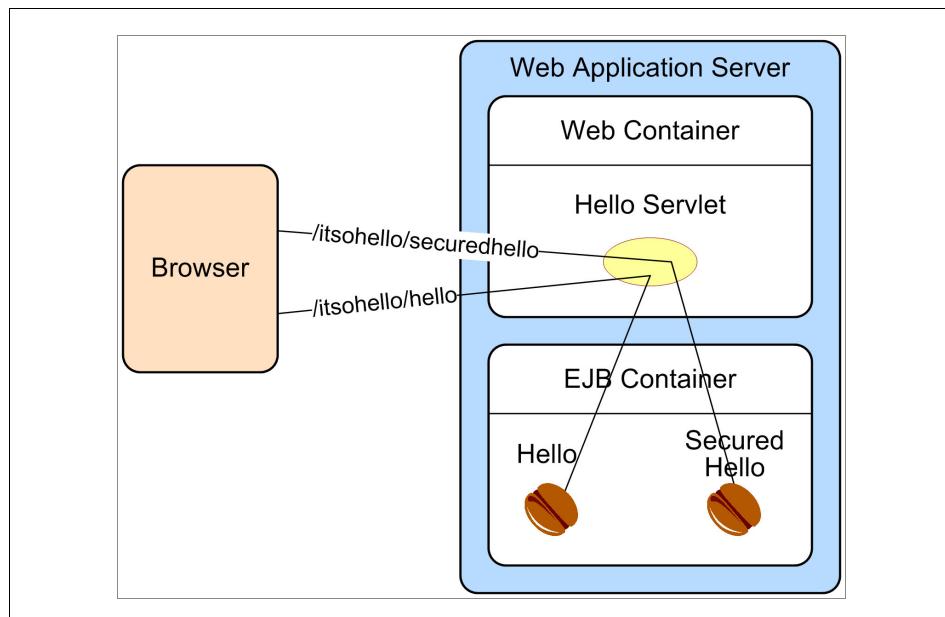


Figure 7-1 Sample application used throughout this section

7.2 Programmatic login (server-side) using JAAS

Programmatic login using JAAS is covered in 8.6.2, “Login process, programmatically” on page 173 and in more detail in 8.6.3, “Client-side programmatic login using JAAS” on page 174. Server-side login is very similar to the client-side login except that the login occurs on the server side, for example a servlet or an EJB performs the login. From the programmer’s point of view, it is the same thing, except that on the server side you cannot use login methods that require user interaction.

7.3 Declarative J2EE security

J2EE security can be applied declaratively or programmatically. WebSphere provides a security infrastructure for application security which is transparent to the application developer. That is, the developer does not need to code for security, since it will all be handled at deployment and runtime; this is called *declarative security*.

7.3.1 Defining J2EE security roles for EJB modules

The method for defining security roles for EJBs and Web components is very similar. To add a role named *BeanVisitor* to the EJB component, follow the steps below.

1. In the J2EE perspective, click **EJB Projects** → **ItsohelloEJB** to expand the tree.
2. Open the Deployment Descriptor of the **ItsohelloEJB** module.
3. The EJB Deployment Descriptor page opens; select the **Assembly** tab.
4. In the Security Roles section, click **Add** to add a new security role.
5. A pop-up window named *Add Security Role* will open. Enter *BeanVisitor* as the value for the Name field and click **Finish**.



Figure 7-2 New Security Role dialog box

A new <security-role> tag with your definition will be added to the assembly section of EJB deployment descriptor; see Example 7-1.

Example 7-1 Security role definition in EJB deployment descriptor

```
<assembly-descriptor>
    ...
    <security-role>
        <description>Authenticated guest for the EJB</description>
        <role-name>BeanVisitor</role-name>
    </security-role>
    ...
</assembly-descriptor>
```

7.3.2 Security role references

Security role references are used to provide a layer of indirection between security roles named in EJB Java code and security roles that are defined at application assembly time. This allows security roles names to be modified without requiring changes to the application code. We can divide security role reference usage process into three major parts (see Figure 7-3 on page 125):

1. Use security role reference in the EJB code

When an EJB uses the `IsCallerInRole(Java.lang.String roleName)` J2EE API method to determine whether or not the caller is a member of a particular role, `roleName` is a security role reference which is later linked to a defined security role in the EJB descriptor file, `ejb-jar.xml`. For example, the following Java code shows how a security role referenced might be used.

Example 7-2 Security role reference example

```
public String isInRole() {  
    if (mySessionCtx.isCallerInRole("RoleReference")) {  
        return "You are a member of the referenced role";  
    } else {  
        return "You are NOT a member of the referenced role";  
    }  
}
```

2. Reference definition

Every security role reference that is coded must be defined in the assembly descriptor and we use the XML tag <security-role-ref> for this purpose. See Figure 7-3 on page 125, step 2.

3. Reference link

At the application assembly time, all the defined security role references must be linked to one of existing security role definitions. The XML tag <role-link> specified within <security-role-ref> in ejb-jar.xml deployment descriptor defines the reference link.

The following XML code shows how the security role reference RoleReference can be linked to the security role BeanVisitor.

Example 7-3 Security role reference in ejb-jar.xml

```
<enterprise-beans>  
    <session id="SecuredHello">  
        ...  
        <security-role-ref>  
            <description>  
                The &quot;RoleReference&quot; string is mapped to BeanVisitor  
                security role</description>  
            <role-name>RoleReference</role-name>  
            <role-link>BeanVisitor</role-link>  
        </security-role-ref>  
        ...  
    </session></enterprise-beans>
```

For a security role reference to work, the security role to which it is linked must be a security role that is defined in the deployment descriptor and mapped to one or more users, groups, or special subjects.

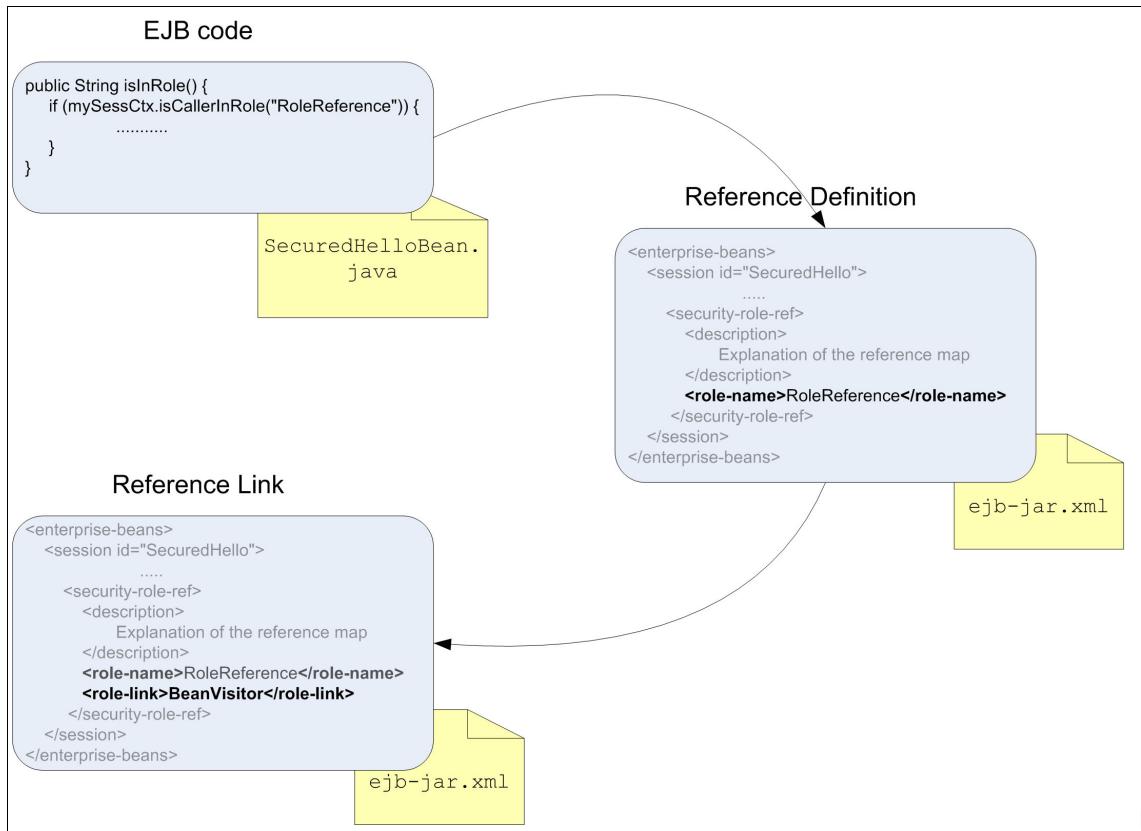


Figure 7-3 Security role references

Linking security role references

To define and link the `RoleReference` security role reference for the `BeanVisitor` security role using Rational Application Developer, do the following:

1. From the Resource Perspective, navigate to the EJB deployment descriptor file, `ejb-jar.xml`, and open this file.
2. Select the **References** tab.
3. Select the bean containing the method which calls the `isCallerInRole()` method and click **Add....** A pop-up window named *Add Reference* will open, as shown in Figure 7-4 on page 126.

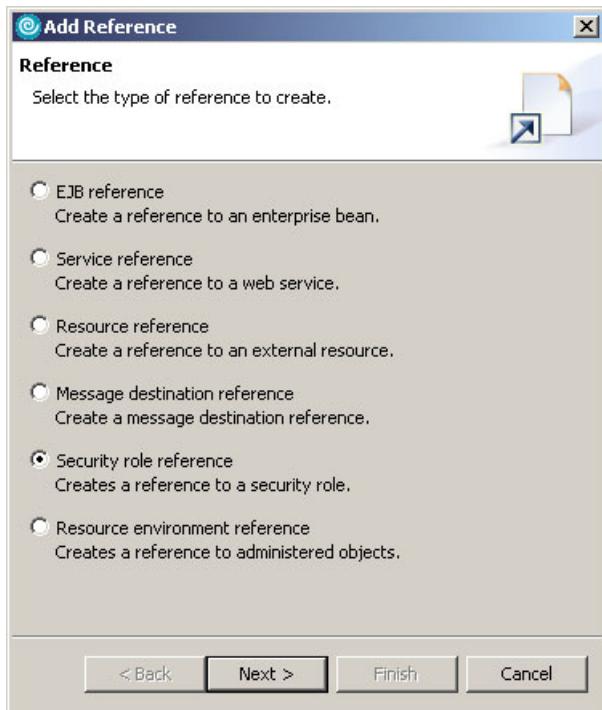


Figure 7-4 Adding security role reference

4. In the Add Reference dialog, select **Security Role Reference** and click **Next**.
5. In the Add Security Role Reference dialog, fill in the values as shown in Figure 7-5 on page 127. The reference's name is the string that is passed to `isCallerInRole()` method in the Java code.

The desired security role is selected from the Link pull-down menu. Only security roles which have previously been defined in the EJB module are shown in this menu.



Figure 7-5 Linking security reference role

You can also optionally enter a Description for this security role reference.

6. Click **Finish** to apply the changes and close the window. Save the deployment descriptor.
7. In Figure 7-6, you can see the Reference tab of the EJB deployment descriptor which shows added security reference role for the SecuredHello bean.

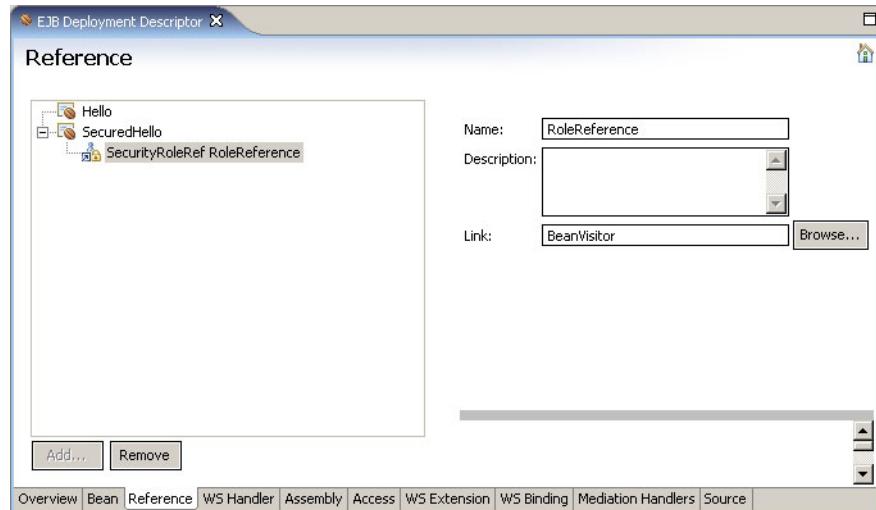


Figure 7-6 Reference tab of EJB deployment descriptor

7.3.3 Configuring method access control

Session and entity bean methods can be secured by preventing access to all but members of the security roles that need to access those methods.

The method permissions are included within the assembly descriptor tag in the application deployment descriptor file ejb-jar.xml. The following example shows the XML elements which would allow members of the BeanVisitor role to call all methods in the SecuredHello EJB, and members of the Anonymous role to call all methods on the Hello EJB and Method permissions in the ejb-jar.xml file.

Example 7-4 Role and method permission definitions in the ejb-jar.xml file

```
<assembly-descriptor>
    <security-role>
        <description>Authenticated guest for the EJB</description>
        <role-name>BeanVisitor</role-name>
    </security-role>
    <security-role>
        <description>Anybody who access the bean</description>
        <role-name>Anonymous</role-name>
    </security-role>
    <method-permission>
        <role-name>Anonymous</role-name>
        <method>
            <ejb-name>Hello</ejb-name>
            <method-name>*</method-name>
        </method>
    </method-permission>
    <method-permission>
        <unchecked />
        <method>
            <ejb-name>SecuredHello</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>getMessageUnprotected</method-name>
            <method-params>
            </method-params>
        </method>
    </method-permission>
    <method-permission>
        <role-name>BeanVisitor</role-name>
        <method>
            <ejb-name>SecuredHello</ejb-name>
            <method-name>*</method-name>
        </method>
    </method-permission>
</assembly-descriptor>
```

Assigning method permissions

To set up these method permissions using Rational Application Developer, follow the steps below.

1. Load the EJB project into the Rational Application Developer, in this example: ItsohelloEAR.ear.
2. Within the J2EE perspective, click **EJB Projects** → **ItsohelloEJB** to expand the tree.
3. Open the Deployment Descriptor for the itshelloEJB project. The EJB Deployment descriptor page opens. Switch to the Assembly tab.
4. Under the Method Permissions section, click **Add**. A pop-up window named *Add Method Permission* will open. As in Figure 7-7, we have an option to either select one of the existing security roles or select the **Unchecked** option. See “Assigning roles to unprotected methods” on page 131 for more details about the Unchecked option.

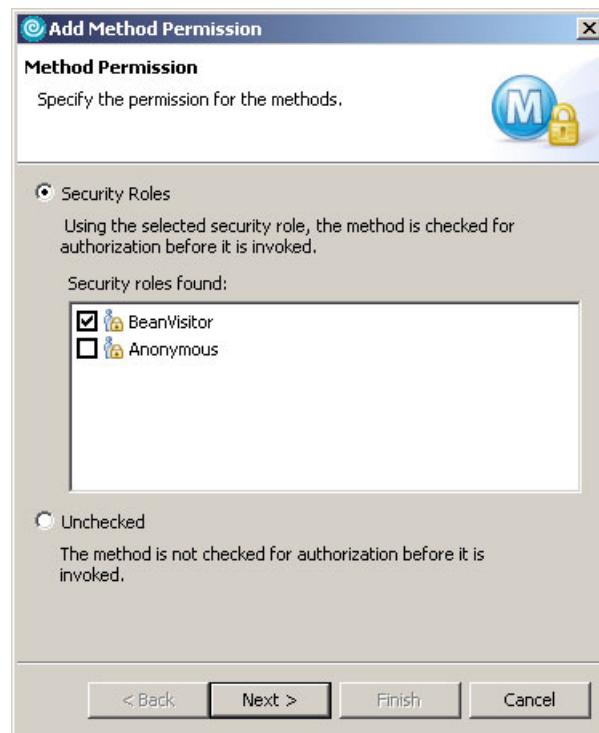


Figure 7-7 Adding method permission for defined security role

5. Choose the **Security Roles** option and select the **BeanVisitor** role. Click **Next** to see the list of EJBs.

6. Select the EJBs on which you want to configure method permissions for the selected security role. Choose **SecuredHello** and click **Next** to see the list of methods.



Figure 7-8 Selecting EJBs for configuring method permissions

7. Select one or more methods that you want to be accessible by a selected security role. You can use the wildcards (*) if desired to include all methods of a given type or all methods for a given EJB.

In this example, we selected all methods. Thus, the *BeanVisitor* security role gets access to all SecuredHello EJB methods. For now, only the users which will be mapped to the *BeanVisitor* security role will have access to this EJB; see Figure 7-9 on page 131.

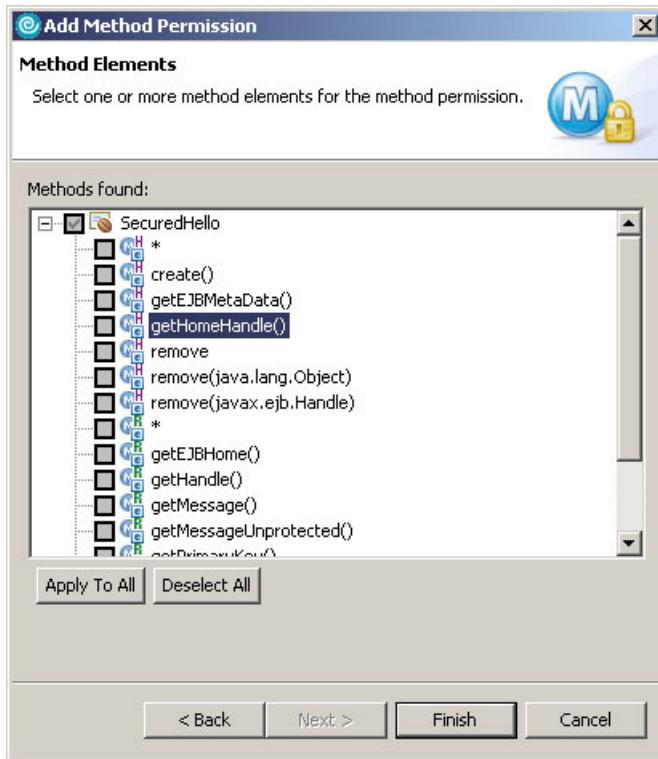


Figure 7-9 Selecting EJB methods

8. Click **Finish** when done.

Assigning roles to unprotected methods

During application installation, the WebSphere Administrative Console allows you to specify what method permissions are applied to session and entity EJB methods that are not explicitly secured in the deployment descriptor. If all session and entity EJB methods are protected, this step is omitted.

Note: When assigning roles to EJB methods, methods can be specified using several types of wildcards to select all home methods, local methods, local home methods, remote methods, and so on. When installing an EJB containing methods that are protected using one method-type wildcard (for example, the home methods wildcard) but whose other methods are unprotected, the WebSphere Application Server does not prompt for how unprotected methods are to be secured. Instead, they are deselected.

These unprotected methods can have one of the three permissions applied:

- ▶ *Uncheck*. This is the default, and indicates that unprotected methods should be left unprotected. Anyone can call these methods.
- ▶ *Exclude*. Unprotected methods are unavailable to all callers.
- ▶ *Role*. Unprotected methods are available only to members of a specific security role.

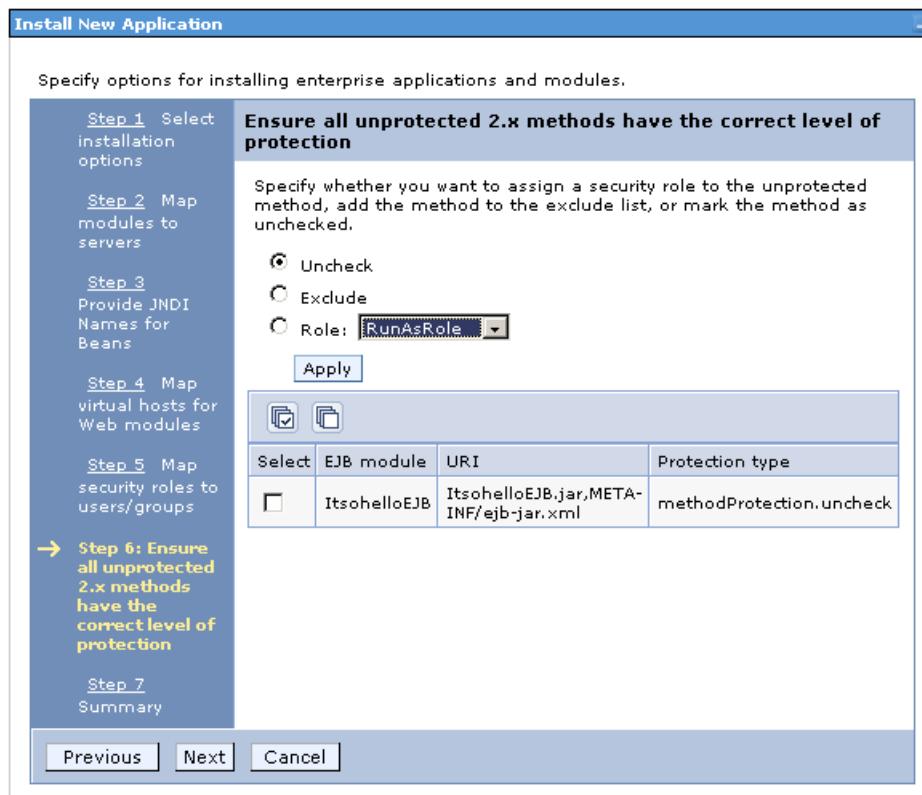


Figure 7-10 Assigning roles to unprotected methods

Note: The default behavior on EJB method protection is for methods that are not explicitly unprotected to be *unchecked*.

7.3.4 Enterprise Java Bean Run-As delegation policy

When an EJB calls a method in another EJB, the identity of the caller of the first EJB is, by default, propagated to the next. In this way, all EJB methods in the calling chain would see the same principal if they were to call the `getCallerPrincipal()` method. Occasionally, however, it is desirable for one EJB to call another with a previously defined identity, for instance one that is a member of a specific role.

For example, let us consider the message-driven bean's `onMessage()` method which calls a protected method in an entity bean. Since message-driven beans' `onMessage()` methods are executed with no caller identity, this method cannot call the protected entity bean method. By delegating the `onMessage()` method to run as a specific role, and adding this role to the protected entity bean method's access permissions, the `onMessage()` method can successfully access the protected method.

Important: Although this feature is commonly referred to as the *Run-as Mode*, it does not have any noticeable effect on the bean to which it is applied. A bean configured to run as a member of a given security role actually executes using the identity of the caller. It is only when calling methods in other EJBs that the Run-as Mode applies. These methods are called using the delegated identity.

7.3.5 Bean level delegation

The EJB 2.x specification defines delegation at the EJB bean level using the `<run-as>` element which allows the application assembler to delegate all methods of a given bean to run as a member of a specific security role. At deployment time, a real user that is a member of the specified role must be mapped to this role, through a process which is called *run-as role mapping*. All calls to other EJBs made by the delegated bean will be called using the identity of this mapped user.

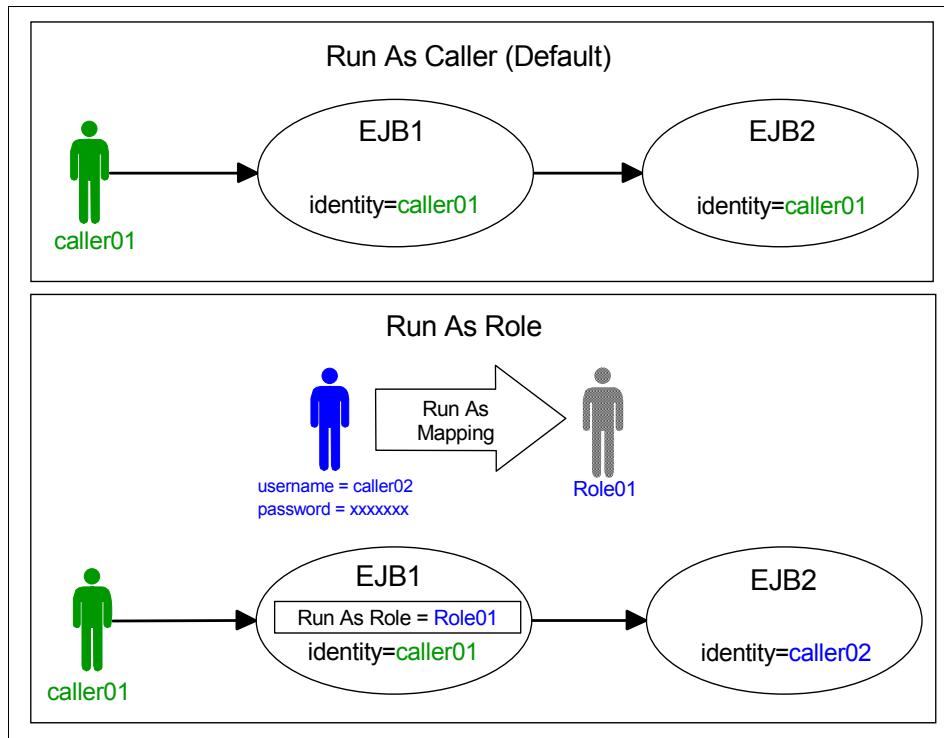


Figure 7-11 Run as Caller versus Run as Role

Figure 7-11 shows EJB delegation in contrast to the default Run-As Caller mode. In the top scenario, the identity of the caller, *caller01*, is propagated from EJB1 to EJB2. In the bottom scenario, EJB1 is delegated to run as *role01*. During run-as mapping, another user, *caller02*, is mapped to *role01*, and therefore it is effectively *caller02* that calls EJB2. If, in the bottom scenario, EJB2 were to call EJB3, EJB3 would also appear to have been called by *caller02*.

The following example shows the XML code in the ejb-jar.xml deployment descriptor file for the default mode (run as caller).

Example 7-5 ejb-jar.xml code for non-delegated EJB

```
<security-identity>
  <description>This bean requires no delegation</description>
  <use-caller-identity />
</security-identity>
```

The next example shows the XML code in the ejb-jar.xml file for a bean which has been delegated to run as a member of the *RunAsRole* security role.

Example 7-6 ejb-jar.xml code for EJB delegated to run as role mdbuser

```
<security-identity>
  <description>This EJB calls protected methods in other EJBs.</description>
  <run-as>
    <description>Methods of this EJB run as the RunAsRole role</description>
    <role-name>RunAsRole</role-name>
  </run-as>
</security-identity>
```

Assigning bean-level Run-as delegation policies

To assign a bean-level Run-as role to an EJB using Rational Application Developer, do the following:

1. Within the J2EE perspective, click **EJB Projects** → **ItsohelloEJB** to expand the tree.
2. Open the Deployment Descriptor for the ItsohelloEJB module. The EJB Deployment descriptor page opens. Switch to the Access tab.
3. In the *Security Identity (Bean Level)* section, click **Add**.
4. Select the desired Run-as mode, you can select either of the following two:

- **Use identity of caller**

If you select this option, the called EJB which we are calling from our bean will be called under our bean's identity. That will apply to all the methods in the called bean.

- **Use identity assigned to specific role (below)**

If you select this option, the called EJB which we are calling from our bean will be called under the specified role identity. That will apply to all the methods in the called bean.

Select the desired role from the options list. This list will contain all security roles which have been defined in the EJB module.

For example, select the specified role **Anonymous**. Enter an optional Role description and an optional Security identity description; see Figure 7-12 on page 136.

5. Click **Next**.

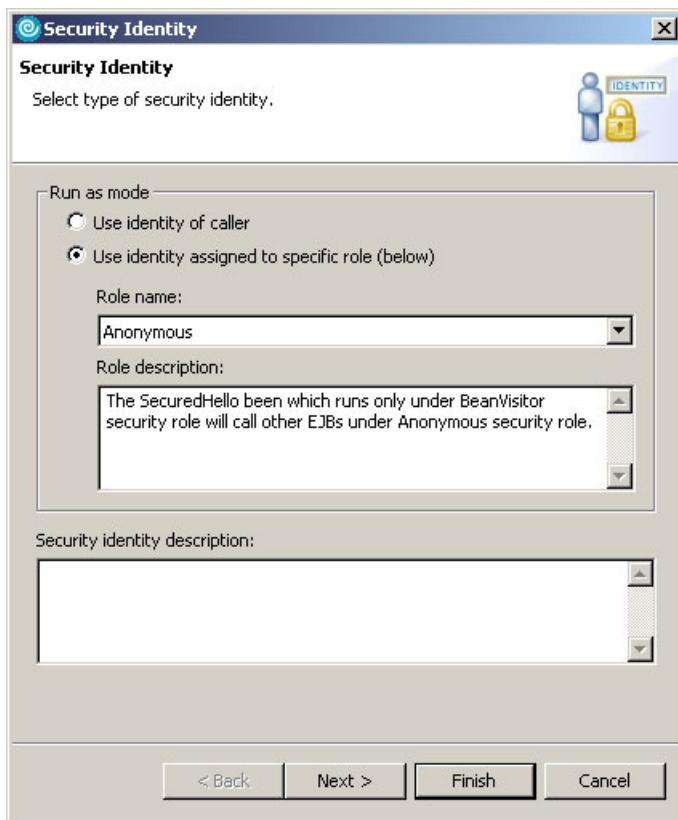


Figure 7-12 Assigning bean level run as delegation policy

6. Select one or more beans that should use this delegation policy, in this example, select **SecuredHello**.
7. Click **Finish**, save and close the deployment descriptor.

7.3.6 Method level delegation

In addition to the bean-level delegation policy defined by the EJB 2.x specification and described in the previous section, WebSphere Application Server provides additional capabilities to perform method-level EJB delegation. This works in the same way as bean-level delegation, but can be applied to specific EJB methods, rather than to the bean as a whole. This finer degree of delegation granularity allows application assemblers to delegate different methods of the same EJB to different security roles.

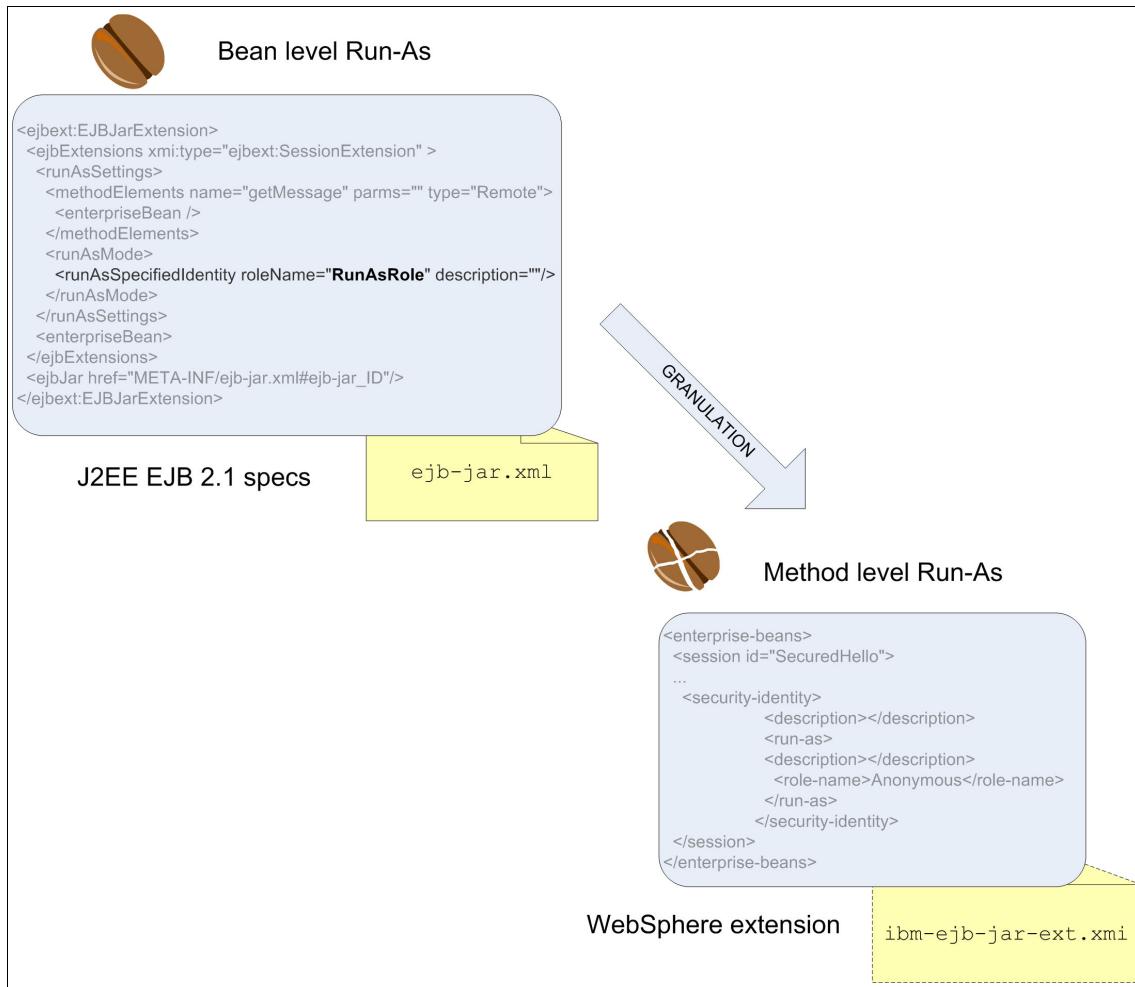


Figure 7-13 Method level Run-As delegation compared to Bean level Run-As delegation

In addition, method-level delegation provides an additional delegation option: *run as server*. This option indicates that the method should make calls to other EJBs using the identity of the application server itself.

Method-level delegation policies are defined in the ibm-ejb-jar-ext.xmi file. The following example shows the XML code for a getMessage() method which is delegated to run as the application server.

Example 7-7 Method-level run as server

```
<runAsSettings description="">
  <methodElements name="getMessage" parms="" type="Remote">
    <enterpriseBean xmi:type="ejb:Session" href="META-INF/ejb-jar.xml#Hello"/>
  </methodElements>
  <runAsMode xmi:type="ejbext:UseSystemIdentity"/>
</runAsSettings>
```

The following example shows the XML code for an getMessage() method which is delegated to run as a member of the *RunAsRole* security role.

Example 7-8 Method-level run as role

```
<runAsSettings>
  <methodElements name="getMessage" parms="" type="Remote">
    <enterpriseBean xmi:type="ejb:Session" href="META-INF/ejb-jar.xml#Hello"/>
  </methodElements>
  <runAsMode>
    <runAsSpecifiedIdentity roleName="RunAsRole" description="" />
  </runAsMode>
</runAsSettings>
```

Assigning method-level Run-as delegation policies

To assign a method-level Run-as role to an EJB using Rational Application Developer, do the following:

1. Within the J2EE perspective, click **EJB Projects** → **ItsohelloEJB** to expand the tree.
2. Open the Deployment Descriptor of the ItsohelloEJB module. The EJB Deployment descriptor page will open. Switch to the Access tab.
3. Scroll down to *Security Identity (Method Level)* section and click **Add**.
4. Select the desired Run-as mode; you can select one of the following:

- **Use identity of caller**

If you select this option, the called EJB methods which we are calling from our bean will be called under our bean's identity. That will apply just to the selected methods.

- **Use identity of EJB server**

If you select this option, the called EJB methods which we are calling from our bean will be called under EJB server identity. That will apply just to the selected methods.

– **Use identity assigned to specific role (below)**

If you select this option, the called EJB methods which we are calling from our bean will be called under specified role identity. That will apply just to the selected methods. Select the desired role from the option list. The specify role list will contain all security roles which have been defined in the EJB module.

If you choose this option, enter an optional Role description and an optional Security identity description.

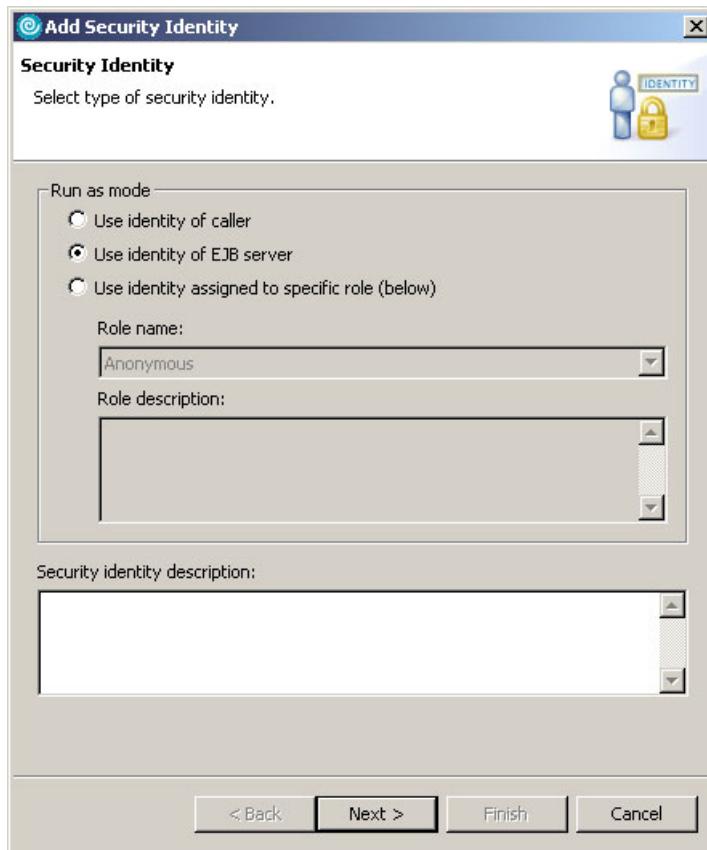


Figure 7-14 Method-level run-as role policy in Rational Application Developer

5. Click **Next**.
6. In the Enterprise Bean Selection dialog, select the EJBs containing the methods to which you want to assign this delegation policy, then click **Next**.
7. Under the Method Elements dialog, select the EJB methods to which this delegation policy should be assigned; see Figure 7-15 on page 140.

8. Click **Finish** and save the deployment descriptor changes.



Figure 7-15 Method Elements selection dialog

7.3.7 Run-as mapping

Run-as mapping refers to the process of assigning a real user from the user registry that is a member of the specified security role to the bean-level or method-level delegation policies. Run-as mapping is very different from, but easily confused with, security role mapping. The following table compares the two concepts.

Table 7-1 Run-as Mapping versus Security Role Mapping

Run-as Mapping	Security Role Mapping
Run-as mapping is used to determine the principal from the user registry that will be used as the caller identity when a delegated EJB makes calls.	Security role mapping is used to determine the users and groups from the user registry that will be considered members of the security role.

Run-as Mapping	Security Role Mapping
Run-as mapping associates a single user that is a member of the specified security role with a delegation policy.	Security role mapping associates one or more users or groups with a security role.
A single user name and password for the mapped identity is stored in the deployment descriptor.	One or more user names and/or group names are stored in the deployment descriptor.
Authentication done at installation time.	Authentication done at runtime.
Run-as mapping is performed using the WebSphere Administrative Console only.	Security role mapping is performed using the Application Server Toolkit, the WebSphere Studio, or the WebSphere Administrative Console.
Cannot be modified after application installation.	Can be modified after application installation using the WebSphere Administrative Console.

Important: The *Map Run-As roles to users* option appears in the WebSphere Administrative Console interface only when your application uses Run-as delegation. During the enterprise application installation, WebSphere detects whether this configuration exists at all and changes the user interface accordingly.

When installing an application which defines either a bean-level or method-level run-as role delegation policy, one of the steps will be to map the Run-as role(s) to a real user, as shown in Figure 7-16 on page 142.

1. Select the Role that you wish to map.
2. Enter a valid user name and password of a user in the registry that is a member of the specified security role.
3. Click **Apply** to authenticate the user and associate that identity with the Run-as role policy.
4. Once all Run-as roles have been mapped to real users, click **Next** to continue the installation.

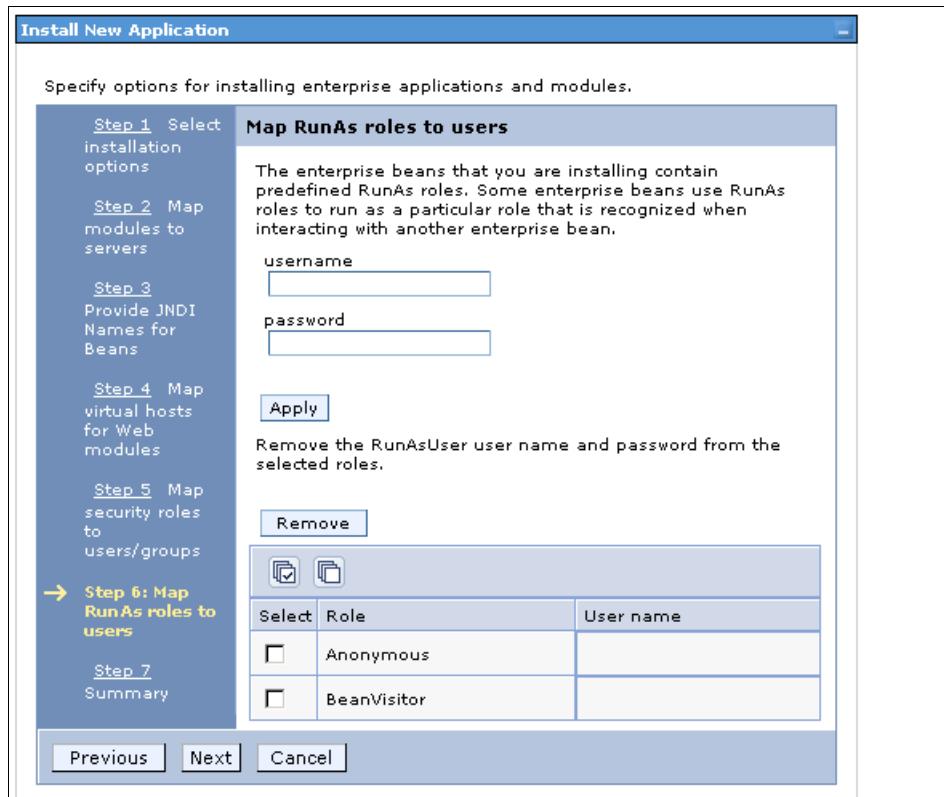


Figure 7-16 Run-as role mapping in WebSphere Application Server version 6

If one or more method-level delegation policies specify the Run-as system, one of the installation steps is going to be to verify this policy. The dialog appears as in Figure 7-16, and for each method that specifies the Run-as system, the application deployer can do one of the following:

- ▶ Do nothing, and allow the method to make calls using the system identity.
- ▶ Assign the method a Run-as role, and map the role to a user from the registry.

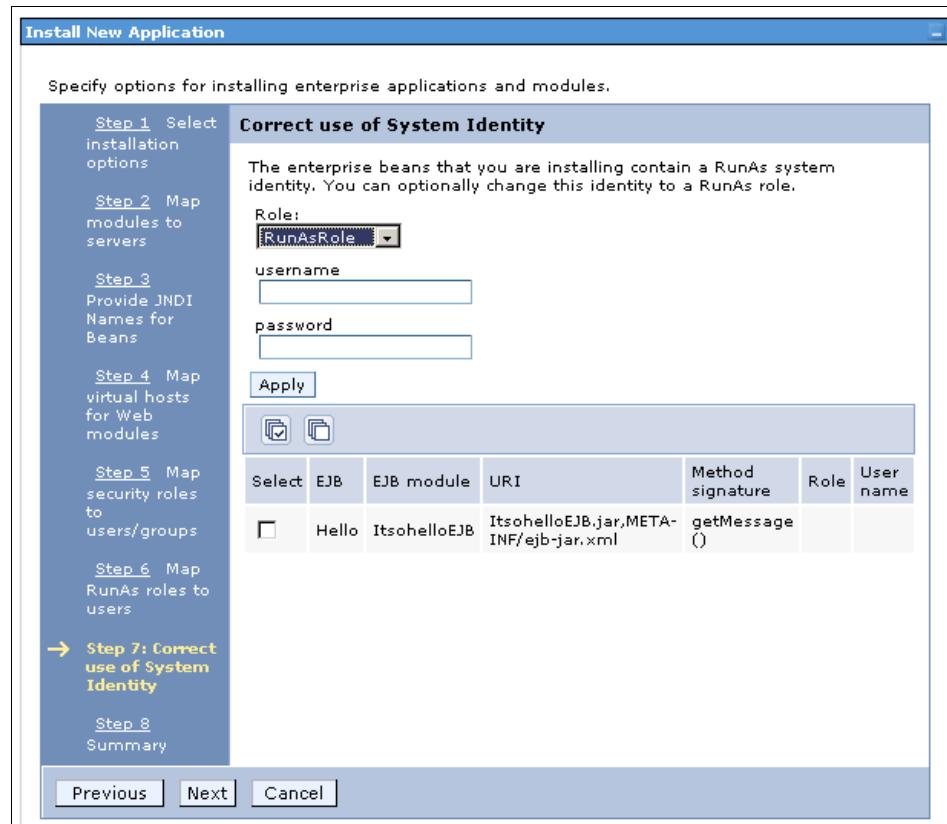


Figure 7-17 Verifying the Run-as system

To override the Run-as system mapping and assign a Run-as role, do the following:

1. Select the methods to which you want to assign the Run-as role.
2. Select the desired Role from the drop-down list of defined security roles.
3. Enter the valid user name and password of a user in the registry that is a member of the specified security role.
4. Click **Apply** to authenticate the user and associate that identity with the run-as role policy.
5. Click **Next** to continue with the installation.

7.4 Programmatic J2EE security

Programmatic security can be used by security-aware applications when declarative security alone is not sufficient to express the security model of the application.

Programmatic security becomes useful when the application server provided security infrastructure cannot supply all the functionalities needed for the application. Using the Java APIs for security, developers could implement security for the whole application without using the application server security functions at all. Programmatic security also gives developers the option to implement dynamic security rules for your applications.

Having said that, when developing servlets and EJBs, there are a few security calls available if the developer wants greater control of what the end user is allowed to do than is provided by the infrastructure.

7.4.1 EJB security methods

The EJB 2.x specification defines two methods that allow programmatic access to the caller's security context, javax.ejb.EJBCtxt.

- ▶ **java.security.Principal getCallerPrincipal()**

The *getCallerPrincipal* method allows the developer to get the name of the current caller. To do this, you need to call *getName()* on the *java.security.Principal* object returned.

```
EJBContext ejbContext;  
...  
// get the caller principal  
java.security.Principal callerPrincipal = ejbContext.getCallerPrincipal();  
// get the caller's name  
String callerName = callerPrincipal.getName();
```

The *Principal.getName()* method returns the login name of the user.

- ▶ **Boolean isCallerInRole(String roleName)**

The *isCallerInRole* method allows the developer to make additional checks on the authorization rights of a user which are not possible, or more difficult, to perform through the deployment descriptor of the EJB.

```
EJBContext ejbContext;  
...  
if (ejbContext.isCallerInRole("")){
    // Perform some function
} else {
    // Throw a security exception
}
```

The `isCallerInRole(String role)` method returns true if the user is in the specified role, and false if it is not. The role name specified in the method is really a security role reference, not a role. If the security role reference is not defined for the EJB, the method returns null.

Sample usage of security methods

The following example is a code snippet from the SecuredHelloBean as part of the ItsHelloEAR application. For more details, check the the original sample application.

Example 7-9 Sample code using the EJB security methods

```
public String getMessageUnprotected() {  
    return "[Not protected] Hello to you " + mySessionCtx.getCallerPrincipal();  
}  
  
public String isInRole() {  
    if (mySessionCtx.isCallerInRole("RoleReference")) {  
        return "You are a member of the referenced role";  
    } else {  
        return "You are NOT a member of the referenced role";  
    }  
}
```

With the security methods, the EJB will not let the user in a restricted role to submit a transfer greater than the maximum transferable amount.

7.5 EJB container access security

In the previous sections, we focused on EJB application security from the J2EE layer perspective. There are some more authentication and transport protection security mechanisms implemented by WebSphere working on the lower, CORBA, messaging layer.

7.5.1 CSIV2 and SAS

When a client component uses services from the WebSphere EJB container, all the communication goes through the RMI/IOP protocol. The client component can be either a standalone Java client, a J2EE client container application, or another EJB container; see Figure 7-18 on page 146.

WebSphere provides a security service which is compliant with Common Security Interoperability version 2, the CSIV2 protocol. There is another service called Security Attribute Service (SAS) which has been used in previous versions

before CSIV2. SAS (IBM) is deprecated and it is only kept to provide interoperability with WebSphere versions older than V5.0.

In short, providing Common Security Interoperability, WebSphere basically provides two important services:

- ▶ *Authentication* capabilites on the CORBA level.
- ▶ *Transport channel encryption*; WebSphere provides IIOP transport channel protection using the SSL protocol.

For more details about CSIV2, please refer to “CSIV2 Security Attribute Service (CSIV2 SAS)” on page 157.

In Figure 7-18, you can see a simple scenario; a J2EE client application needs to invoke some methods in an EJB which runs in Server A. Furthermore, Server A needs to run some methods in EJBs which run in Server B.

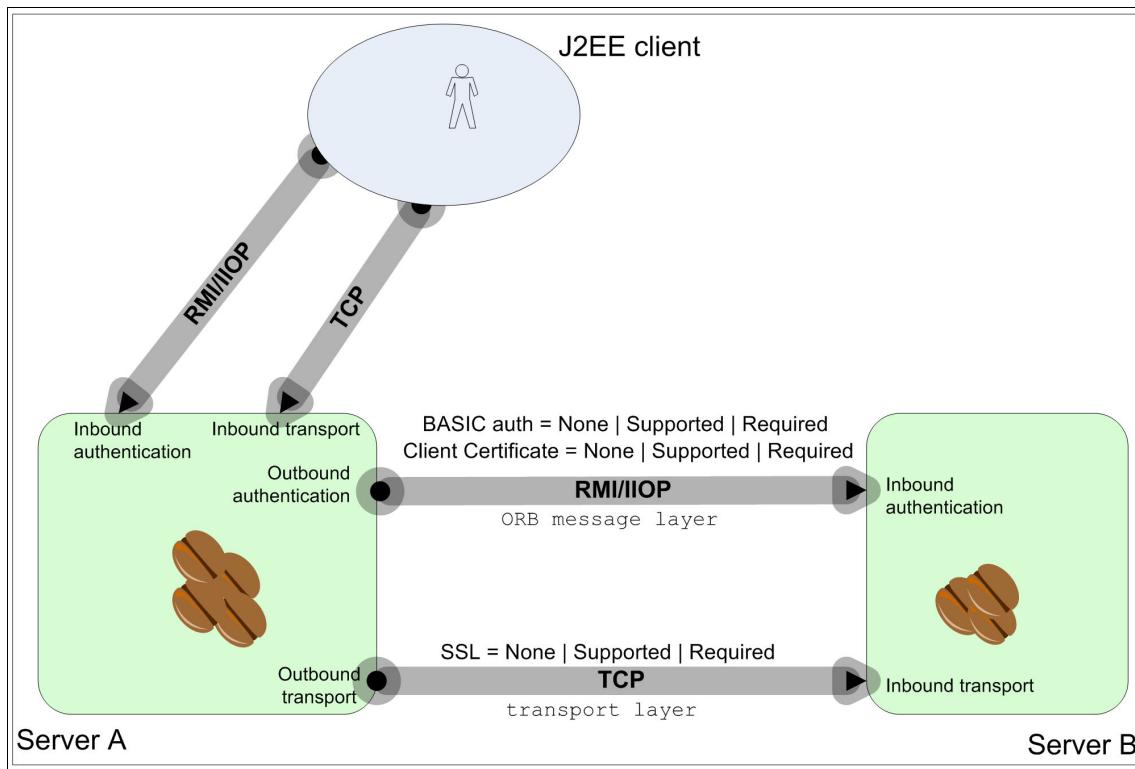


Figure 7-18 CSIV2 configurational options

7.5.2 Container authentication

When invoking EJB methods, the WebSphere Application Server environment determines the type of authentication required between the client and the server for each request. The following options are available:

- ▶ Basic Authentication

In this case, plain userid and password information is passed from client to server through the CORBA message layer.

For more details, refer to the *Message Layer authentication* document in the WebSphere Application Server 6 Information Center.

- ▶ Client Certificate Authentication

The client certificate authentication does not occur at the message layer as in the previous case, but occurs during the connection handshake using SSL certificates.

For more details, refer to the *Secure Sockets Layer client certificate authentication* document in the WebSphere Application Server 6 Information Center.

Because basic authentication and client certificate authentication occur at a different level, they can be set independently. For example, you can have both authentications be required or just basic authentication set and the other type supported.

Configuration of container authentication

The configuration of EJB container authentication can be done through the WebSphere Administrative Console:

1. In the *Global Security* administration page, go to the Authentication section.
2. Click **Authentication protocol** to display all the available options as shown in Figure 7-19 on page 148.
3. Container authentication can be set for *inbound* and *outbound* requests independently. Inbound means all the incoming communication that comes from a client to the server, outbound means all the outgoing communication that goes from the server toward other servers.

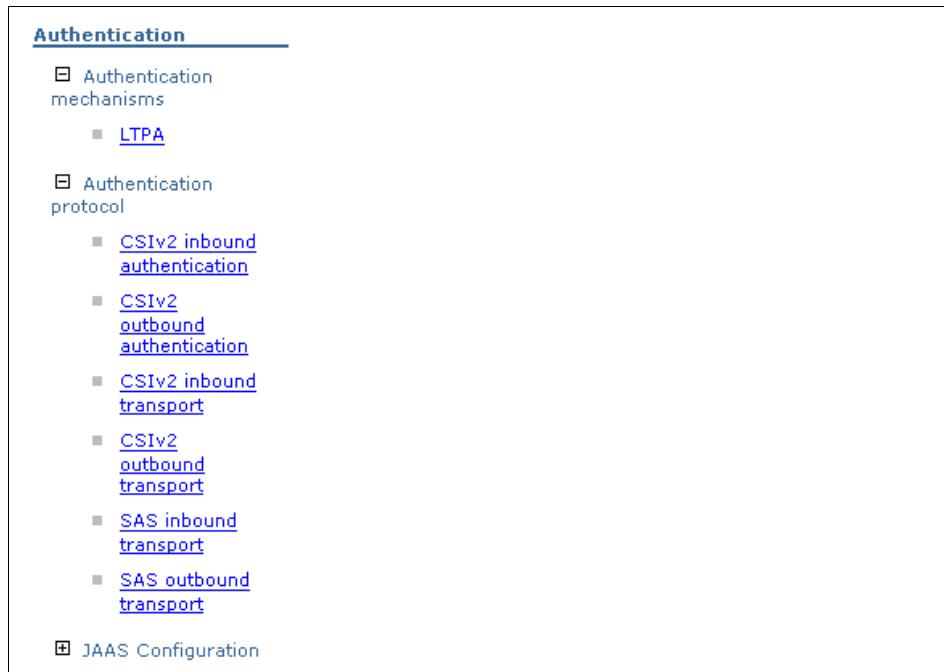


Figure 7-19 The Authentication section of Global Security administration page

4. After selecting the **CSIV2 inbound** or the **outbound authentication** page link, you will get a page as displayed in Figure 7-20 on page 149.

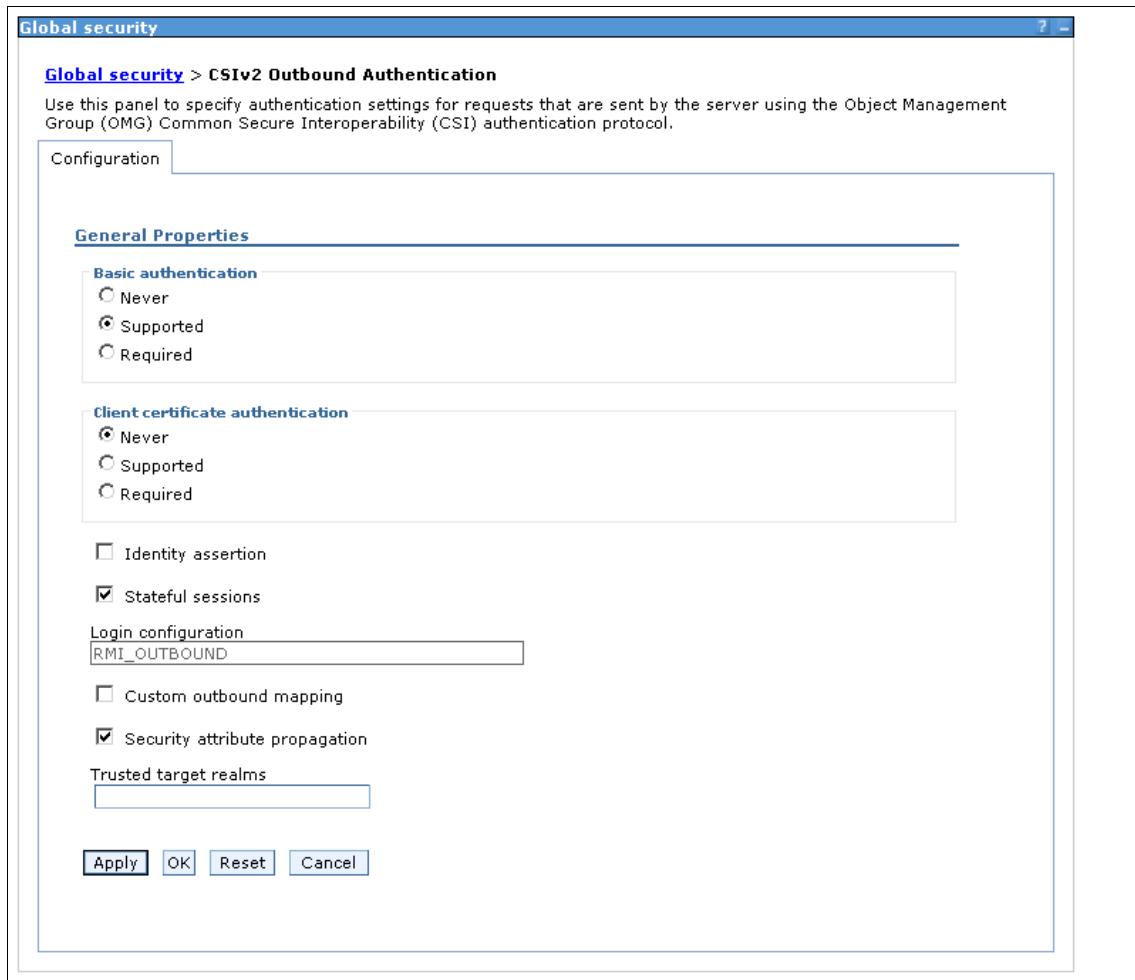


Figure 7-20 Setting CSIV2 Outbound authentication properties

5. Set Basic Authentication with one the following options:

– **Never**

Select **Never** to indicate that the server is not configured to accept message layer authentication from any client.

– **Supported**

Selecting **Supported** indicates that this server accepts basic authentication. However, other methods of authentication can occur if configured and anonymous requests are accepted.

- **Required**

Selecting **Required** indicates that only clients configured to authenticate to this server through the message layer can invoke requests on the server.

6. Set *Client certificate authentication* with one of the following options:

- **Never**

Selecting **Never** indicates that the server is not configured to accept client certificate authentication from any client.

- **Supported**

Selecting **Supported** indicates that the server accepts SSL client certificate authentication; however, other methods of authentication can occur (if configured) and anonymous requests are accepted.

- **Required**

Selecting **Required** indicates that only clients that are configured to authenticate to the server through SSL client certificates can invoke requests on the server.

To enable client certificate authentication for the IIOP transport layer, set the SSL to be required or supported. The prerequisite on the client side is that the client must have set a key database with a client certificate. As always, the certificate can be signed by a known Certificate Authority. Using an imported self-signed public key from the client is also an option, although is not recommended.

7.5.3 RMI/IIOP transport channel protection

When accessing EJB services, the client and server communicate through the Object Request Broker (ORB) service, using the IIOP protocol. Prior to any request flowing, a connection is established between the client ORB and the server ORB over the TCP transport. WebSphere provides the option of encrypting the connection using SSL. According to the connection encryption policies of both the client and the server, they negotiate the level of security for the connection used for the IIOP communication.

Configuring IIOP transport channel protection

The configuration of IIOP transport channel protection can be done using the WebSphere Administrative Console.

1. In the Global Security administration page, go to the Authentication section.
2. Select **Authentication protocol** to display all the available options as shown in Figure 7-19 on page 148.

3. Transport channel protection can be set for *inbound* and *outbound* transport independently. After selecting the **CSIV2 inbound** or the **outbound transport** page link, you will see a page as displayed in Figure 7-21.
4. Set *Transport* with one the following options:
 - **TCP/IP**
Server only supports TCP/IP and cannot accept SSL connections.
 - **SSL supported**
Server can support either TCP/IP or SSL connections.
 - **SSL required**
Any client communicating with this server must use SSL.

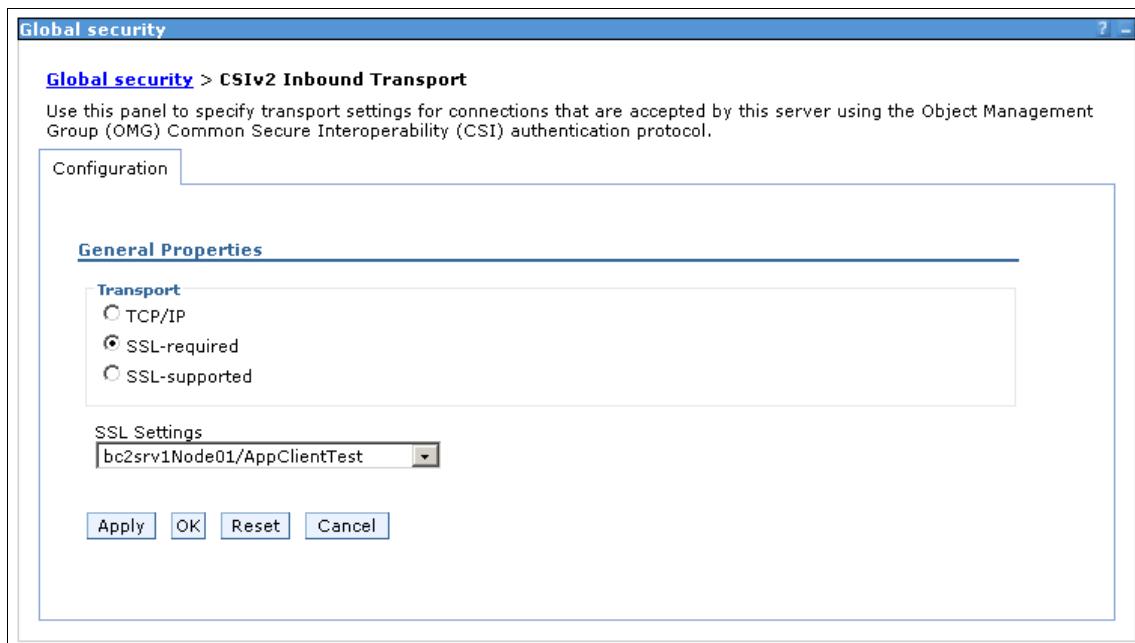


Figure 7-21 Setting CSIV2 Inbound transport properties

5. Configure *SSL Settings* by selecting one of the defined SSL repertoires from the drop-down list. For more information about SSL configuration, refer to 3.6, “SSL configurations” on page 45.

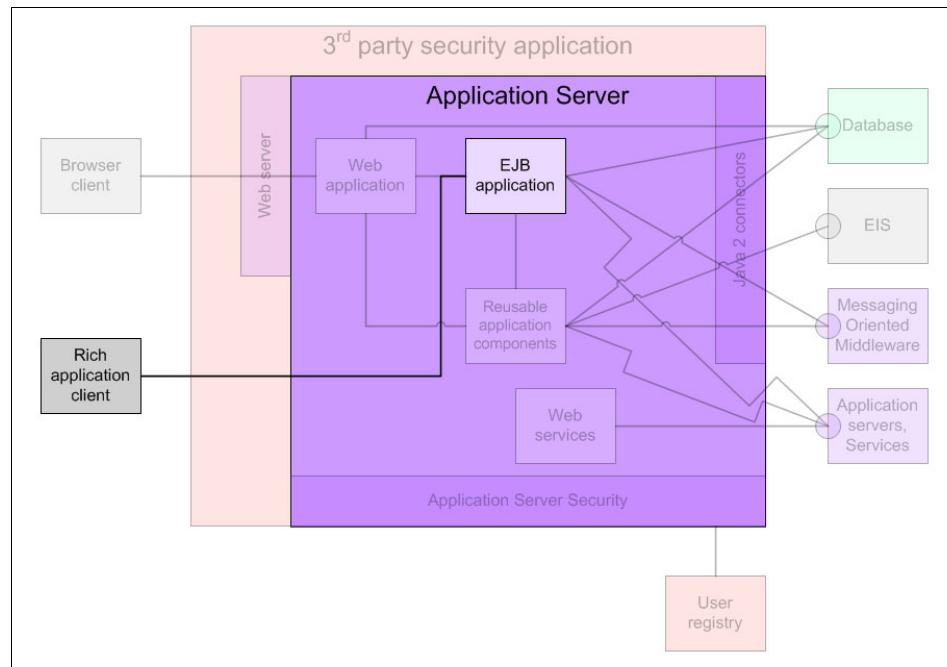
By default, the ORB transport listener ports are dynamically allocated during runtime. You might consider fixing the listener ports used for CSIV2. Since each application server runs its own ORB, they all have their own set of listening ports. The listener ports are managed by changing the application server's endpoints.

In this case, you need to specify CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS, CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS and ORB_LISTENER_PORT endpoints in order to fix the port numbers. Please check the WebSphere Application Server V6 Information Center for further details about how to configure the endpoints.



Chapter 8

Client security



8.1 Application clients in WebSphere

A *client* is a generic term used to refer to the process typically responsible for requesting a service. And the service itself is provided by the so called *server*. In this chapter, Java-based specific application clients accessing a remote enterprise bean server will be discussed. There are several important models of Java application clients supported by WebSphere Application Server version 6:

- ▶ **J2EE application client**, which uses the services provided by the J2EE client container. This is a Java application that may access any of the J2EE resources, such as: Enterprise Java Beans (EJB), Java Message Service (JMS) message queue and Java Database Connectivity (JDBC) using their Java Naming and Directory Interface (JNDI) names. Although the program follows the same model like other Java program, this application depends on the *Application Client runtime* (for example using <WebSphere_home\bin\launchClient.bat) to configure its execution runtime.
- ▶ **Thin application client**, which does *not* use the J2EE client container services. This means that when accessing a server's resources (for example EJB) from this client, the client should consider the object as a CORBA object instead of using the JNDI name. This lightweight Java client is useful whenever an existing Java application does not need all the overhead of the J2EE platform on the client machine.
- ▶ **Pluggable application client**, which is a kind of thin application client that uses a Sun JRE instead of the JRE that IBM provides.
- ▶ **Applet application client**, which a Java applet embeds in an HTML document resides in the client machine. This applet may access the resources in the WebSphere Application Server.
- ▶ **ActiveX application client**, which uses the Java Native Interface (JNI) architecture to access the JVM API. This implies that the JVM code exist in the same process space as the ActiveX application.

Further extensive information regarding the application clients and their capabilities can be found in the WebSphere Application Server InfoCenter. In this chapter, we are going to show in detail, with examples, on how the J2EE and thin clients access enterprise beans resources.

Itsohello client example

Throughout this chapter, we will use an example Itsohello application. Figure 8-1 on page 155 shows two enterprise beans: Hello and SecuredHello, as the core of the Itsohello application, installed in a WebSphere Application Server. These resources are accessible from different remote clients: users's browser (via the HelloServlet servlet), four J2EE Java application clients and four thin Java application clients.

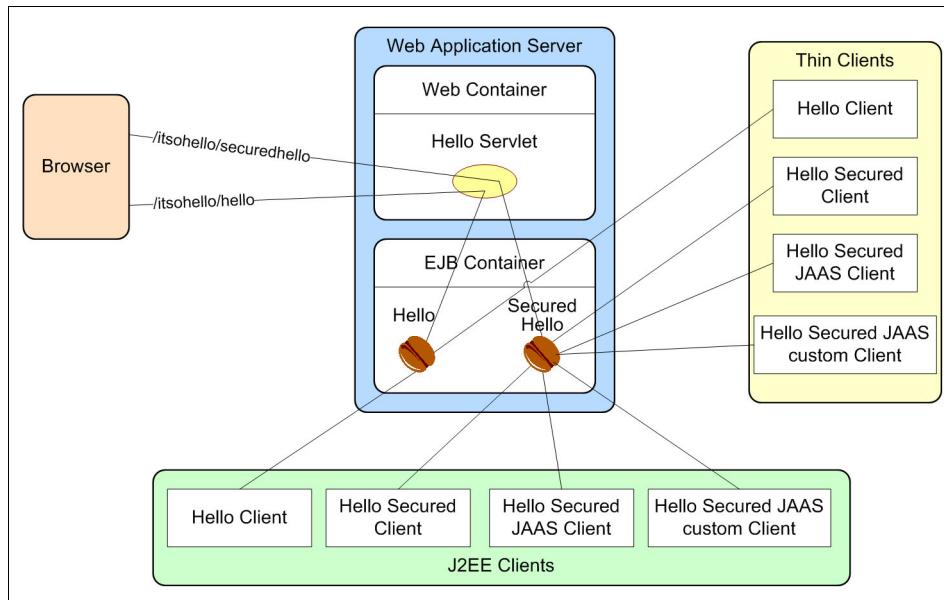


Figure 8-1 Interaction diagram of the example, Itsohello client applications, used for this chapter. Four J2EE application clients and four thin application clients, accessing secure and unsecure Hello beans in the EJB Container.

The components are described as follows:

- ▶ An enterprise application ItsohelloEAR.ear, installed in WebSphere Application Server. This .ear file contains a servlet (HelloServlet) which accesses two simple session beans: ejb/itsohello/hello (unsecure) and ejb/itsohello/securedhello (secure). The latter implies that only authenticated user allows to access the bean. To verify the installation, access the beans using your browser with these http addresses:
`http://<hostname>:<port>/itsohello/hello` for the unsecure session bean, and using `http://<hostname>:<port>/itsohello/securedhello` for the secure one. The default <port> number is 9080. “Hello” replies should be seen if the application is installed correctly.
- ▶ J2EE application clients (marked as J2EE Clients in the Figure 8-1). There are four J2EE application clients wrapped also in the ItsohelloEAR.ear file above:
 - HelloClient which is a J2EE client that accesses the unsecure hello bean in the EJB container directly, discussed in “Itsohello unsecure J2EE client” on page 165
 - HelloSecuredClient which a J2EE client that accesses the secure hello bean in the EJB container. See “Itsohello secure J2EE client” on page 166.

- HelloSecuredJAASClient. It behaves like the HelloSecuredClient, but the authentication process is controlled programmatically within the client. See “J2EE Java application client” on page 175.
- HelloSecureJAASClientC. Similar like HelloSecuredJAASClient but using a custom *CallbackHandler* for collecting authentication information. See “Custom CallbackHandler” on page 178.
- ▶ Thin application clients (marked as Thin Clients in the Figure 8-1 on page 155). It contains similar as the four J2EE application clients above but written as thin application clients. The application contains two jar files: ItsHelloTHINCLIENT.jar and ItsHelloEJB.jar with additional configuration and key files.

The installation process for that ItsHello application can be found in Appendix A, “Additional configurations” on page 387.

8.2 Java client authentication protocol

Accessing secure EJB resources in a secure WebSphere Application Server environment requires an authentication protocol to determine the level of security and the type of authentication between the client and the server. The authentication protocol will merge the server and client authentication requirements and come up with an authentication policy specific for them. This authentication policy will, among others, determine the following.

- ▶ The kind of connection used, SSL or TCP/IP.
- ▶ If SSL is used then the strength of the encryption should be known.
- ▶ The way to authenticate the client, whether user ID and password combination or client certificate, etc.

In WebSphere Application Server version 6, there are two authentication protocols available: IBM’s Secure Authentication Service (IBM’s SAS) and the Common Secure Interoperability version 2 (CSIV2). IBM’s SAS is the only authentication protocol used by all WebSphere Application Server prior to version 5. The CSIV2, defined by the Object Management Group (OMG), is a standard protocol defined so that vendors can interoperate securely. It is considered as the strategic protocol and is implemented with more features than IBM’s SAS within the WebSphere Application Server version 6.

In preparation for a request to flow from client to server, two client and server side Object Request Brokers (ORBs) must establish a connection over TCP/IP (or SSL) transport layer. Internet Inter-ORB Protocol (IIOP) is the protocol used for handling the communication between these two ORBs object. The authentication protocols IBMs SAS and CSIV2, explained above, are add-on services for the IIOP.

Note: The IBM's SAS and CSIV2 authentication protocols, used in WebSphere Application Server, are add-on services to the standard IIOP protocol for handling communication between two ORBs. Within WebSphere Application Server version 6, the authentication protocol IBM's SAS is deprecated, but is still included for backwards compatibility.

CSIV2 Security Attribute Service (CSIV2 SAS)

The Common Security Interoperability version 2 specification is defined by the OMG (see <http://www.omg.org>). The specification defines the CSIV2 Security Attribute Service (CSIV2 SAS) protocol to address the requirements of CORBA security for interoperable authentication, delegation and privileges.

Note: Do not confuse the term IBM's SAS (Secure Authentication Service) and CSIV2 SAS (Security Attribute Service).

The CSIV2 SAS protocol is designed to exchange its protocol elements in the *service context* of a General Inter-ORB Protocol (GIOP) request and reply messages that are communicated over a connection-based transport. The protocol provides client authentication, delegation, and privilege functionality that may be applied to overcome corresponding deficiencies in an underlying transport. The CSIV2 SAS protocol facilitates interoperability by serving as the higher-level protocol under which secure transports may be unified.

The CSIV2 SAS protocol is divided into two layers:

- ▶ The authentication layer is used to perform client authentication where sufficient authentication could not be accomplished in the transport.
- ▶ The attribute layer may be used by a client to deliver security attributes, such as identity and privilege, to a target where they may be applied in access control decisions.

The attribute layer also provides the means for a client to assert identity attributes that differ from the client's authentication identity (as established in the transport or CSIV2 SAS authentication layers). This identity assertion capability is the basis of a general-purpose impersonation mechanism that makes it possible for an intermediate to act on behalf of some identity other than itself. This can improve the performance of a system since the authentication of a client is relatively expensive. The server can validate the request by checking its trust rules.

Authentication process

Authentication is a process of establishing whether a client: a user, a machine or an application, is valid or not. The authentication process between client and server ORBs is shown in Figure 8-2 on page 159. The process can be summarized as follows:

1. Client ORB calls the connection interceptor to create the connection.
The client ORB invokes the authentication protocol's client connection interceptor. It is used to read the tagged components in the Interoperable Object Reference (IOR) of the server-based object being requested. This is how the authentication policy is established. Once the policy has been established, the ORB will make the connection, with the optional addition of the SSL cipher.
2. Client ORB calls the request interceptor to get client security information.
The client ORB invokes the client request interceptor once the connection has been established and sends security information other than what was established by the transport. This may include one of the following:
 - A user ID and password token (authenticated by the server)
 - An authentication mechanism-specific token (validated by the server)
 - An identity assertion token (allows an intermediate to act on behalf of some identity other than itself)This additional security information is sent with the message in a GIOP's *service context*. Once the client request interceptor finishes adding the service context to the message, the message is sent to the server ORB.
3. Server ORB calls the request interceptor to receive the security information, authenticate, and set the received credential.
Upon receiving the message, the server ORB invokes the authentication protocol's server request interceptor, which looks for the *service context*.
 - In case *service context* is found.
A method is invoked to the security server to validate the client identity. When the client identity is valid, a *credential* is returned. This credential contains additional information about the client, retrieved from the used user registry; and is used for authorization process. The authorization process determines whether the user is allowed to access an EJB resource or not.
 - In case *service context* is not found.
The server request interceptor looks at the transport connection to see if a client certificate chain is supplied. This is the case when SSL client authentication is configured between the client and server. If such a certificate is found, the distinguished name (DN) is extracted and is mapped to an identity in the selected user registry.
 - If the certificate does not map, no credential is created and the request is rejected.

- If the certificate maps, but the presented security information is invalid, the method request is rejected and an exception is sent back with the reply.
 - If the certificate maps, but no security information is presented, an *unauthenticated credential* is created. Only EJB methods with no security roles or EJB methods with a special *Everyone* role can be accessed using this unauthenticated credential.
4. Server ORB calls the request interceptor so that security can send information back to the client with the reply.
When the method invocation is completed, the server request interceptor is invoked again to complete the server authentication and a new reply service is created to inform the client request interceptor of the outcome.
 5. Client ORB calls the request interceptor so that the client can clean up and set the session status as good or bad.
The client request interceptor receives a reply from the server. The CSIV2 SAS supports both *stateless* and *stateful* security context. Stateless context exists only for the duration of the GIOP request that was used to establish the context. Stateful context endures until they are discarded. If a stateful is used, only the first request between a client and server requires that security information is sent. All subsequent method requests need to send a unique context ID only and the server can look up the credential stored in its session table.

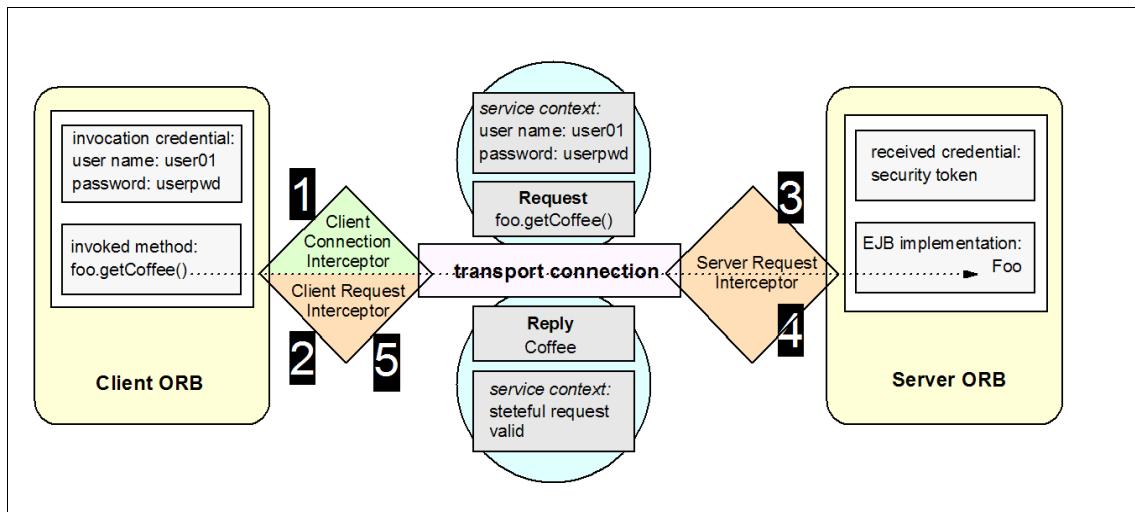


Figure 8-2 Authentication process

8.3 Java client configuration

As explained in 8.2, “Java client authentication protocol” on page 156, accessing secure EJB resources in a secure WebSphere Application Server environment requires an authentication protocol. This is needed to determine the level of security and the type of authentication between the client and the server such as the kind of connection used (for example SSL or TCP/IP), the strength of the encryption used, the type of authentication used (for example user ID/password or certificate) etc.

A Java client application accessing a secure EJB resources within WebSphere Application Server needs to specify those properties. These configuration properties are specified in a file defined by the system property com.ibm.CORBA.ConfigURL of the client's JVM, and can be found in the sample file sas.client.props. The Application Server should also be configured to communicate with a client in the required fashion. If a Java client requires that client certificates be transmitted via SSL, for example, then the server must be set to expect this.

Example 8-1 Starting Java client application com.ibm.Foo using CORBA configuration file properties/sas.client.props and using JAAS login configuration file properties/wsjaas_client.conf

```
java -Dcom.ibm.CORBA.ConfigURL=file:properties/sas.client.props  
-Djava.security.auth.login.config=file:properties/wsjaas_client.conf  
com.ibm.Foo
```

The sas.client.props file

The sas.client.props configuration file contains several sets of properties explained below. Default values are marked by *.

► Client Security Enablement

- com.ibm.CORBA.securityEnabled (true*, false). Determines if client security has been enabled. If the server's Global Security is enabled, the value of this property should be set to true, otherwise all the secured remote EJB resources cannot be accessed by the client.

► RMI/IOP Authentication Protocol

- com.ibm.CSI.protocol (sas, csiv2, both*). Determines which add-on authentication protocol is used.
 - both is used when communicating with all kind of WebSphere Application Server versions.
 - csiv2 is used when communicating with only servers versions 5.x or 6.x.
 - sas is used when communicating with only servers prior to version 5.x.

► Authentication Configuration

- `com.ibm.CORBA.authenticationTarget` (`BasicAuth*`). `BasicAuth` is the only supported option for pure Java client. The username and password will be send to the server for message layer authentication only. The SSL client certificate authentication should be configured under CSIV2 configuration.
- `com.ibm.CORBA.validateBasicAuth` (`true*`, `false`). Determines if the user details are authenticated immediately or deferred until the first method request is communicated to the server, when the `com.ibm.CORBA.authenticationTarget` property is set to `BasicAuth`.
- `com.ibm.CORBA.authenticationRetryEnabled` (`true*`, `false`). Determines whether a failed login should be retried. This also applies to stateful CSIV2 sessions and validations that have failed due to an expired credential. Only those failures which are known to be correctable will be retried. This option is valid when `com.ibm.CORBA.validateBasicAuth` is set to `true`.
- `com.ibm.CORBA.authenticationRetryCount` (an integer value, `3*`). Determines how many retries will be attempted for failed login when `com.ibm.CORBA.authenticationRetryEnabled` to be set to `true`.
- `com.ibm.CORBA.securityServerHost`. Name (or IP address) of the security server to validate the user ID and password.
- `com.ibm.CORBA.securityServerPort`. Port number of the security server.
- `com.ibm.CORBA.loginSource` (`prompt*`, `keyfile`, `stdin`, `none`, `properties`). Determines how the authentication request interceptor will log in if it does not find an invocation credential set.
 - `prompt` will display a window requesting a user name and password;
 - `keyfile` will extract the user details from the file specified by `com.ibm.CORBA.keyFileName`
 - `stdin` will display a command line prompt requesting user details
 - `none` should be selected if the client uses programmatic login
 - `properties` will retrieve the user details from the `com.ibm.CORBA.loginUserId` and `com.ibm.CORBA.loginPassword` properties.
- `com.ibm.CORBA.loginUserId`. The user ID used when the `com.ibm.CORBA.loginSource` property is set to `properties`.
- `com.ibm.CORBA.loginPassword`. The user password used when the `com.ibm.CORBA.loginSource` property is set to `properties`.
- `com.ibm.CORBA.keyFileName`. The location of the key file that contains a list of realm/userid/password combinations (see file `<WebSphere_home>/profile/default/properties/wsserver.key`). Used when the `com.ibm.CORBA.loginSource` property is set to `keyfile`.

- com.ibm.CORBA.loginTimeout (an integer within the range 0 and 600. default is 300). It is the amount of time, in seconds, that the login prompt will be available before the login will be considered invalid.

► **SSL Configuration**

- com.ibm.security.useFIPS (false*, true). Indicates that the client wants to be in FIPS (Federal Information Processing Standard) -approved cryptographic algorithms mode.
- com.ibm.ssl.contextProvider (IBMJSSE2*, IBMJSSE, IBMJSSEFIPS). It is the Java Secure Socket Extension (JSSE) provider used. Specifying IBMJSSEFIPS means that the client wants to be in FIPS-approved cryptographic algorithms mode, and the runtime uses the IBMJSSE2 provider in combination with the IBMJCEFIPS.
- com.ibm.ssl.protocol (SSL*, SSLv2, SSLv3, TLS, TLSv1). Determines which variety of the SSL and TLS protocols are used to perform transport-layer encryption.
- com.ibm.ssl.keyStoreType (JKS*, JCEK, PKCS12). It is the format of the SSL key store file.
- com.ibm.ssl.keyStore. (for example keys/DummyClientKeyFile.jks). It is the location of the SSL key store file, which has personal certificates and private keys.
- com.ibm.ssl.keyStorePassword. It is the password with which the key store file is protected.
- com.ibm.ssl.trustStoreType (JKS*, JCEK, PKCS12). It is the format of the SSL key trust file.
- com.ibm.ssl.trustStore (for example keys/DummyClientTrustFile.jks). It is the location of SSL key trust file.
- com.ibm.ssl.trustStorePassword. It is the password with which the key trust file is protected.

► **IBM's SAS add-on authentication protocol**

- com.ibm.CORBA.standardClaimQOPModels (low, medium, high*). It determines the quality of protection (in other words, the security level). If the server and client values differ then the highest value will be chosen and the connection will be initialized with this QOP property.

► **CSIV2 add-on authentication protocol**

Certain security properties have supported/required property pairs. The required properties take precedence over the supported properties pair. So, if the required property is enabled then communication with the server must satisfy this property.

- `com.ibm.CSI.performStateful` (`true*`, `false`). It determines whether the client supports the stateful or stateless session.
- `com.ibm.CSI.performClientAuthenticationRequired` (`true*`, `false`) and `com.ibm.CSI.performClientAuthenticationSupported` (`true*`, `false`).
When supported, message layer client authentication is performed when communicating with any server that supports or requires authentication. Message layer client authentication will transmit a user ID and password if the authenticationTarget property is BasicAuth, or it will transmit a credential token if the authenticationTarget property is one of the token-based mechanism (for example LTPA, Kerberos).
When required, message layer client authentication must occur when communicating with any server. If transport layer authentication property is also enabled (see below), both authentications are performed. However, the message layer client authentication takes precedence at the server side.
- `com.ibm.CSI.performTLClientAuthenticationRequired` (`true*`, `false`) and `com.ibm.CSI.performTLClientAuthenticationSupported` (`true*`, `false`).
When supported, transport layer client authentication can be performed and the client will send digital certificate to the server during the authentication stage.
When required, the client will only authenticate with servers that support transport-layer client authentication.
- `com.ibm.CSI.performTransportAssocSSLTLSRequired` (`true*`, `false`) and `com.ibm.CSI.performTransportAssocSSLTLSupported` (`true*`, `false`).
When supported, the client can use either TCP/IP or SSL to communicate with the server.
When required, the client will only communicate with servers that support SSL.
- `com.ibm.CSI.performMessageIntegrityRequired` (`true*`, `false`) and `com.ibm.CSI.performMessageIntegritySupported` (`true*`, `false`).
These properties only valid when SSL is enabled.
When supported, it can make SSL connection either with 40-bit ciphers or with digital-signing ciphers.
When required, the connection will fail if the server does not support 40-bit ciphers.
- `com.ibm.CSI.performMessageConfidentialityRequired` (`true*`, `false`) and `com.ibm.CSI.performMessageConfidentialitySupported` (`true*`, `false`).
These properties only valid when SSL is enabled.
When supported, it can make SSL connection either with 128-bit ciphers or with a lower encryption strength.

When required, the connection will fail if the server does not support 128-bit ciphers.

► **Additional CORBA configuration**

- com.ibm.CORBA.requestTimeout (integer value, 180*). This property specifies the timeout period, in seconds, for responding to requests sent from the client. Care should be taken when specifying this property, and set it only if the application is experiencing problems with timeouts.

For a more complete list of directives, refer to the WebSphere Application Server InfoCenter.

8.4 J2EE application client

A J2EE application client operates in a similar fashion to a J2EE server-based application. It makes use of the RMI-IIOP protocol and of CORBA services that are provided by the J2EE platform. This enables the J2EE application client to access to both EJB and CORBA object references. The J2EE platform allows the J2EE application client to use the JNDI names, defined in the deployment descriptor, to access the EJBs or other resources (JDBC, JMS, JavaMail, etc.).

The `ItsohelloJ2EEClient.jar` (wrapped in `ItsohelloEAR.ear` application) provided with this book has four J2EE application clients which request services to enterprise beans operating in a remote EJB container. The first two clients will be discussed in this section. The other two clients, which use JAAS APIs for the programmatic login, are discussed in the .

The J2EE client application depends on the *Application Client runtime* to configure its execution runtime. The `LaunchClient` command (`<WebSphere_home>\bin\launchClient.bat`) can be used to start and configure the J2EE application client environment by examining the application client's descriptor (`application-client.xml`). Access to the EJB jar file that contains the EJB's home interfaces, and access to the client's class file should be referenced in the client's `MANIFEST.MF` file.

Example 8-2 Client MANIFEST.MF which shows access to the main class file `com.ibm.itsohello.j2eeclient.J2EEClient` using EJB jar file `ItsohelloEJB.jar`

```
Manifest-Version: 1.0
Class-Path: ItsohelloEJB.jar
Main-Class: com.ibm.itsohello.j2eeclient.J2EEClient
```

Itsohello unsecure J2EE client

This unsecure Itsohello J2EE client application is a text based Java application which accesses the unsecure bean HelloBean in a remote EJB container. Upon successful execution, it will simply show a hello message created by the bean. The JNDI name for this bean is: ejb/itsohello/hello; see also Figure 8-1 on page 155. The code snippet below shows the part of the client for connecting to the bean and for getting the message.

Example 8-3 J2EE Client to unsecured Hello bean. Code snippet from com.ibm.itsohello.J2eeclient.HelloClient class

```
InitialContext ic = new InitialContext();
Object homeObject = ic.lookup("ejb/itsohello/hello");
HelloHome helloHome = (HelloHome) PortableRemoteObject.narrow(homeObject,
    HelloHome.class);
msg = helloHome.create().getMessage();
```

One thing important to see from the code snippet above is that there is no reference that indicates the server where the remote enterprise bean is located. As mentioned already, the *Application Client runtime* is responsible for this configuration. This is one of the nature of the J2EE application client.

The following steps describe how to start the application in order to access the unsecured bean:

1. Make sure the ItsohelloEAR.ear application has already been installed in the destination WebSphere Application Server. Among others, two Enterprise Java Beans with appropriate access privilege are automatically installed using JNDI names: ejb/itsohello/hello and ejb/itsohello/securedhello. These two enterprise beans resources are the remote resources for the example application clients. See also Figure 8-1 on page 155.
2. From a command prompt, use the following command to launch the J2EE client.

```
<WebSphere_home>\bin\launchClient ItsohelloEAR.ear
-CCBootstrapHost=<Server_hostname> -CCBootstrapPort=<RMICConnector_port>
```

The default value for <Server_hostname> is *localhost* and <RMICConnector_port> is 2809. You can also use the included batch file runJ2EEClient.bat for running the application. Do not forget to modify the parameters according to the setup of your system.

The application will show:

J2EE Itsohello clients:

- a. UNSECURED CLIENT.
...
b. SECURED CLIENT.

```
...
c. SECURED CLIENT with JAAS.
...
d. SECURED CLIENT with JAAS using custom callback handler.
...
Please enter your choice (a/b/c/d):
```

3. Press **a** and ENTER.

The client application will use `com.ibm.itsohello.j2eeclient.HelloClient` class to connect to the unsecure HelloBean. When finish, it will show this message:

```
Accessing unsecured Hello bean
Message from Hello bean: Hello to you UNAUTHENTICATED (role: Anonymous)
```

Note: If the Global Security is enabled, and the value of the property `com.ibm.CORBA.loginSource` is set to prompt in the file `sas.client.props`, the client will show a window requesting for a user identity and password, even if the bean is not secured. To disable the window, set the property `com.ibm.CORBA.loginSource` to none.

Itsohello secure J2EE client

This application is similar with the unsecure one above, except it shows a message created by the secure SecuredHelloBean in a remote EJB container. The JNDI name for this bean is: `ejb/itsohello/securedhello`. The code snippet below shows that there is no difference in client code for accessing a secure or unsecure enterprise beans.

Example 8-4 J2EE Client to secure Hello bean. Code snippet from `com.ibm.itsohello.j2eeclient.SecuredHelloClient` class

```
InitialContext ic = new InitialContext();
Object homeObject = ic.lookup("ejb/itsohello/securedhello");
SecuredHelloHome helloHome = (SecuredHelloHome) PortableRemoteObject.narrow
    (homeObject, SecuredHelloHome.class);
msg = helloHome.create().getMessage();
```

The procedure to start the included secure Itsohello example application is similar as the Itsohello unsecure J2EE client except, in step 2, SECURED CLIENT (option **b**) is now chosen. This will start the secure client `com.ibm.itsohello.j2eeclient.SecuredHelloClient`. When Global Security is enabled and the `com.ibm.CORBA.loginSource` property in the CORBA client configuration file (for example `sas.client.props`) is set to prompt, a window is displayed, prompting for user ID and password like in Figure 8-3 on page 167.



Figure 8-3 Challenge window

Once the client has been authenticated, the appropriate remote method in `com.ibm.itsohello.bean.SecuredHelloHome` will be invoked. When the access is successful, it will show something like:

```
Accessing Secured Hello bean
Message from Hello bean: [Secured] Hello to you viking (role: BeanGuest)
```

Note: There is no difference in the J2EE client code for accessing a secure or unsecure EJB resources, unless JAAS APIs are used. The behavior of the authentication process, is controlled by the client configuration file (for example `sas.client.props`).

8.5 Thin application client

The thin application client phrase refers to the Java client that is not running within the J2EE client container. It is a stand-alone Java application, that implements EJB clients connecting to a remote EJB container of WebSphere Application Server. Since it is not running under a J2EE client container, when resolving to an enterprise bean, the client must know the location of the name server and the fully qualified name used for the remote resource. Thin application client has to initialize, and to code explicitly, access to any of the services that the client might require. The process of developing a thin application client can be summarized as follows:

1. Initialize the `org.omg.CORBA.ORB` object.
2. Optionally, initialize the `org.omg.Cosnaming.NamingContextExt` if CosNaming is used.
3. Use the ORB object (or the derived `NamingContextExt` object) to get a reference to the enterprise bean using the fully qualified physical location of the enterprise bean in the name space. WebSphere Application Server provides a script: `<WebSphere_home>\bin\dumpNameSpace.bat` which is useful to find out the fully qualified physical location names. An example of such an output is:

```

...
=====
Name Space Dump
  Provider URL: corbaloc:iiop:localhost:2809
  Context factory: com.ibm.websphere.naming.WsnInitialContextFactory
  Requested root context: cell
  Starting context: (top)=mka0k1myNode01Cell
  Formatting rules: jndi
  Time of dump: Mon Nov 08 11:45:20 EST 2004
=====

=====
Beginning of Name Space Dump
=====

1 (top)
2 (top)/persistent          javax.naming.Context
...
38 (top)/nodes/mka0k1myNode01/servers/server1/ejb/itsohello
38                               javax.naming.Context
39 (top)/nodes/mka0k1myNode01/servers/server1/ejb/itsohello/hello
39                               com.ibm.itsohello.bean.HelloHome
40 (top)/nodes/mka0k1myNode01/servers/server1/ejb/itsohello/securedhello
40                               com.ibm.itsohello.bean.SecuredHelloHome
...

```

4. The rest of the code is similar with the J2EE client.

Note: The fully qualified name, for example `cell/nodes/mka0k1myNode01/servers/server1/ejb/itsohello/hello` above, can also be reached via *unqualified name*, with the help of the ORB method `string_to_object(corbaname:iiop:<host>:<port>/NameServiceServerRoot#ejb/itsohello/hello)`.

Running thin application client

There are certain configurations need to be set for running thin application client in order to operate in a secure environment. In case the WebSphere Application Clients product is installed, you can use the provided the *buildClientRuntime* tool (found in `<WebSphereClient_home>\bin\buildClientRuntime.bat`) to build the required components for the client. However, in case it is not installed, the components can be obtained using the installed WebSphere Application Server, as follows:

- ▶ Get the Java 2 Runtime Environment (JRE) that is provided by WebSphere, including the libraries under directories `<WebSphere_home>\java\jre\lib` and `<WebSphere_home>\java\jre\lib\ext`.

- ▶ Collect all application client runtime, properties and configuration files. For a secure environment, the JVM should point to a CORBA configuration file (for example file sas.client.props) using the JVM system property: com.ibm.CORBA.ConfigURL. Optionally, if the application client uses the JAAS APIs for login, the JVM should probably include the JAAS login configuration file indicated by the JVM system property java.security.auth.login.config.
- ▶ Collect the KeyStore and TrustStore files, in case SSL is used. These files are referred in the CORBA configuration file above.
- ▶ Collect some libraries from the WebSphere runtime library under directory <WebSphere_home>\lib. Not all of them is needed, but for Itsohello thin application client, the files below are needed.

activity.jar	admin.jar	bootstrap.jar	cluster.jar
ecutils.jar	emf.jar	idl.jar	iwsorb.jar
j2ee.jar	lproxy.jar	management.jar	naming.jar
namingclient.jar	ras.jar	runtime.jar	runtimefw.jar
sas.jar	securityimpl.jar	txClient.jar	txClientPrivate.jar
utils.jar	wccm_client.jar	wsexception.jar	wssec.jar

Example 8-5 Script for running ItsohelloTHINCLIENT application

```

set WAS_HOME=C:\WebSphere\AppServer
set SERVER_HOST=mka0k1my.itso.ral.ibm.com
set SERVER_PORT=2809

set CLASSPATH=.\.;.\prop\;itsohelloEJB.jar;itsohelloTHINCLIENT.jar
set JAVA_LIB=-Djava.ext.dirs=%WAS_HOME%\java\jre\lib;
               %WAS_HOME%\java\jre\lib\ext;%WAS_HOME%\lib

set CORBA_CONFIG=-Dcom.ibm.CORBA.ConfigURL=file:prop\sas.client.props
set LOGIN_CONFIG=-Djava.security.auth.login.config=file:prop\wsjaas_client.conf
set CLIENT_TRACE=-Dcom.ibm.CORBA.CommTrace=false

%WAS_HOME%\java\bin\java -cp %CLASSPATH% %CORBA_CONFIG% %LOGIN_CONFIG%
%CLIENT_TRACE% %JAVA_LIB% com.ibm.itsohello.thinclient.ThinClient %SERVER_HOST%
%SERVER_PORT%

```

Note: Do not run the application client using different JVM version than what is used by the destination server because different classes implementation might give unexpected results.

Itsohello unsecure thin client

All the thin application clients used, can be found in ItsohelloTHINCLIENT.jar file. To be able to run the code, you need also the EJB jar file: ItsohelloEJB.jar. The setup of the clients is exactly the same like the J2EE application clients, see 8.4, “J2EE application client” on page 164. However, the batch runThinClient.bat

is used to start the application client. Again, do not forget to modify the parameters according to the setup of your system.

There are two ways of programming thin application client to access a remote enterprise bean: using CosNaming with fully qualified resource name and using ORB method `string_to_object` with unqualified resource name. Both approaches is illustrated below:

► **Using CosNaming with qualified name**

Example 8-6 Code snippet of thin client to unsecure Hello bean using CosNaming. Notice that the fully qualified name is used here.

```
// initialize ORB object
java.util.Properties props = new java.util.Properties();
props.put("org.omg.CORBA.ORBClass", "com.ibm.CORBA.iop.ORB");
props.put("com.ibm.CORBA.ORBInitRef.NameService","corbaloc:iiop:" +
    serverHostname + ":" + serverPort + "/NameService");
props.put("com.ibm.CORBA.ORBInitRef.NameServiceServerRoot","corbaloc:iiop:" +
    serverHostname + ":" + serverPort + "/NameServiceServerRoot");
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init((String[]) null, props);

// get the home object
Object obj = orb.resolve_initial_references("NameService");
org.omg.CosNaming.NamingContextExt initCtx =
    org.omg.CosNaming.NamingContextExtHelper.narrow(obj);
Object homeObject = initCtx.resolve_str(
    "ce11/nodes/mka0k1myNode01/servers/server1/ejb/itsohello/hello"
);

HelloHome helloHome = (HelloHome) PortableRemoteObject.narrow(homeObject,
    HelloHome.class);
msg = helloHome.create().getMessage();
```

► **Using ORB `string_to_object` method with unqualified name**

With the help of the name server, the Hello bean can also be reached using unqualified name as follows:

Example 8-7 Code snippet of thin client to unsecure Hello bean using ORB `string_to_object`. Notice that the unqualified name is used here.

```
java.util.Properties props = new java.util.Properties();
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init((String[]) null, props);

// get the home object
String resourceName = "corbaname:iiop:" + serverHostname + ":" + serverPort +
    "/NameServiceServerRoot#ejb/itsohello/hello";
Object homeObject = orb.string_to_object(resourceName);
```

```
HelloHome helloHome = (HelloHome) PortableRemoteObject.narrow(homeObject,  
    HelloHome.class);  
msg = helloHome.create().getMessage();
```

Itsohello secure thin client

Exactly like J2EE application client, see “Itsohello secure J2EE client” on page 166, there is no difference in coding for secure or for unsecure client. The behavior of the application is determined by the client configuration file (for example sas.client.props) explained in 8.3, “Java client configuration” on page 160.

8.6 Programmatic login

In case it is needed to implement a custom login mechanism for the application client, for example because the provided security infrastructure cannot supply all the functionalities needed, a programmatic login using the Java Authentication and Authorization Service (JAAS) can be used. JAAS contains a collection of strategic authentication APIs which enable developers to create their own login module.

8.6.1 JAAS login module in WebSphere

The authentication process between a Java application client and a remote EJB is explained in “Authentication process” on page 158. In Figure 8-4 on page 172 below, the simplified authentication process within WebSphere Application Server is given again to indicate the role of JAAS:

1. Java Clients send the authentication information to the Enterprise JavaBean (EJB) authenticator module. The authentication information can be Basic Authentication (simply a user ID and password pair) or a credential token (for LTPA).
2. The EJB authenticator module pass the authentication information to the Java Authentication and Authorization Service (JAAS) login module.
3. The login module used the specified authentication mechanism: either LTPA or SWAM.
4. For validating the authentication information, the authentication module uses either LocalOS, LDAP or custom registry.
5. Once authenticated, the login module creates a JAAS *Subject* (`javax.security.auth.Subject`). This subject, besides having the user's realm (`getPrincipals()`), also contain a CORBA credential in its public

credential list attribute (`getPublicCredentials()`). This credential will be used by the authorization service to perform further access to any resources.

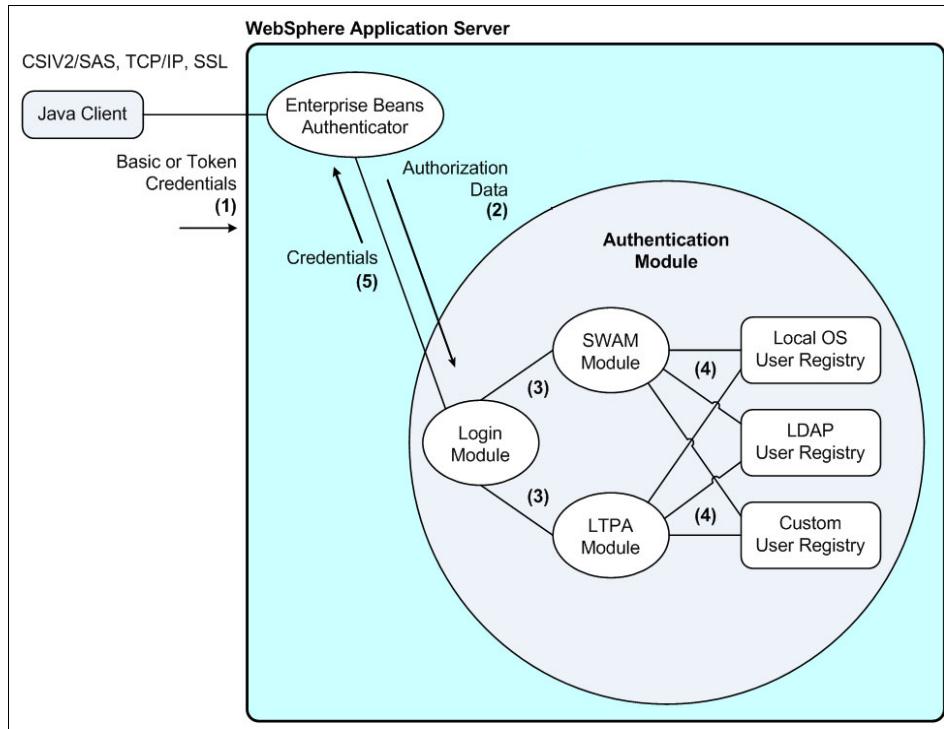


Figure 8-4 Authentication process within WebSphere Application Server. Note the importance of the JAAS login Module.

WebSphere Application Server provides JAAS login module for application, to perform programmatic authentication to the WebSphere Application Server security runtime. It has already several built-in JAAS login configurations that programmers can use directly, such as:

- ▶ *WSLogin*. This is a very generic JAAS login configuration that can be used by almost any application, including the Java application client, to perform authentication based on a user ID and password or a token.
- ▶ *ClientContainer*. Similar to *WSLogin*, this JAAS login configuration honors the *CallbackHandler* specified in the client container deployment descriptor. The login module of this login configuration uses the *CallbackHandler* in the client container deployment descriptor if one is specified, even if the application code specified one *CallbackHandler* in the *LoginContext*.

In WebSphere, the information of the supported built-in JAAS login configurations can be found in the file `wsjaas_client.conf`. This file should be

referred by the JVM runtime system property `java.security.auth.login.config` of the application client (see Example 8-1 on page 160).

8.6.2 Login process, programmatically

Programmatically, the login process and access to secure resource using the JAAS APIs can be explained using the interaction diagram in Figure 8-5

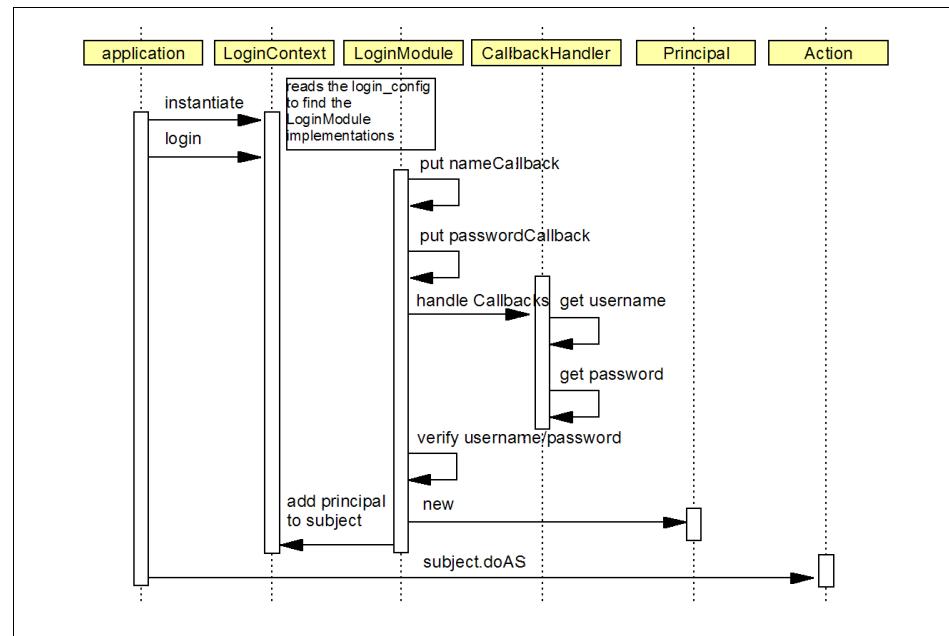


Figure 8-5 Interaction diagram for login process using JAAS APIs

- ▶ Application starts the login process.
 - `LoginContext` is initialized.
 - `LoginModule` is invoked. Depending on the design of the code, a user ID/password combination could be provided with the help of a created `CallbackHandler` object.
 - Upon successful verification of the supplied user ID and password, the `LoginModule` creates a `Subject` which contains the user's realm and a credential.
 - ▶ Application retrieves the created Subject from `LoginContext`
 - ▶ Using the `doAs` method, application invokes an `Action` under the acquired Subject.

Having read the description above, the Java code could be as simple as follows:

Example 8-8 Java code snippet for interaction diagram shown in Figure 8-5 on page 173

```
// login block
CallbackHandler loginHandler = new WSCallbackHandlerImpl("uid", "pwd");
LoginContext lc = new LoginContext("WSLogin", loginHandler);
lc.login();
Subject subject = lc.getSubject();

// create Action for accessing the protected bean method
java.security.PrivilegedAction getHelloMessage = new
    java.security.PrivilegedAction() {
        public Object run() {
            try {
                Object obj = ic.lookup("ejb/itsohello/securedhello");
                SecBeanHome hello = (SecBeanHome)
                    PortableRemoteObject.narrow(obj, SecBeanHome.class);
                return hello.create().getMessage();
            }
            catch (Exception e) {
                ...
            }
        }
}

// run the created Action with the acquired subject
msg = (String) com.ibm.websphere.security.auth.WSSubject.doAs
    (subject, getHelloMessage);
```

8.6.3 Client-side programmatic login using JAAS

A client-side login is useful when the user needs to login to a security domain on a remote system. However, this requires that both client and server use a same way on how to authenticate, as well as on how to collect the login information for authentication purposes. The JAAS interface `javax.security.auth.callback.CallbackHandler` defines how the security services may interact with the application to retrieve the authentication data.

Built-in CallbackHandler in WebSphere

WebSphere Application Server provides several class implementations of the `javax.security.auth.callback.CallbackHandler`. The following are the most useful `CallbackHandler` for *client-side programmatic login*:

- ▶ `com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl`. This implementation presents a graphical user interface login panel to prompt users for authentication data.

- ▶ `com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl`.
This callback prompts a user for authentication data which is useful for a text-based client application.

Whenever the above CallbackHandler implementations do not fulfill the user's requirement, a custom CallbackHandler implementation can be made. We will discuss this in "Custom CallbackHandler" on page 178

J2EE Java application client

The following code snippet shows how to perform a programmatic login using graphical user interface callback handler `WSGUICallbackHandlerImpl`. In case text-based login is preferred, the callback handler in the code should be changed into `WSStdinCallbackHandlerImpl`.

Example 8-9 J2EE client to secure Hello bean, using JAAS APIs. Graphical user interface login callback handler WSGUICallbackHandlerImpl is used. Change this to WSStdinCallbackHandlerImpl if the text-based version is preferred. Notice that there is no reference to which server this authentication should be validated; and neither for the enterprise bean

```
try
{
    ic = new InitialContext();

    // Invoke the JAAS Login module
    CallbackHandler loginCallbackHandler = new WSGUICallbackHandlerImpl();
    LoginContext lc = new LoginContext("WSLogin", loginCallbackHandler);
    lc.login();
    Subject subject = lc.getSubject();

    // create action to access the protected bean method
    java.security.PrivilegedAction getHelloMessage = new java.security.PrivilegedAction() {
        public Object run() {
            try {
                Object homeObject = ic.lookup("ejb/itsohello/securedhello");
                SecuredHelloHome helloHome = (SecuredHelloHome)
                    PortableRemoteObject.narrow(homeObject, SecuredHelloHome.class);
                return helloHome.create().getMessage();
            }
            catch (CreateException ce) {
                ...
            }
        }
    };
}

// invoke the secure action using the created subject
msg = (String) com.ibm.websphere.security.auth.WSSubject.doAs(subject, getHelloMessage);
}
catch (NamingException ne) {
    ...
}
```

The procedure to start the included secure Itsohello using JAAS APIs client example is similar as the Itsohello unsecure J2EE client except, in step 2, SECURED CLIENT with JAAS (option c) is now chosen. This will start the client com.ibm.itsohello.j2eeclient.SecuredHelloJAASClient. Since the login process is now controlled programmatically, the value of the com.ibm.CORBA.loginSource property in the CORBA client configuration file (for example sas.client.props) has no longer any influence.

Thin Java application client

Just like the J2EE application client, the JAAS programmatic login can be implemented for thin Java application client. The difference in the implementation can bee seen in the code snippet below.

Example 8-10 Thin application client to secure Hello bean. Notice how the security realm is established.

```
try {
    // initialize the ORB object
    orb = ORB.init((String[]) null, new Properties());

    // IMPORTANT: this is a dummy call to server to establish security realm for JAAS.
    // it should be done before the JAAS login
    orb.string_to_object("corbaname:iiop:" + serverHostname + ":" + serverPort);

    // Invoke the JAAS login module
    CallbackHandler loginCallbackHandler = new WSGUICallbackHandlerImpl();
    LoginContext lc = new LoginContext("WSLogin", loginCallbackHandler);
    lc.login();
    Subject subject = lc.getSubject();

    final String resourceName = "corbaname:iiop:" + serverHostname + ":" + serverPort
        + "/NameServiceServerRoot#ejb/itsohello/securedhello";

    // create action to access the protected bean method
    java.security.PrivilegedAction getHelloMessage = new java.security.PrivilegedAction() {
        public Object run() {
            try {
                Object homeObject = orb.string_to_object(resourceName);
                SecuredHelloHome helloHome = (SecuredHelloHome)
                    PortableRemoteObject.narrow(homeObject, SecuredHelloHome.class);
                return helloHome.create().getMessage();
            }
            catch (CreateException ce) {
                ...
            }
        }
    };
    msg = (String) com.ibm.websphere.security.auth.WSSubject.doAs(subject, getHelloMessage);
}
catch (LoginException le) {
    ...
}
```

The code snippet above shows the usual differences in programming between J2EE and thin application clients. However there is another difference, needed only when JAAS APIs programmatic login is used, which is the ORB method call:

```
orb.string_to_object("corbaname:iop:<serverHostname>:<serverPort>");
```

The above call is needed to establish connection to the security realm server. This is needed since the JAAS programmatic login needs to know where the security realm server is, to validate the user ID and password. So the above call should be done *before* the `LoginContext.login()` method is invoked.

Custom CallbackHandler

When needed, a custom callback handler that implements the `CallbackHandler` interface could also be created. The interface has only one method that has to be implemented:

```
public void handle(javax.security.auth.callback.Callback[] callbacks)
```

There are different types of `Callbacks` objects that can be used in the method above. This gives a programmer the ability to interact with a calling application, to retrieve specific authentication data such as username and password, or to display certain information, such as error and warning messages. Some of the callbacks implementation are listed below (for complete list, see WebSphere Information Center and the JAAS `javax.security.auth.callback` APIs):

- ▶ `javax.security.auth.callback.TextOutputCallback`. It is used to display information messages as well as warning and error messages.
- ▶ `javax.security.auth.callback.NameCallback`. It is used to retrieve the name information (login name).
- ▶ `javax.security.auth.callback.PasswordCallback`. It is used to retrieve the password information.

A simple example for custom callback handler is also included in the `ItsoHello` client. The code snippet for the custom callback handler is shown below:

Example 8-11 Code snippet from custom CallbackHandler `HelloCallbackHandlerImpl` class. Change the `WSGUICallbackHandlerImpl` with `HelloCallbackHandlerImpl` in the Example 8-9 on page 176 and/or Example 8-10 on page 177 if you want to use this custom CallbackHandler.

```
public void handle(Callback[] callbacks) throws IOException,
UnsupportedCallbackException {

    System.out.println("Custom CallbackHandler");
    System.out.println("Realm:" + WSLoginHelperImpl.getDefaultRealmName());

    for(int i = 0; i < callbacks.length; i++)
        if (callbacks[i] instanceof TextOutputCallback)
        {
            TextOutputCallback toc = (TextOutputCallback)callbacks[i];
            switch(toc.getMessageType())
            {
                case TextOutputCallback.TEXT_MESSAGE:
                    System.out.println("Text message: " + toc.getMessage());
                    break;
                case TextOutputCallback.ERROR_MESSAGE:
                    System.out.println("Error message: " + toc.getMessage());
                    break;
            }
        }
}
```

```

        case 0: // '\0'
            System.out.println(toc.getMessage());
            break;
        ...
        default:
            throw new IOException("Unsupported message type: " +
                toc.getMessageType());
    }
}
else if (callbacks[i] instanceof NameCallback)
{
    NameCallback nc = (NameCallback)callbacks[i];
    System.out.print(nc.getPrompt());
    System.out.flush();
    nc.setName((new BufferedReader(new
        InputStreamReader(System.in))).readLine());
}
else if (callbacks[i] instanceof PasswordCallback)
{
    PasswordCallback pc = (PasswordCallback)callbacks[i];
    System.out.print(pc.getPrompt());
    System.out.flush();
    String pwd = (new BufferedReader(new
        InputStreamReader(System.in))).readLine();
    pc.setPassword(pwd.toCharArray());
}
else if (!(callbacks[i] instanceof WSCredTokenCallbackImpl))
    throw new UnsupportedCallbackException(callbacks[i],
        "Unsupported callback");
}

```

Running the client with the above callback handler will show a challenge text based prompt like:

```

Custom CallbackHandler
Realm : <default>
Username: viking
Password: thepwd

```

8.7 Securing the connection

As explained in 8.2, “Java client authentication protocol” on page 156, the IIOP protocol is used when an application client access EJBs service using ORB objects. However, in preparation for a request to flow between these two ORB objects, client and server, a connection over TCP/IP transport layer has to be established (IIOP over TCP/IP). When a secure connection between client and

server is required, WebSphere provides option to encrypt the connection using SSL (IIOP over SSL). Securing EJBs in WebSphere is discussed in 7.5.3, “RMI/IIOP transport channel protection” on page 150. For the application client, enabling IIOP over SSL involves several configuration properties in the CORBA client configuration file (for example `sas.client.props`, see “The `sas.client.props` file” on page 160):

- ▶ All properties under the *SSL Configuration* block. Make sure the values are synchronized with the ones specified in the server side.
- ▶ Some properties under the *CSIV2 add-on authentication protocol* block:
 - `com.ibm.CSI.performTransportAssocSSLTLSRequired` is set to true. This makes sure that the client only communicate with servers that support SSL.
 - `com.ibm.CSI.performMessageIntegritySupported` and `com.ibm.CSI.performMessageConfidentialitySupported` properties are set to true. This will make sure that the client can operate with different SSL encryption levels. If needed, the *required* version of those properties can also be set to true.

8.7.1 IIOP over SSL, a thin client example

A simple thin application client, explained in “Thin application client” on page 167, will be used below to show the IIOP over TCP/IP and IIOP over SSL connections between a Java application client and an enterprise bean resource. In order to do that, you need to modify/verify both the thin application client and the WebSphere Application Server where the enterprise bean resource is installed:

1. Run the script `runThinClient.bat` provided in this book. Make sure that you get a correct result. For example, when you access the `UnsecuredClient` example (choice **a**), you should see this output:
Accessing unsecured Hello bean
Message from Hello bean: Hello you you UNAUTHENTICATED (roles: Anonymous)
2. Open WebSphere Administrative Console and verify that the **CSIV2 Inbound Transport** is set to SSL-supported as in Figure 8-6 on page 181. This means that the server will accept both SSL and non-SSL connections.

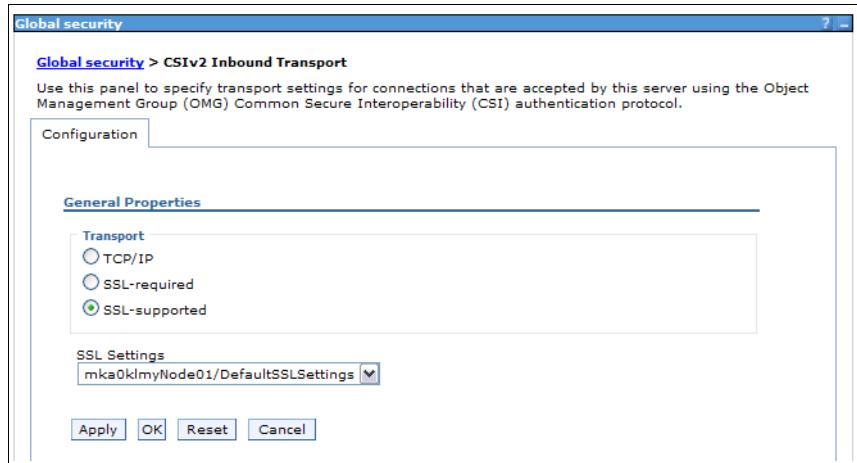


Figure 8-6 CSIV2 Inbound Transport default setup. Notice that SSL is supported but not required.

3. Modify the script runThinClient.bat change the

```
set CLIENT_TRACE=-Dcom.ibm.CORBA.CommTrace=false into
set CLIENT_TRACE=-Dcom.ibm.CORBA.CommTrace=true.
```

This will enable the tracing for the thin application client example, where the trace output can be found in file orbtrc.<timestamp>.txt.

IIOP over TCP/IP

In this example, we will show a thin client, which does not support SSL connection, connecting to an enterprise bean resource in a server which support (but not required) SSL connection.

1. Edit the CORBA configuration client file, found in

```
thinClient\properties\sas.client.props
```

(not the one on the server). Set SSL connection properties

```
com.ibm.CSI.performTransportAssocSSLTLSRequired=false and
com.ibm.CSI.performTransportAssocSSLTLSSupported=false.
```

This means that the client does not supports the SSL connection. With this setup, although the server supports SSL connection (but not required), the connection between this client and the server will be done using TCP/IP. We will verify this by examining the trace output file.
2. Run the script runThinClient.bat and choose option (a). When done, examine the trace output file orbtrc.<timestamp>.txt.

Example 8-12 Snippet of a trace output file where a client connect to a server using TCP/IP.

```
12:02:52.303 com.ibm.rmi.ras.Trace dump:80 P=968498:0=0:CT ORBRas[default]
...
Date:      November 17, 2004 12:02:52 PM EST
Thread Info: RT=0:...:WSTCPTTransportConnection[addr=9.42.171.128,port=2809,...]
...
Date:      November 17, 2004 12:02:53 PM EST
Thread Info: RT=1:...:WSTCPTTransportConnection[addr=9.42.171.128,port=9100,...]
...
Date:      November 17, 2004 12:02:55 PM EST
Thread Info: RT=1:...:WSTCPTTransportConnection[addr=9.42.171.128,port=9100,...]
...
```

3. From the trace output file above, it can be seen that the connection is kept in the TCP/IP level, as compared with the trace output shown in Example 8-13 for an SSL connection.

IIOP over SSL

In this example, we will show a thin client, which support SSL connection, connecting to an enterprise bean resource in a server which support (but not required) SSL connection.

1. Similar like above, edit the CORBA configuration client file, found in `thinClient\properties\sas.client.props`. Set SSL connection properties `com.ibm.CSI.performTransportAssocSSLTLSRequired=false` and `com.ibm.CSI.performTransportAssocSSLTLSSupported=true`. This means that the client supports the SSL connection (but not required). Since now both client and server support the SSL connection, whenever this client connects to server, the connection will be done in SSL mode.
2. Run the script `runThinClient.bat` and choose option (a). When done, examine the trace output file `orbtrc.<timestamp>.txt`.

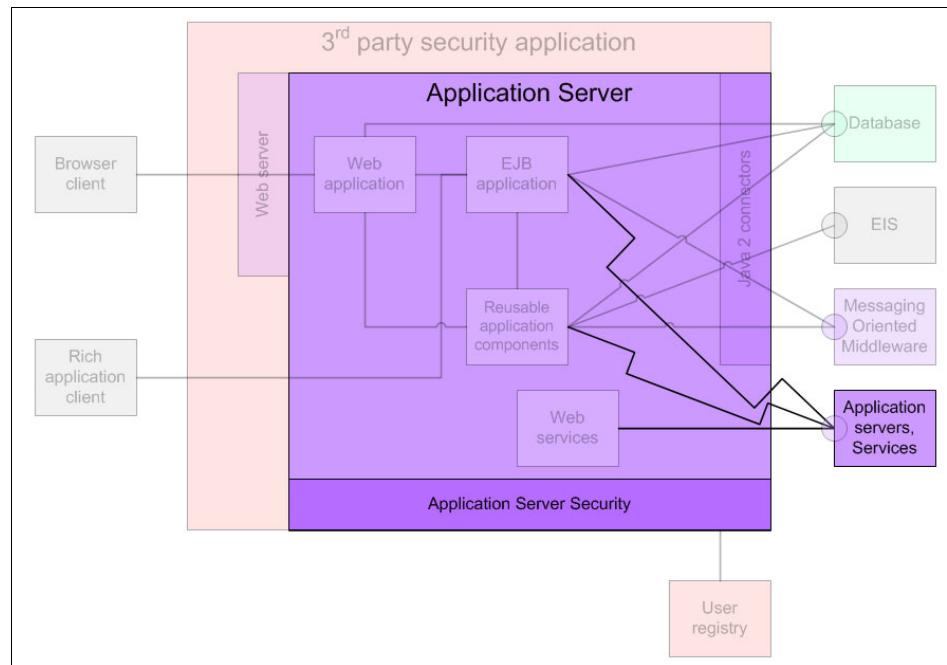
Example 8-13 Snippet of a trace output file where a client connect to a server using SSL.

```
11:28:54.323 com.ibm.rmi.ras.Trace dump:80 P=930648:0=0:CT ORBRas[default]
...
Date:      November 17, 2004 11:28:54 AM EST
Thread Info: RT=0:...:WSTCPTTransportConnection[addr=9.42.171.128,port=2809,...]
...
Date:      November 17, 2004 11:28:57 AM EST
Thread Info: RT=1:...:WSSLTransportConnection[addr=9.42.171.128,port=9100,...]
...
Date:      November 17, 2004 11:28:59 AM EST
Thread Info: RT=1:...:WSSLTransportConnection[addr=9.42.171.128,port=9100,...]
...
```

3. From the trace output file above, it can be seen that the connection is switched from TCP/IP to SSL.

Extending security beyond the application server

Security attribute propagation



9.1 Introduction

JAAS provides a standard API for defining pluggable authentication and Java 2 authorization extensions. Many LoginModules can be chained together using JAAS configuration files. User authentication is done by LoginModules and the authenticated user is represented by a Subject. A Subject may also own security-related attributes, which are referred to as credentials. Sensitive credentials that require special protection, such as private cryptographic keys, are stored within a private credential Set. Credentials intended to be shared, such as public key certificates, are stored within a public credential Set. WebSphere Application Server V5.1.1.and above uses JAAS for authentication. In WebSphere Application Server, Login modules authenticate the user, create the subject and populate it with security attributes information.

The security attribute propagation feature enables WebSphere Application Server to send security attribute information regarding the original login from one server to another server. Prior to V5.1.1, WebSphere Application Server authenticated the user and got the group information during login but passed only the identity of the user downstream. This has been significantly enhanced in version V5.1.1. and above. This enhanced feature is called security attribute propagation using which WebSphere Application Server can now pass security attribute information, including authenticated Subject contents and other custom security attributes downstream. These security attributes that can be transported to other application servers may be obtained during the **initial login** in the following ways:

1. When WebSphere Application Server does the authentication, it can query the user registry for static security attributes like users language preference or e-mail, etc. and the subject is populated with these attributes.
2. The security attributes may also be populated by using a custom login module in WebSphere Application Server. The custom login module may be used for populating the dynamic attributes like users login time, location of the login, IP address of the original user. The custom login module can insert custom security attributes in the Subject which contains the static as well as dynamic information.
3. If there is an external security server like Tivoli Access Manager involved, the security attributes may be propagated using the appropriate Trust Association Interceptor for that reverse proxy server. The enhanced TAI++ interface is able to assert a fully populated subject which can be propagated to other servers.

Important: The custom attributes or tokens in Subject are not used by WebSphere Application Server for authentication or authorization. However WebSphere Application Server will still handle propagation of these customized tokens but it does not do serialization or deserialization of the custom tokens. The Java programming language specifies the rules for how Java code can serialize and deserialize an object. The serialization and deserialization of the custom tokens should be carried out by the implementation and handled in the custom login module.

Why is Security Attribute important?

The Security attribute propagation is useful when you want to propagate the security attributes of the authenticated user specially the dynamic attributes like login time, logon location. Security attribute propagation makes the JAAS Subject based runtime more useful. When you use the Reverse Proxy Server, the originating attributes are very important, because they define the ACL of the originating caller through out the down stream system. For example, when you want to maintain the information about the originating caller identity, authenticated user strength, location and so on, you can use the security attribute propagation feature and add these attributes to the Subject that is propagated downstream.

9.2 Initial Login versus Propagation Login

Before we go into Initial Login and Propagation Login, let's define **Identity Propagation** and **Identity Assertion**.

Identity propagation refers to the low level capability of passing the users identity to another server or system. For example, let's say there are two systems A and B. A knows who the user is. The system A passes the identity of the user to system B. This is known as *Identity Propagation*. In the context of WebSphere Application Server, this means the WebSphere Application Server A does the initial authentication, authenticates the user and creates a Subject and then propagates the users identity to another WebSphere Application Server server B in its trust domain.

Identity assertion is the manner in which identity of the user or system is projected (that is, asserted) from one system to another. With respect to **Identity assertion and propagation** here are some important considerations:

1. The basis for Identity propagation and Assertion is the establishment of the trust relationship between Systems A and B. Systems can authenticate to each other by using SSL based client certificates or by using a system

- password which represents a “shared secret” shared only by systems A and B.
2. Strong network protection is a must while doing identity assertion. It is important that intruders not be able to attack the system from within the network and then take advantage of the identity assertion trust relationship. Thus, for example, if a password is used for authenticating, network protection must be in place to protect that password.
 3. When asserting identities, the identities should be the same in the registry of System A or System B. In the figure we have assumed that the registry is the same for System A and System B. If the registries are not the same, some sort of identity will have to done which complicates things more. We will not discuss the scenario when the identities are different.

When WebSphere Application Server authenticates a request, it first checks to see if the authentication should occur using Initial Login or a Propagation Login. An *initial login* is the process of WebSphere Application Server authenticating the user information. Typically the user proves his identity through a credential which may be userid and a password, or a certificate, and WebSphere Application Server then validates the user against the user registry and looks up secure attributes that represent the user access rights.

Propagation Login is the process of validating the user information, typically an Lightweight Third Party Authentication (LTPA) token, and then deserializing a set of tokens that constitute both custom objects and token objects known to the WebSphere Application Server. For example, when the user identity is propagated from WebSphere Application Server on system A to WebSphere Application Server on system B, WebSphere Application Server B does a propagation login to validate the tokens typically the LTPA token it received from the WebSphere Application Server A to ensure that its a valid LTPA token.

9.3 Token framework

WebSphere Application Server provides a token framework to enable populating the JAAS Subject with Java objects and to provide the serialization functionality for those objects. The token framework is able to identify the uniqueness of the token contained in the Authenticated Subject. This uniqueness of the token determines how the Subject gets cached and the purpose of the token. This uniqueness of the token also determines how the token gets recreated when the Subject is lost. The Token framework is very useful in propagating custom security attributes downstream. WebSphere Application Server Token framework defines four token interfaces that enable the WebSphere Application Server runtime to determine how to propagate the token. All of the token types defined by the propagation framework have similar interfaces. Basically, the token types

are *marker interfaces* (marker interface is a Java interface which does not actually define any fields and is just used to “mark” Java classes) that implement the `com.ibm.wsspi.security.token.Token` interface. This interface defines most of the methods. There are four tokens provided by the WebSphere Application Server Token framework are as follows:

1. **Authorization Token** - This token is User specific and it contains the authorization related security attributes for the authenticated Subject. It is used by WebSphere Application Server to make J2EE Authorization decisions
2. **Single Sign-On Token** - A SingleSignonToken is also a user specific token that is added to the JAAS Subject. It enables WebSphere Application Server to do Single Signon to other WebSphere Application Servers. It is added to the response as an HTTP cookie and sent to the browser and represents unique authentication. The default value of this token is the LTPA Token version 2. This LTPA Token version 2 is significantly enhanced than the previous LTPA Token versions.
3. **Propagation Token** - The propagation token is not a user specific token and therefore it is not stored in the Subject. Instead, the propagation token is stored on the thread context. The default propagation token records all user switches and host switches.
4. **Authentication Token** -The authentication token contains the identity of the user. This token is equivalent to the Lightweight Third Party Authentication (LTPA) token in previous versions. This token type is typically reserved for internal WebSphere Application Server purposes. The Authentication Token is added to the HTTP Response as a LTPAToken cookie to maintain backward compatibility with previous versions.

Table 9-1 Token framework

Token Name	Interface <code>com.ibm.wsspi.websphere. security.token.*</code>	Subject based or Thread based	Notes
Authorization Token	<code>com.ibm.wsspi.security.token. AuthorizationToken</code>	Based on authenticated Subject	Propagated downstream
Single Signon Token	<code>com.ibm.wsspi.security.token. SingleSignonToken</code>	Based on authenticated Subject	Sent to the browser as a cookie named LtpaToken2 by default. Propagated downstream
Authentication Token	<code>com.ibm.wsspi.security.token. AuthenticationToken</code>	Based on authenticated Subject	Exists for backward compatibility. Has the old LtpaToken for backward compatibility. Propagated downstream

Token Name	Interface <code>com.ibm.wsspi.websphere.security.token.*</code>	Subject based or Thread based	Notes
Propagation Token	<code>com.ibm.wsspi.security.token.PropagationToken</code>	Based on the thread and not based on Subject.	Propagated downstream

9.4 Custom Implementation of Tokens

Each of the WebSphere Application Server tokens discussed above can be customized by implementing the appropriate Interface. This customization may be done in two ways:

- ▶ You can add custom attributes to the default token.
- ▶ You can create your own implementation of the token by extending the specific Token Interface.

First you must carefully consider your need to implement a custom token. In most cases you can add custom attributes to the default token and be able to retrieve them in your application code. You should carefully consider writing your own implementation if you want to accomplish one of the following tasks.

1. Isolate your attributes within your own implementation.
2. Serialize the information using custom serialization; that is, your java code should be able to serialize and deserialize the token. If you are using the WebSphere Application Server provided default token, WebSphere Application Server takes care of this for you. Make custom decisions based on the information in the customized token at the appropriate time.
3. You may need to use custom encryption and decryption for tokens.

Adding custom attributes to the default token is usually sufficient for propagating the user or non-user specific attributes. Writing custom implementations is usually for Service Providers to enable them to provide custom services.

Steps for writing custom implementations of Tokens

However if you are modifying the default implementation of the tokens, you will need to go through the following steps:

1. If you plan to implement more than one token type, consider creating an abstract class that implements the `com.ibm.wsspi.security.token.Token` interface. All of your token implementations might extend the abstract class and then most of the work is completed. However if there are considerable

differences between how you handle the various token implementations, you can implement the interface directly.

2. If you need to implement a custom token interface, you should ensure that the methods required by the specific token that you are trying to implement are implemented. When the custom token object is added to the Subject, it does affect the cache lookup of the Subject if you return something in the `getUniqueID()` method. Therefore when you are implementing a custom Token, you must ensure that the `getUniqueID` method returns a unique token.
3. After you implement the specific token interface, you can place your compiled code in the `<WebSphere_root>/classes` directory. Alternatively, you can place the class in any private directory. If you place it in any private directory, you need to add the JAR file or the directory that contains your code into the `server.policy` file so that it has the necessary permissions that are needed by the server code.
4. Write a JAAS login module so that the customized tokens are added and received and processed properly during WebSphere Application Server logins. You may make your specific Token read-only in the commit phase of the login module. If you make the token read-only, additional attributes cannot be added to the token within your applications.
5. You will need to add the JAAS login module to the specific application and system login configurations. You can also add the implementation from an application. However, in order to deserialize the information, you will still need to plug in a custom login module, so that when the token is propagated the WebSphere Application Server logins receive the serialized version of the custom token.
6. In most cases while implementing a custom JAAS Login module, you will add your custom login module, after the `com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule` for receiving serialized versions of your custom token. The `com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule` is contained in the following JAAS aliases: **WEB_INBOUND, RMI_INBOUND, DEFAULT**.
7. Also, you can add this login module to any of the application logins where you might want to generate your custom Token. For details on writing JAAS Login module, refer to 4.2.2, “Login module” on page 51.

9.4.1 Authorization token

The authorization token contains most of the user's information. It contains the authorization-related security attributes that are propagated through the Subject. The default authorization token is used by the WebSphere Application Server authorization engine to make Java2EE authorization decisions.

Default Authorization Token

You can use the default Authorization Token when you want to add security attributes that will get propagated downstream. These security attributes must be specific to the user associated with the authenticated Subject. If they are not specific to the user, then you should consider adding them in the propagation token which we will discuss later.

Add custom attributes to the default Authorization Token

To add attributes into the default AuthorizationToken, you must use a custom JAAS login module. This custom login module needs to be configured in the WEB_INBOUND JAAS login module configuration (system login).

WEB_INBOUND Jaas Alias has 2 login modules defined:

- ▶ com.ibm.ws.security.server.lm.ItpaLoginModule
- ▶ com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule

The two configurations are shown below:

Global security > System login configuration > WEB_INBOUND > JAAS Login Modules

Each entry in the login configuration must contain at least one login module. However, you can define more than one login module for a login configuration. If you define more than one login module for a login configuration, they are processed in the order that they are defined.

Select	Module class name	Authentication strategy	Module order
<input type="checkbox"/>	com.ibm.ws.security.server.lm.ItpaLoginModule	REQUIRED	1
<input type="checkbox"/>	com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule	REQUIRED	2

Total 2

Figure 9-1 JAAS login configurations

You can insert the custom login module after the `com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule` using a higher module order number in the definition.

First login occurs in ItpaLoginModule and after that a default AuthorizationToken is created in the wsMapDefaultInboundLoginModule. As a third step, your custom login module can add custom attributes to the Authorization token.

Custom Authorization Token

When should you implement a Custom Authorization Token?

The default AuthorizationToken is sufficient for propagating attributes that are user-specific. However you can write a custom Authorization Token implementation if you want to accomplish one of the following:

- ▶ Isolate your attributes within your own implementation.
- ▶ Serialize the information using custom serialization. You must deserialize the bytes at the target and add that information back on the thread. This task also might include encryption and decryption.
- ▶ Affect the overall uniqueness of the Subject using the getUniqueId() method.
- ▶ If you want to make custom authorization decisions using the information in the token at the appropriate time.

Steps for Custom Authorization Token Implementation

To implement a custom authorization token, you must complete the following steps:

1. The sample code implements the *com.ibm.wsspi.security.token.AuthorizationToken* interface directly. You can extend from an abstract class if you have other custom token implementations.
2. Write a custom JAAS login module that will add and receive the custom AuthorizationToken during WebSphere Application Server login.
3. Add the custom login module to the application and system login configurations that already contain the *com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule* for receiving serialized versions of your custom authorization token. You will need to add your custom login module after *com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule* since the custom AuthorizationToken implementation relies on the information added by *wsMapDefaultInboundLoginModule*.

9.4.2 Single Sign-On (SSO) token

This default Single Sign-On token is used by the WebSphere Application Server runtime code only. It is added to the authenticated Subject and also added to the HTTP response as an HTTP Cookie.

Default Single Sign-On Token

WebSphere Application Server defines a default *SingleSignonToken* with the name as *LtpaToken* and the version 2. The name and version together form the cookie name and therefore the cookie name added is *LtpaToken2*. There are size limitations for this token when it is added as an HTTP cookie and hence be careful about adding extra attributes to this token.

It is also recommended that any time you use cookies, use the Secure Sockets Layer (SSL) protocol to protect the request. Using an SSO token, Web users can authenticate once when accessing Web applications across multiple WebSphere Application Server.

Custom Single Sign-On Token

You may implement your own custom SSO token which will get added as an HTTP Response as an HTTP Cookie. Consider writing your own implementation of the Single Sign-On token if you want to accomplish one of the following:

- ▶ Separate your attributes within your custom implementation.
- ▶ Use custom serialization, or custom encryption/decryption.
- ▶ Check the uniqueness of the subject using the `getUniqueId()` method.

Keep in mind the following guidelines while implementing your custom Single Sign-On Token:

- ▶ HTTP cookies have a size limitation so do not add too much data to this token.
- ▶ This cookie is not used and nor is handled by the WebSphere Application Server runtime.

The steps below explain the process of developing a custom Single Sign-On Token.

1. The first step is to write your custom token properly. The sample code contains an implementation of
com.ibm.wsspi.security.token.SingleSignonToken interface directly
2. Add the class to <WebSphere_root>/classes, then add that directory or the JAR file to the server.policy file so that WebSphere Application Server can load your classes.
3. Write the JAAS login module that will create and add your tokens properly during WebSphere Application Server logins.
4. Add your JAAS login module to WebSphere Application Server system login configurations that contain the
com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule for receiving serialized versions of your custom propagation token.

9.4.3 Propagation token

The Propagation Token is not user specific and thus not part of the Subject. A default PropagationToken is stored on the thread of execution for applications. WebSphere Application Server propagates this PropagationToken downstream and the token stays on the thread context. When a request is sent outbound to

another server, the propagation token on that thread is sent with the request and the token is executed by the target server.

Default Propagation Token

The default propagation token does the following:

1. It monitors and logs all user switches and host switches. The token data should be available from within the container of any resource where the PropagationToken lands. Remember that you must enable the propagation feature at each server where a request is sent in order for propagation to work.
2. There is a WSSecurityHelper class that has application programming interfaces (APIs) for accessing the PropagationToken attributes and for adding custom attributes to the propagation token in your application code.
3. After you add attributes to the PropagationToken, you cannot change these attributes. This enables the WebSphere Application Server security runtime to add auditable information and have that information remain there for the life of the invocation. Any time that you add an attribute to a specific key, an ArrayList is stored to hold that attribute. The order of the attributes added is preserved. The first element in the String Array returned is the first attribute added for that specific key.
4. In the default PropagationToken, any data changes to the token is recorded using a change flag. These changes are tracked to enable WebSphere Application Server to know when to re-send the authentication information downstream so that the downstream server has those changes. A Common Secure Interoperability version 2 (CSIV2) session is maintained between servers for an authenticated client. Whenever the PropagationToken changes, a new CSIV2 session is generated and a new authentication occurs. Therefore if there are frequent changes to the PropagationToken during a method, it will cause frequent downstream calls which may impact performance.
5. Whenever the PropagationToken is propagated either horizontally or downstream, the name of the receiving application server is logged into the PropagationToken. The format for each server in the list is "Cell:Node:Server", which provides you access to the cell name, node name, and server name of each application server that receives the invocation.
6. You can also get the caller list from the PropagationToken. Anytime a authenticated Subject is generated, it is logged in the token. Basically, whenever an authenticated user is set on the thread, the user is logged in the default PropagationToken. At times, the same user might be logged in multiple times if the RunAs user is different from the caller.

Adding custom attributes to the default Propagation token

You can add custom attributes to the default PropagationToken for application usage. This token is transported along with the request to downstream servers, so that the attributes are available in your downstream EJBs or in your application servers when they are needed.

There are some considerations when you use the default PropagationToken to add attributes, as follows:

- ▶ When you add information to the PropagationToken, it affects CSIV2 session caching. Add information sparingly between remote requests.
- ▶ After you add information with a specific key, the information cannot be removed.
- ▶ You can add as many values to a specific key as your need. However, all of the values will be returned as a string array in the order they were added. So you have to keep track of values added and their sequence.
- ▶ The PropagationToken is available only on servers where security attribute propagation is enabled and Global Security is enabled.
- ▶ An application cannot use keys that begin with either com.ibm.websphere.security or com.ibm.wsspi.security. These prefixes are reserved.

Implementing a Custom Propagation Token

The default PropagationToken is typically sufficient for propagating attributes that are not user-specific. Consider writing your own implementation if you want to:

- ▶ Isolate your attributes within your own implementation.
- ▶ Use custom serialization.
- ▶ Use custom encryption and decryption for your tokens.

Follow the steps below to implement a custom Propagation token.

1. Code your implementation of the PropagationToken Interface. The sample code implements com.ibm.wsspi.security.token.PropagationToken. You can download the sample from the additional materials.
2. Add the class to <WebSphere_root>/classes, then add the JAR file to the server.policy file so that WebSphere Application Server can load your classes.
3. Write the JAAS Login module that will create and add your tokens properly during WebSphere Application Server logins.
4. Add your JAAS login module to WebSphere Application Server system login configurations that contain the

com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule for receiving serialized versions of your custom propagation token. The *com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule* is used in the following JAAS login module configurations: WEB_INBOUND, RMI_INBOUND, DEFAULT. You can also add this login module to any of the application logins where you might want to generate your custom PropagationToken and store it on the thread context during the login.

9.4.4 Authentication token

As the name indicates the Authentication token contains the authentication information of the user. The Authentication Token serves the same function as the old Lightweight Third Party Authentication (LTPA) token in previous versions of WebSphere Application Server (prior to V5.1.1). The default authentication token is reserved for WebSphere Application Server runtime and is authentication mechanism specific. The Single Sign-On token is the new token format and the authentication token just serves the purpose of backward compatibility. Any modifications to this token by custom code can potentially cause interoperability problems.

9.4.5 Changing the Token Factory associated with the default token

When WebSphere Application Server generates the default Tokens, it utilizes an appropriate TokenFactory class for creating the default tokens. This token factory class is specified using a custom property in the WebSphere Application Server. To view the token class that is used by default for the tokens, follow the steps below.

1. Launch the WebSphere Application Server Administrative Console and login.
2. Select **Security** → **Global Security**.
3. Under Additional properties select **Custom properties** and you will see a list of token factories.

You can plug in your own custom TokenFactory class implementation. You need to locate the specific Token factory you want to modify and associate your custom Token Factory implementation class with the TokenFactory property value. Also you need to verify that your implementation classes are available for the WebSphere Application Server classloader.

If you need to perform your own signing and encryption of the default token, you must implement the following classes:

- ▶ com.ibm.wsspi.security.ltpa.Token
- ▶ com.ibm.wsspi.security.ltpa.TokenFactory

You can use the Lightweight Third Party Authentication (LTPA) keys or you can use your own keys for instantiating and validating your token implementation. If you use your own keys, they must be the same everywhere in order to validate the tokens that are generated using those keys.

- ▶ **Authorization Token, Propagation Token** - For Both these tokens, the default TokenFactory used is called *com.ibm.ws.security.ltpa.AuthzPropTokenFactory*. This token factory encodes the data, but does not encrypt the data in the AuthorizationToken. This is so because the AuthorizationToken is transmitted over Common Secure Interoperability version 2 (CSIV2) using Secure Sockets Layer (SSL) and therefore there is no need to encrypt the token. If you need additional security for the AuthorizationToken, you can associate a different TokenFactory implementation with this property to get encryption.

For example, if you associate *com.ibm.ws.security.ltpa.LTPAToken2Factory* with this property, the token uses the AES encryption. However, there may be a performance impact with the encryption.

- ▶ **Single Sign-On Token** - Single Sign-On token by default uses *com.ibm.ws.security.ltpa.LTPAToken2Factory* class which creates the token LtpaToken2. This TokenFactory uses the AES/CBC/PKCS5 Padding cipher for encoding.

Note: If you change this TokenFactory, you lose the interoperability with any servers running a version of WebSphere Application Server prior to V5.1.1 that use the default TokenFactory. Only servers running WebSphere Application Server V5.1.1 or later with propagation enabled are aware of the LtpaToken2 cookie.

- ▶ **Authentication Token** - The default TokenFactory for Authentication Token is called *com.ibm.ws.security.ltpa.LTPATokenFactory*. The LTPATokenFactory uses the DESede/ECB/PKCS5Padding cipher. This token factory creates an interoperable Lightweight Third Party Authentication (LTPA) token.

Note: If you modify this TokenFactory, you lose the interoperability with any servers running a version of WebSphere Application Server prior to version 5.1.1 and any other servers that do not support the new TokenFactory implementation.

If you associate *com.ibm.ws.security.ltpa.LTPAToken2Factory* with the *com.ibm.wsspi.security.token.authenticationTokenFactory* property, the token is AES encrypted. However, you need to weigh the performance against your security needs.

9.5 Horizontal Propagation

In horizontal propagation, the Subject containing the security attributes are propagated amongst the front-end WebSphere Application Servers. The default Single Sign-On token is LTPAToken version 2. You can create your own custom token and add that to the Subject in a custom Login module. The token contains the following information:

- ▶ The users unique ID.
- ▶ Timestamp.
- ▶ The key to lookup the serialized security attributes.
- ▶ The originating servers' JMX Administration endpoint which tells the receiving server how to communicate with it.

During the WebSphere Application Server Initial Login process, the Single Sign-On token is added to the Subject and the token is added to the HTTP response as a cookie. This Login process can also be customized to add custom information to the Single Sign-On token or to the Subject by using JAAS Login Modules. If you have horizontal propagation enabled, it enables the front end receiving servers to retrieve the Subject information and extract the security attributes information from the Subject. In this case, Initial Login occurs at the originating server and propagation login occurs at the receiving servers.

Horizontal Propagation using Dynacache

When WebSphere Application Servers are configured in a cluster and in the same Distributed Replication Service (DRS) replication domain, the application server propagates the serialized information to all of the servers within the same domain. Let's take a look at what happens during Horizontal propagation using Dynacache.

1. In the figure below server1 and server 2 are members of the same DRS replication domain. Application1 is deployed on server1 and server2. Let's say the user is logged in on server1. During the Initial Login process on server1, a fully populated JAAS Subject containing the tokens is created and placed in Dynacache. The Single Sign-On token is created and placed on the HTTP response as a cookie.
2. Dynacache is replicated in the DRS Replication Domain.
3. An HTTP request from application1 on server1 makes another call to application1 on server2. The original login attributes are found on server2 without additional remote requests. This is so because the Single Sign-On token is passed to server2 via cookie.
4. WebSphere Application Server security searches for authentication information, using the Single Sign-On token as the key. It first searches in the

local security cache for the Subject. Since this login was done on server1, the subject was instantiated on server1. Hence the local security cache on server2 does not have the instantiated subject. Then WebSphere Application Server security searches in Dynacache for tokens. Since server2 is in the same DRS Replication Domain, the tokens are found in the Dynacache.

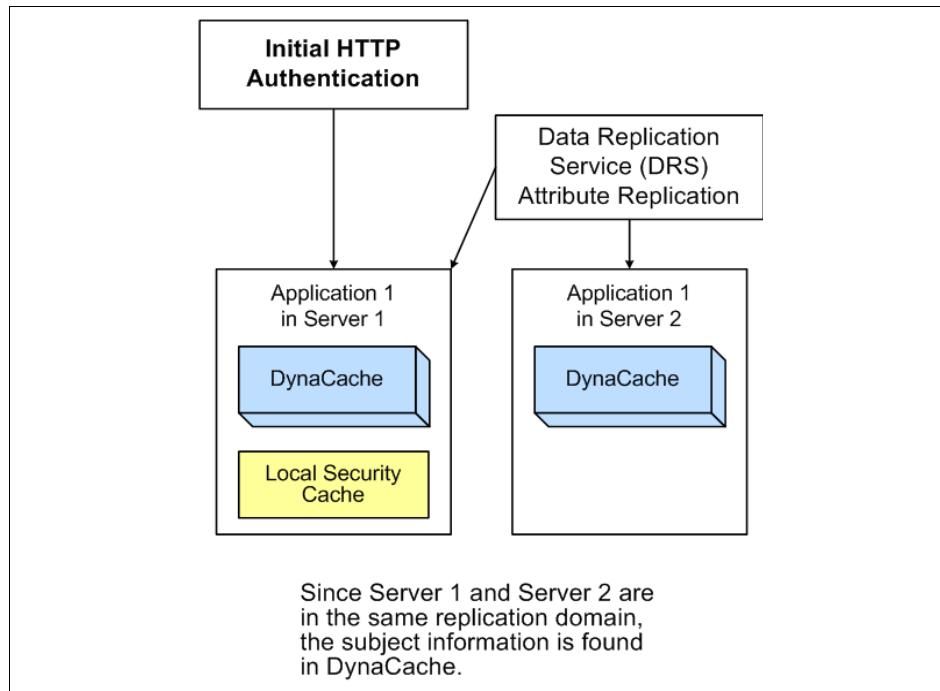


Figure 9-2 Horizontal propagation sample

The Lifetime of Dynocache entry is the same as the LtpaToken lifetime (120 minutes by default).

Horizontal propagation using JMX

Horizontal propagation can be also accomplished by using the JMX infrastructure.

In the figure below, server1 and server2 are configured in the same Data Replication Service Domain. Server 3 and server4 are configured in a separate Data Replication Service Domain.

1. The request originates from application1 on server1 (or server2). During the Initial Login, a fully populated Subject is created and put in DynaCache which gets replicated by DRS to all the servers within the DRS Domain.

2. The request is redirected to application2 on either server3 or server4. Server3 gets the Single Sign-On token from server1. It uses the Single Sign-On token as a key, and checks the DynaCache for the serialized information. The serialized information is not found in the DynaCache because the server3 and server4 are not configured in the same DRS replication domain. As a result, a secure remote Java Management Extensions (JMX) request is sent back to the originating server (server1), that hosts application1 to obtain the Subject information. Server1 sends the serialized information to server3.
3. Server3 is able to deserialize the Subject and decrypt the Tokens to get the security attribute information. Note this results in a propagation login by server 3.

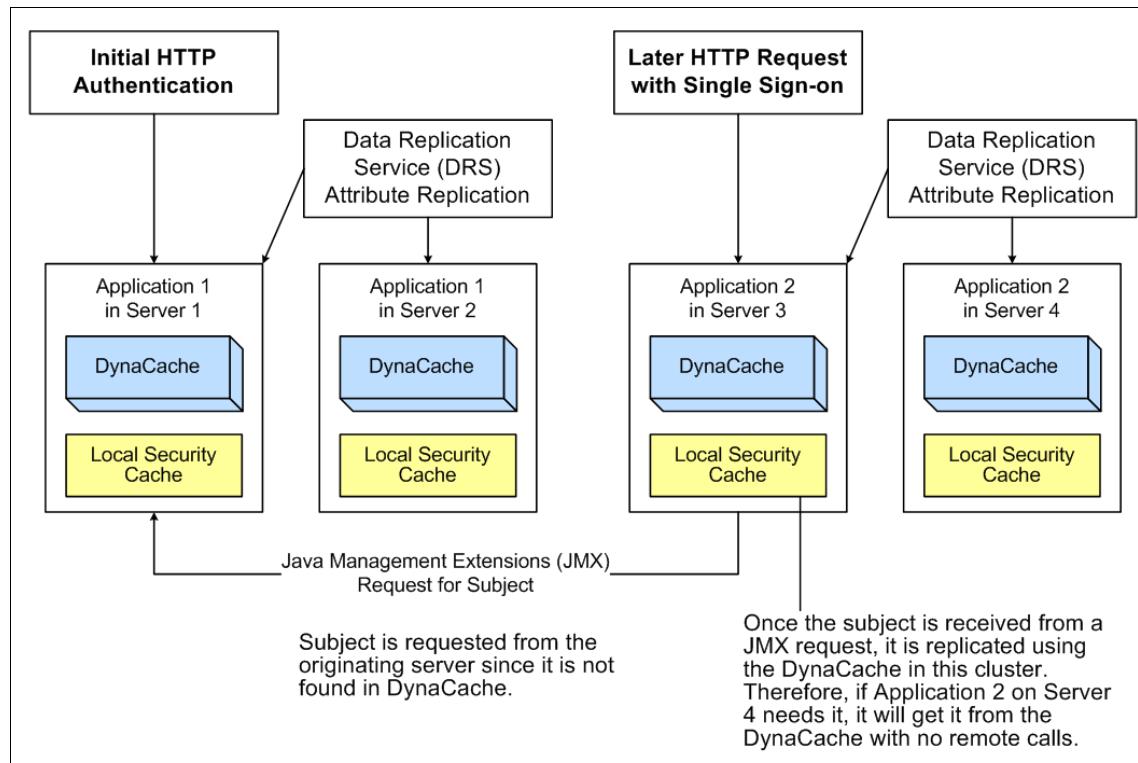


Figure 9-3 Horizontal Propagation using JMX

By using a single JMX remote call back to the originating server, the following benefits are realized.

- ▶ You get the login information from the original server.
- ▶ You do not need to perform any User registry calls because the application server can regenerate the subject from the serialized information.

- Once the server3 gets the serialized information, it regenerates the Subject and also puts that in the DynaCache for subsequent horizontal propagation using dynacache for its DRS Replication Domain.
- If the JMX Call fails for some reason, WebSphere Application Server will fall back to an Initial Login. In this scenario, the Login modules will be called and the Subject will be recreated.

Note: For the remote JMX Administration call across the cell to succeed, the 2 servers must share common security infrastructure, registries, SSL Keys, etc. Also the cell's security server ID has admin access to remote cell.

Earlier versions of WebSphere Application Server supported Single Sign-On from server1 to server2 or server3 using the LTPA token. This is still supported using the Single Sign-On token in WebSphere Application Server V6. This means that if you do not enable Web propagation, the Single Sign-On will still work using the Single Sign-On token which gets sent as a cookie to the browser and the WebSphere Application Server servers. The information contained in the Single Sign-On token enables the WebSphere Application Server servers to perform Single Sign-On. By enabling horizontal propagation, you can pass the complete Subject to other front end WebSphere Application Servers.

There are some performance implications of enabling horizontal propagation. Enabling Web inbound propagation will eliminate some user registry calls. However, the deserialization and the decryption of tokens are processing intensive tasks and may impact performance. It is recommended that you run performance tests in your environment with typical number of users with the propagation enabled and with propagation disabled to determine the implications.

9.6 Downstream propagation

There are two authentication protocols supported by IBM. Secure Association Service (SAS) is the authentication protocol used by all previous releases of the WebSphere product. SAS is deprecated and it is maintained for backwards compatibility. The Object Management Group (OMG) has defined a new authentication protocol, called Common Secure Interoperability Version 2 (CSIV2), so that vendors can interoperate securely. CSIV2 is implemented in WebSphere Application Server with more features than SAS and is considered the strategic protocol.

Downstream propagation uses Remote Method Invocation (RMI) over the Internet Inter-ORB Protocol (RMI over IIOP) to access enterprise beans running on a back-end, that is, a downstream server. The security attributes are passed

to the enterprise beans running on the downstream server by using the CSIV2 protocol that is established between the WebSphere Application Servers. Basically downstream propagation enables a downstream server to accept the client identity established on an upstream server, without having to reauthenticate.

There are two types of downstream propagation using RMI:

- ▶ **RMI_INBOUND**

When you enable security attribute propagation for RMI_INBOUND, then this indicates that the server can receive propagated security attributes from other servers in the same realm over CSIV2 protocol.

- ▶ **RMI_OUTBOUND**

When you enable security attribute propagation for RMI_OUTBOUND, this indicates that the server can send (propagate) security attributes from itself to other server in the same realm over CSIV2 protocol. Let's take a scenario described in figure where server1 makes an RMI call to server5. The following occurs:

- a. Subject contents and the PropagationToken contents are serialized at server1.
- b. Server1 makes an RMI call to server5.
- c. Serialized content sent over CSIV2 protocol to the target server (server5) that has RMI_INBOUND propagation enabled.
- d. If the receiving server does not support security attribute tokens, WebSphere Application Server sends the Lightweight Third Party Authentication (LTPA) token only.

Downstream propagation scenario

Server1 and Server2 are in the same DRS Replication Domain. Server5 is a downstream server to which an RMI call is made from server1.

1. User authenticates to server1. Subject is created during the login process at the front-end server (server1) in this case, either by a propagation login or a user registry login.
2. Server1 and server5 have downstream propagation enabled.
3. The user makes an RMI request to an EJB running on application3 on server5.
4. WebSphere Application Server propagates the tokens from Subject including custom tokens from server1 to the downstream WebSphere Application Server, server5. Thus, the security information is available for server5 for enterprise bean invocations.

5. If tokens are available, a propagation login is performed otherwise an initial login is performed. The subject is generated at the downstream server (server5) and is added to the CSIV2 session.

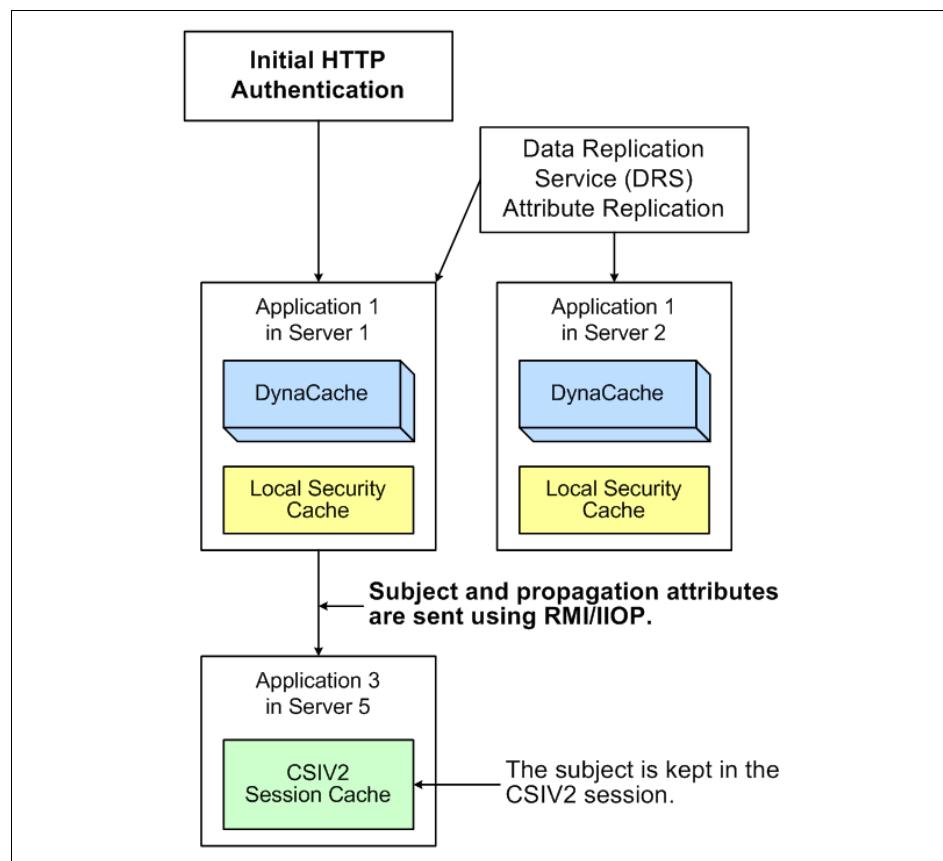


Figure 9-4 Downstream propagation scenario

The downstream server trusts the upstream server, both Remote Method Invocation (RMI) outbound and inbound propagation have to be enabled.

9.7 Enabling security attribute propagation

Common Security Interoperability Version 2 (CSIV2) defines the Security Attribute Service that enables interoperable authentication, delegation, and privileges. You can selectively enable parts of security attribute propagation depending on your server configuration and requirements. For example, you can choose to enable Horizontal Propagation among front-end WebSphere

Application Servers using DynaCache or JMX. You can also choose to enable downstream propagation. Typically, both types are enabled for any given cell.

9.7.1 Security attribute propagation for Horizontal Propagation

Complete the following steps to configure WebSphere Application Server for Horizontal Propagation.

1. Launch the Administrative Console and login.
2. Select **Security** → **Global Security**. Select **Authentication Mechanisms** then **LTPA**. Under additional Properties, select **Single Signon (SSO)**.
3. (Optional) Earlier versions of WebSphere Application Server, prior to V5.1.1, did not support security attribute propagation. It used an LTPA token for Single Sign-On purposes. If you need to interoperate with such servers, select the **Interoperability Mode** option. WebSphere Application Servers that do not support security attribute propagation receive the Lightweight Third Party Authentication (LTPA) token and the propagation token, but ignore the security attribute information that it does not understand.
4. Check the option for Web inbound security attribute propagation. This option enables horizontal propagation.

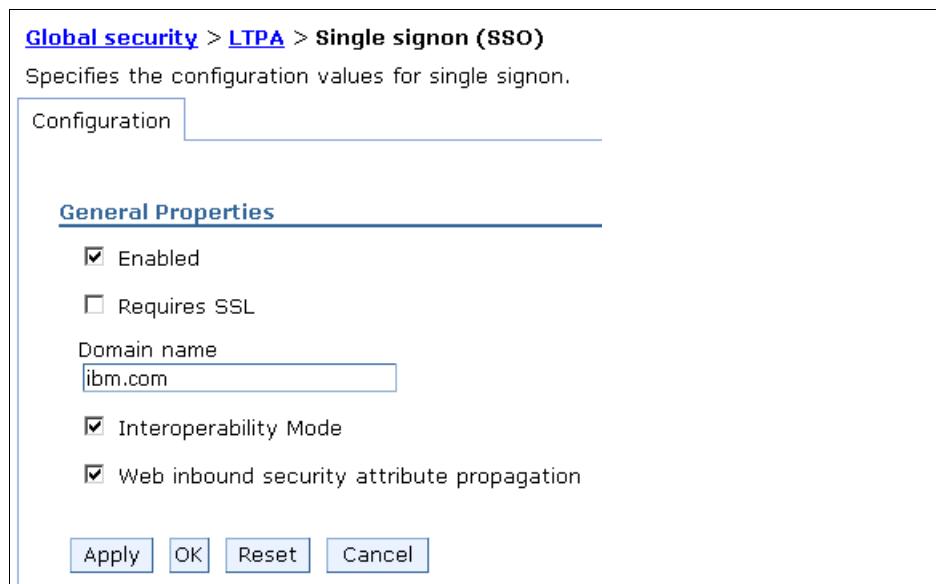


Figure 9-5 Horizontal propagation

With the Web Inbound security Attribute propagation enabled, the security attributes of the originating server where the initial login occurred, will get

propagated to the receiving server. These security attributes include any custom attributes or token that are set in the custom Login modules in the Login server.

9.7.2 Enabling Downstream Propagation

For Downstream Propagation, CSLV2 inbound and CSLV2 outbound need to be configured.

1. Select **Security** → **Global security**.
2. Under Authentication, select **Authentication protocol**. You will find the list of Authentication protocols supported by WebSphere Application Server.
3. Click **CSIV2 inbound authentication**. The Login configuration field specifies *RMI_INBOUND* as the system login configuration used for inbound requests, this cannot be changed. However, you can chain custom login modules to the Login configuration.
On this panel, ensure that **Security Attribute Propagation** is checked. Click **Apply**.
4. Select **Security** → **Global Security**. Under Authentication protocols, select **Authentication Protocol** → **CSIV2 Outbound authentication**. Note that the Login configuration says *RMI_OUTBOUND*. You cannot change the Login Configuration but you can add additional custom login modules to the configuration.
5. Ensure that **Security Attribute Propagation** is checked in this panel whenever outbound security attribute propagation is selected.

Important: WebSphere Application Server propagates only the objects within the Subject that it can serialize. For the custom objects within the Subject, you will have to take care of serialization for it to be propagated properly.

6. Click **Apply**.
7. Save the configuration for WebSphere.

Optional: Select the **Custom Outbound Mapping** option if you deselect the Security Attribute Propagation option and you want to use the *RMI_OUTBOUND* login configuration. If the Custom Outbound Mapping option nor the Security Attribute Propagation option is selected, WebSphere Application Server does not call the *RMI_OUTBOUND* login configuration. If you need to plug in a credential mapping login module, you must select the Custom Outbound Mapping option.

9.8 Advantages of security attribute propagation

The propagation of security attributes in WebSphere Application has significant benefits. It eliminates the need to perform registry lookups at each hop along an invocation.

In your environment, you might use a Web proxy server (for example: WebSEAL) to perform user authentication and gather group information and other security attributes. In previous releases, WebSphere Application Server could only use the identity of the user and disregarded all the other security attributes. In the current release of the application server, information that is obtained from the Web proxy server can be used by WebSphere Application Server and propagated downstream to other server resources without additional calls to the user registry.

Another significant benefit of the security attribute propagation is that the user switches that occur because of J2EE Run-As configurations do not cause the application server to lose the original caller information. This information is stored in the propagation token that is located on the running thread.

This also enables third-party providers to plug in custom tokens which can then be propagated via custom login modules. The token interface contains a `getBytes()` method that enables the token implementation to define custom serialization, encryption methods or both.

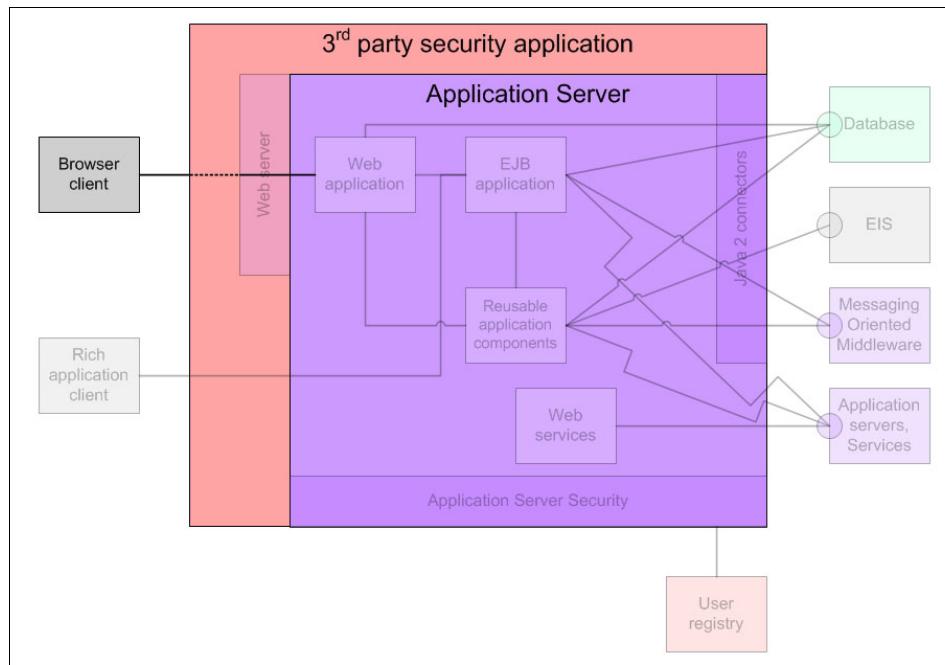
Security attribute propagation provides the ability to have a unique ID for each token type. This unique ID is used to formulate a more unique subject identifier than just the user name in cases where dynamic attributes might change the context of a user login. The token interface has a `getUniquelId()` method that is used for returning a unique string for caching purposes.

For example, you might need to propagate the time of the day when the user logs into the system. This time of the day can be generated during the login using either a Web proxy server or by configuring a custom login module in the `WEB_INBOUND` login configuration. This information can then be added to the subject prior to serialization. Other attributes might be added to the subject and use a unique ID. All of the unique IDs must be considered for the uniqueness of the entire Subject. WebSphere Application Server has the ability to specify what is unique about the information in the Subject, which might affect how the user accesses the subject later.



Chapter 10

Securing a WebSphere application using Tivoli Access Manager



10.1 Tivoli Access Manager introduction

IBM Tivoli Access Manager for e-business (Access Manager) is a policy-based access control solution for e-business and enterprise applications. Access Manager is a collected suite of security management services with a variety of distributed blades and plug-ins for the infrastructure components of e-business applications.

10.1.1 Benefits

Tivoli Access Manager allows you to control access across your entire e-business infrastructure, without multiple and possibly conflicting security policies, for any enterprise with multiple Web-based applications.

There is a business-wide change in focus from implementing application-specific security in order to prevent inappropriate users from accessing resources, towards attempting to develop a common and consistent security policy and base its implementation on common reusable security services and infrastructure.

This is about controlling network identity, correctly identifying a user once via authentication and passing that identity together with credentials through to the other components of the e-business infrastructure, applications included. Then the permissions for that identity can be tested locally and access can be given, depending on the security policy for those resources via authorization.

The externalized security provided by Access Manager includes strategies to include legacy applications in Single Sign-On solutions through integration with pre-existing user registries and authorization databases.

If, regardless of which application a user accesses within an enterprise, they always log on with the same ID and password (although there may be a requirement for stronger authentication or re-authentication, perhaps token or certificate-based around particularly sensitive information or high value transactions), then that consistent user experience appears, from the user's viewpoint at least, as Single Sign-On. Attempting to ensure users have only a single identity within your network increases the likelihood of leveraging existing infrastructure to actually provide it.

The central definition and management/administration of security policies provides a number of benefits.

- ▶ Reduced security risk through ensured consistency from a services-based security architecture.

- ▶ Lower administration costs due to centralized administration of a reduced number of security systems. This also allows for the “de-skilling” of support staff since the security policies are based on a single application suite rather than, as in many current examples, the multiple and different operating systems of chained infrastructure platforms.
- ▶ Faster development and deployment with a common services-based architecture.
- ▶ Reduced application development and maintenance costs from increased efficiency and productivity by saving on isolated system and/or application specific security development efforts
- ▶ For those industries where legislative compliance impacts security, for example privacy requirements, centralized architecture provides a more responsive environment as well as a single point to apply policy.

All of these benefits contribute to enabling an enterprise to be faster to market with new applications and features.

The down side of having a single security solution based on a single technology or product is that any product-specific security exploitation results in enterprise-wide vulnerability. It does, however, let you concentrate your defenses rather than be forced to dissipate your efforts across multiple platforms and products.

10.1.2 Access Manager Secure Domain

The Access Manager Secure Domain provides a secure computing environment in which Access Manager enforces security policies for authentication, authorization, and access control. Ignoring performance, redundancy and availability considerations which must be addressed in production systems, the essential components can be seen in Figure 10-1 on page 214.

IBM Tivoli Access Manager V5.1 requires a user registry and can be configured to use different products, including Microsoft Active Directory and iPlanet, but the product itself ships with IBM Tivoli Directory Server V5.2, underpinned by the IBM DB2 Universal Database.

The Access Manager Policy Server maintains the master authorization policy database which contains the security policy information for all resources and all credentials information of all participants within the secure domain, both users and servers. A secure domain contains physical resources requiring protection. These resources include programs, files and directories. A virtual representation of these resources, protected by attaching ACL and POP entries, is stored by the Policy Server.

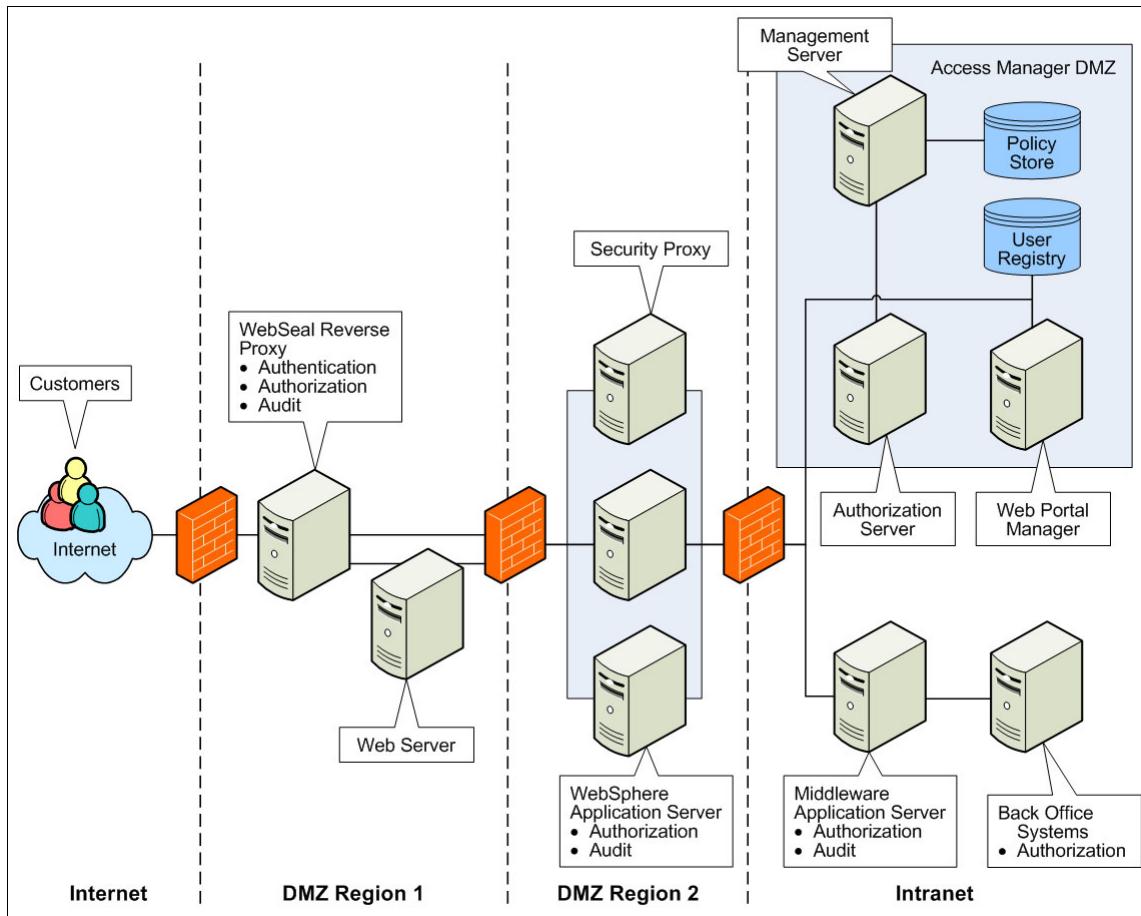


Figure 10-1 Typical three-tier infrastructure supporting e-business applications

The Policy Server replicates this database to all the local authorization servers, including WebSEAL, throughout the domain, publishing updates as required. The Policy Server also maintains location information about the other Access Manager and non-Access Manager servers operating in the secure domain. There can be only one Policy Server active within a domain.

Access Manager provides C and Java authentication and authorization APIs which can be used programmatically within other applications and clients. Client calls for authorization decisions, through the Access Manager runtime service, which must be on every server participating in the secure domain, are always

referred to an Authorization Server. Programmatically made calls can be local or remote; they will be passed to an Authorization Server. When running local node API, the application communicates to the security server (Access Manager), no authorization server is required.

Authorization servers are the decision-making servers that determines a client's ability to access a protected resource based on the security policy. Each server has a local replica of the policy database. There must be at least one within a Secure Domain.

Web Portal Manager, a WebSphere-hosted application is provided to enter and modify the contents of the policy store and the user registry. There is also a command line utility, **pdadmin**, which extends the commands available to include the creation and registration of authentication blades such as WebSEAL which will be described a little later.

Access Manager can be configured to integrate with many of the WebSphere branded products and ships with explicit plug-ins for the following products:

- ▶ WebSphere Application Server
- ▶ WebSphere Edge Server
- ▶ Web Server Plug-in supporting IBM HTTP Server (IHS)

For details of the supported operating systems for every component please consult the Tivoli Information Center at:

http://publib.boulder.ibm.com/infocenter/tiv2help/index.jsp?toc=/com.ibm.itame.doc_5.1/toc.xml

Table 10-1 shows the components that were installed for the sample configurations in the book. The components were installed in a mixture of AIX, Linux and Windows servers. For installation instructions, see the original product documentation that comes with the package or read the documentation at the InfoCenter.

Table 10-1 IBM Tivoli Access Manager V5.1 components used

Server	Required Component
Tivoli Directory Server V5.2	Directory Server
	Directory Client
	DB2 Universal Database Edition
	Global Security Toolkit

Server	Required Component
Tivoli Access Manager Policy Server V5.1	Access Manager Runtime
	Access Manager policy Server
	Global Security Toolkit
	Tivoli Directory Client
Tivoli Access Manager Authorization Server V5.1	Access Manager Authorization Server
	Access Manager Runtime
	Global Security Toolkit
	Tivoli Directory Client
Tivoli Access Manager Web Portal Manager V5.1	Web Portal Manager (WebSphere enterprise application)
	WebSphere Application Server
	Global Security Toolkit
	Directory Client
	Access Manager Runtime
Tivoli Access Manager WebSEAL Server V5.1	Access Manager WebSEAL Server
	Access Manager Runtime
	Global Security Toolkit
	Tivoli Directory Client

10.1.3 Access Manager and WebSphere integration

Providing a standard-based authorization framework for WebSphere applications, Tivoli Access Manager supports the Java 2 security model as well as the Java Authentication and Authorization Services (JAAS) and Java 2 Enterprise Extensions (J2EE).

Integrating WebSphere and Access Manager adds WebSphere resources to the significant list of elements that can be managed via Tivoli Access Manager's consistent authorization policy, and it also adds to WebSphere applications the benefits that accrue in an Access Manager protected environment. The examples of this discussed in the previous chapters include URI-based access control, availability and scalability characteristics inherent in Access Manager implementations, and the ability to support many authentication mechanisms.

without any impact to the target application and Web single sign-on, which are fully applicable for WebSphere Application Server.

Figure 10-2 shows where WebSphere communicates with Access Manager components such as WebSEAL and the Policy Server, secure communications can be achieved at every single level between the different servers of an e-business infrastructure around Access Manager.

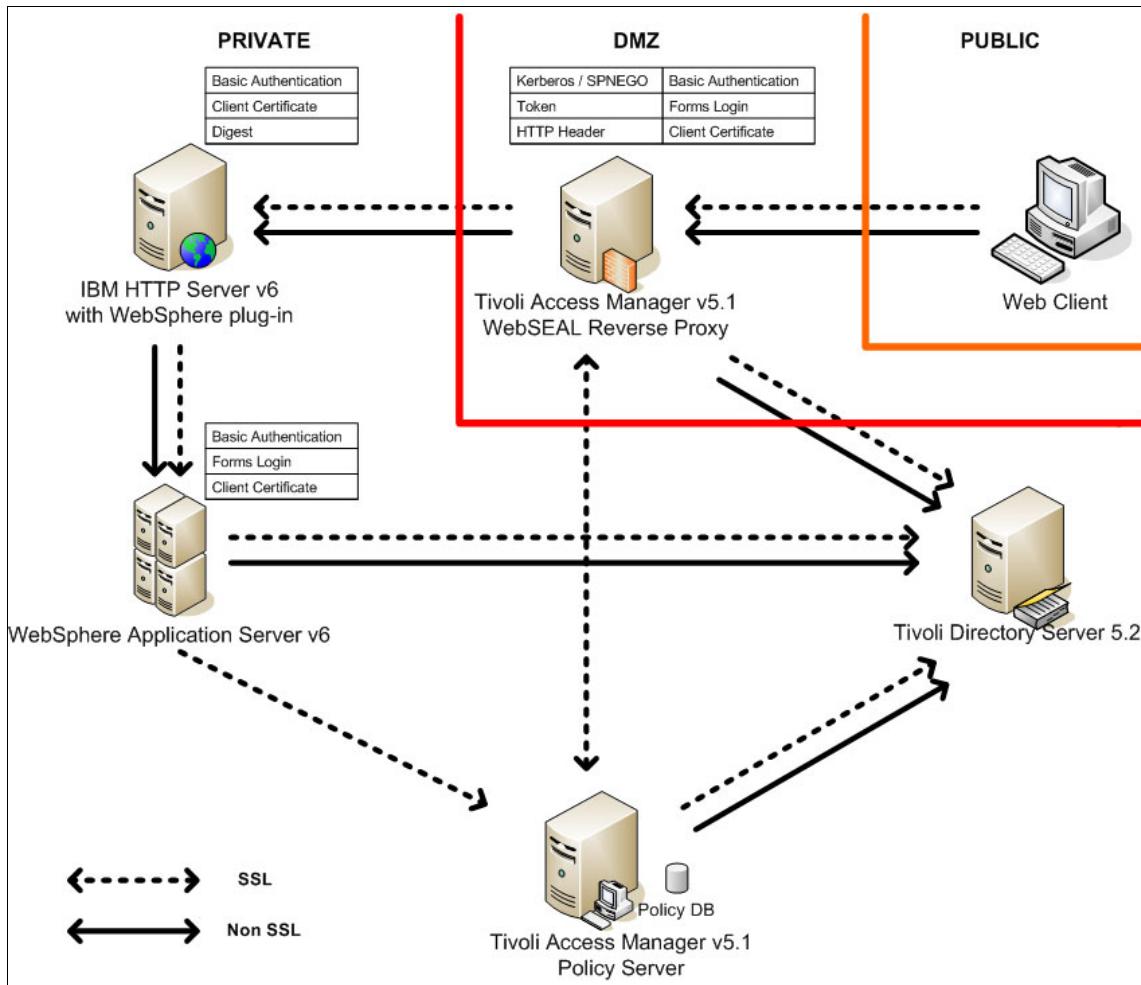


Figure 10-2 Tivoli Access Manager and WebSphere Communications

In the diagram, the arrows represent communication channels, dashed lines indicate that the communications are encrypted and the arrow heads indicate what component initiates the connection. The diagram shows boxes, at all the different points where the Web client could authenticate, listing the HTTP and

HTTPS authentication methods available. Not all the communications in the diagram must happen at the same time, but they are shown to give an idea of all the possibilities.

Some components could be taken out of the picture and the architecture will still be valid in some scenarios. For example, without WebSEAL, WebSphere could still use the Policy Server for its authentication decisions, the Web server is also an optional component, as WebSEAL could junction directly to the Application server, but generally it is considered good practice to use the Web server to serve static content. This chapter concentrates around the Tivoli Access Manager components; previous chapters have already covered the security around the IBM HTTP Server, WebSphere Application Server and the Tivoli Directory Server; please refer to those chapters for more information.

Communications to the Access Manager Policy server are always encrypted for security reasons but this is transparent to the other components as it is handled internally by the Access Managers runtimes. WebSEAL provides different authentication mechanisms from the client and integrates by using different authentication mechanisms with the backend servers to provide a true single sign-on even to Application Servers not aware of it.

The integration of WebSphere Application Server and Access Manager offers the following additional options/possibilities:

- ▶ Shared user registry
- ▶ Web single sign-on using:
 - Tivoli Global Sign-On (GSO) junctions
 - Web Trust Association Interceptor (TAI)
 - WebSEAL LTPA cookie support
- ▶ Application integration utilizing:
 - Authorization Application Programming Interface (aznAPI)
 - JAAS
 - JACC
 - PDPermission
 - J2EE security

10.2 IBM Tivoli Access Manager security model

The security policy for a Tivoli Access Manager secure domain is maintained and governed by two key security structures:

- ▶ User registry
- ▶ Policy database

10.2.1 User registry

The user registry (such as LDAP, Lotus Domino, or Microsoft Active Directory) contains all users and groups who are allowed to participate in the Tivoli Access Manager secure domain. In the example used in this book, the IBM Tivoli Directory Server LDAP directory contains the user registry shared by Tivoli Access Manager and WebSphere Application Server.

10.2.2 Master authorization (policy) database

The authorization database contains a representation of all resources in the domain (the protected object space). The security administrator can dictate any level of security by applying rules, known as access control list (ACL) policies, protected object policies (POP) and authorization rules, to those resources requiring protection

The Tivoli Access Manager authorization service enforces security policies by comparing a user's authentication credentials with the policy permissions assigned to the requested resource. The resulting recommendation is passed to the resource manager (for example, WebSEAL or WebSphere Application Server), which completes the response to the original request. The user credential is essential for full participation in the secure domain.

The protected object space

The protected object space is a hierarchical portrayal of resources belonging to an Access Manager secure domain. The virtual objects that appear in the hierarchical object space represent the actual network resources in the domain. They could be *system resources* - the actual file or application, and *protected objects* - the logical representation of an actual system resource used by the authorization service and other Access Manager management components.

Policy templates can be attached to objects in the object space to provide protection of the resources. The authorization service makes authorization decisions based on these templates.

These rules can be explicitly attached or inherited. The Access Manager protected object space supports inheritance of the security policies or rules. This is an important consideration for the security administrator who manages the object space. The administrator needs to apply explicit policies only at points in the hierarchy where the rules must change.

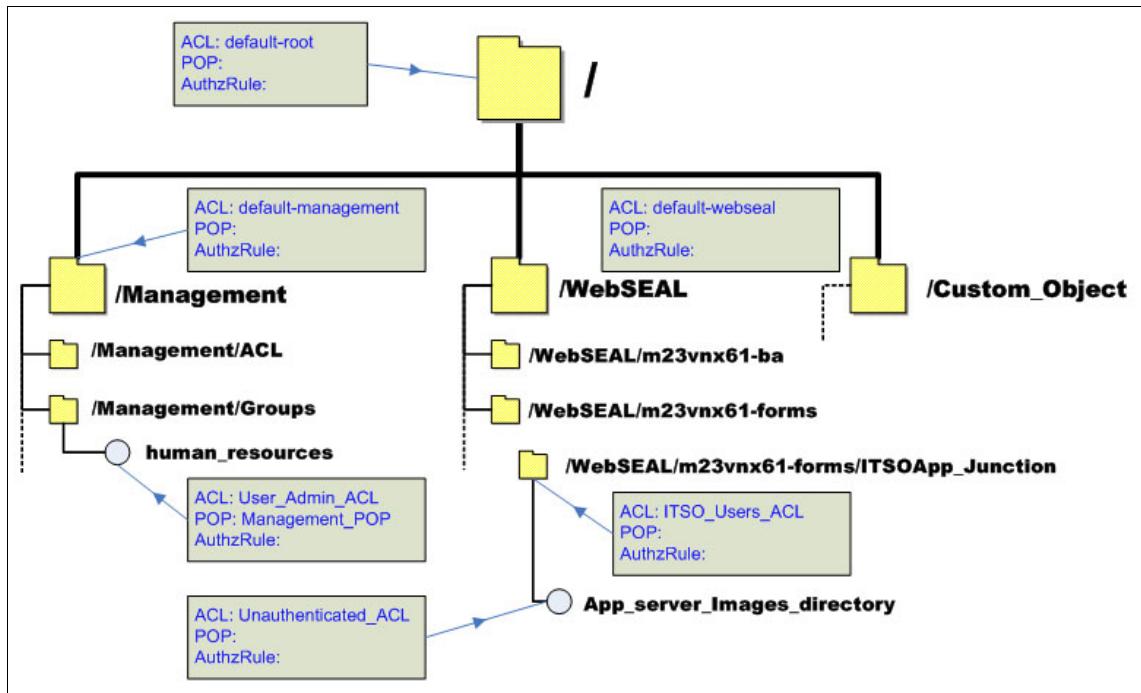


Figure 10-3 Tivoli Access Manager object space

The following object space categories are used by Access Manager:

- ▶ Web objects

Web objects represent any resource that can be addressed by a URL, including static and dynamic contents. The WebSEAL server is a component of Access Manager, responsible for protecting Web resources.

- ▶ Access Manager management objects

Management objects represent the management activities that can be performed through the Web Portal Manager. The objects represent the tasks necessary to define users and set security policy. Access Manager supports delegation of management activities and can restrict an administrator's ability to set security policy to a subset of the object space. An example of an Access Manager management object will be a defined group for example /Management/Groups/boardmembers; ACLs could be attached to the object to restrict who can add members to that group.

- ▶ User-defined objects

User-defined objects represent customized tasks or network resources protected by applications that access the authorization service through the Access Manager authorization API, for instance in library application you

could map actions to objects and allow *everyone* access to the object /library/book/summary but only allow *authenticated* access users to the object /library/book/reservation.

Access Manager Authorization Engine

The Access Manager authorization service performs authorization decisions based on the policies applied to the objects as explained earlier. For each request for access to an object inside the protected object space the request will be evaluated against the ACL, the POP and the Authorization rule attached to the object or inherited by it, in the order described. A single object can have none, one, two, or all three types of rule attached to it but only one of each type.

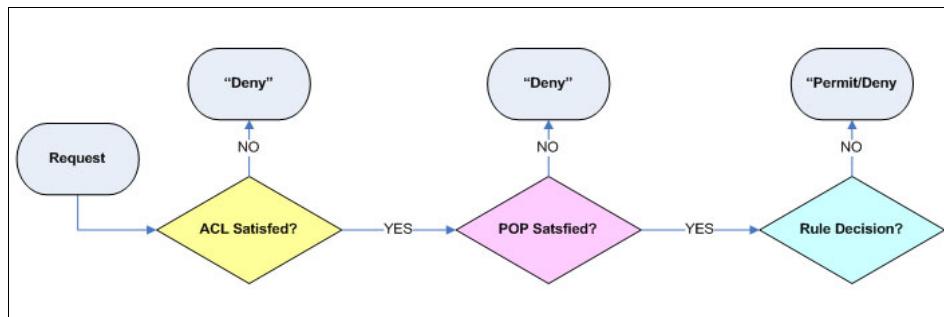


Figure 10-4 Access Manager Authorization Flow

Access control list (ACL)

An access control list policy, or ACL policy, is the set of rules (permissions) that specifies the conditions necessary to perform certain operations on a resource. ACL policy definitions are important components of the security policy established for the secure domain. ACL policies, like all policies, are used to stamp an organization's security requirements onto the resources represented in the protected object space. An ACL policy specifically controls:

- ▶ What operations can be performed on the resource.
- ▶ Who can perform these operations.

An ACL policy is made up of one or more entries that include user and group designations and their specific permissions or rights. An ACL can also contain rules that apply to unauthenticated users.

Protected object policies (POP)

ACL policies provide the authorization service with information to make a "yes" or "no" answer on a request to access a protected object and perform some operation on that object. POP policies contain additional conditions on the request that are passed back to the Access Manager Base and the resource

manager (such as WebSEAL) along with the "yes" ACL policy decision from the authorization service. It is the responsibility of Access Manager and the resource manager to enforce the POP conditions.

Authorization rules

An Access Manager authorization rule is a policy type similar to an access control list (ACL) or a protected object policy (POP). Authorization rules provide the flexibility needed to extend an ACL or POP by tailoring security policy to your needs. The rule is stored as a text rule within a rule policy object and is attached to a protected object in the same way and with similar constraints as ACLs and POPs.

Rules allow you to make decisions based on the attributes of a person or object and the context and environment surrounding the access decision. For example, you can use a rule to implement a time-of-day policy that depends on the user or group. You also can use a rule to extend the access control capabilities that ACLs provide by implementing a more advanced policy, such as one based on quotas. While an ACL can grant a group permission to write to a resource, a rule can go a step further by allowing you to determine if a group has exceeded a specific quota for a given week before permitting that group to write to a resource.

Note: For more detailed information about Tivoli Access Manager security and administration, refer to the following documents:

- ▶ *Base Administration Guide, IBM Tivoli Access Manager V5.1*, SC32-1360
- ▶ *WebSEAL Administration Guide, IBM Tivoli Access Manager V5.1*, SC32-1359
- ▶ *Enterprise Security Architecture Using IBM Tivoli Security Solutions*, SG24-6014
- ▶ *IBM Tivoli Access Manager for e-business*, REDP3677

10.3 Lab environment

To test some different Access Manager - WebSphere integration scenarios described in this chapter, we will be using a lab environment with all the elements as shown in Figure 10-2 on page 217; use the following diagram to understand the lab environment used for the examples.

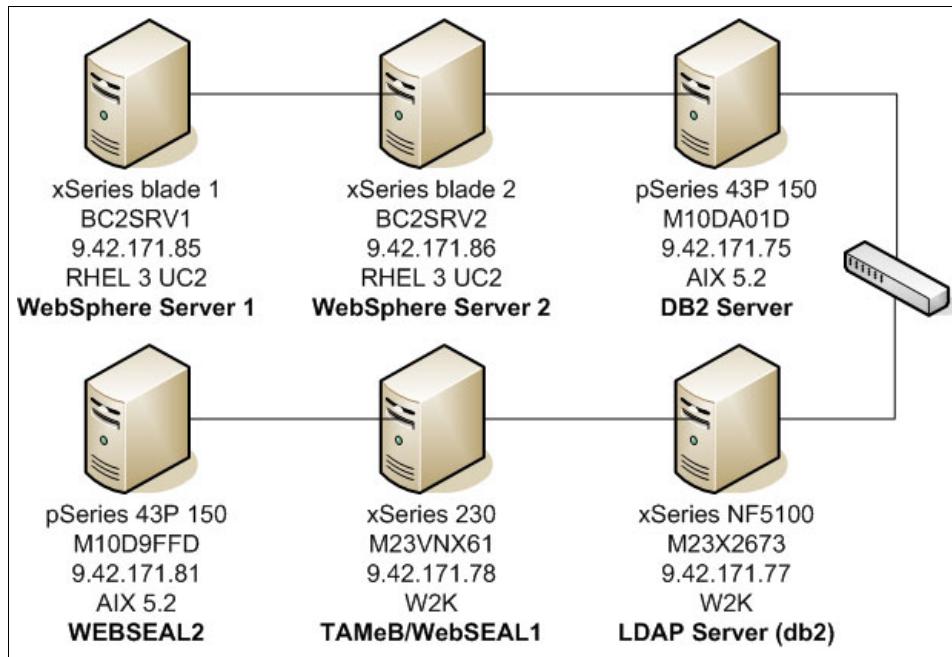


Figure 10-5 Lab environment

All the WebSphere Application Servers have the ITSOBank and ITSOHello installed, for details refer to Appendix A, “Additional configurations” on page 387.

They also have Security enabled with LDAP as the user registry, for details refer to Chapter 2., “Configuring the user registry” on page 7. The LDAP server is also the registry configured with Tivoli Access Manager.

10.4 Role of Tivoli Access Manager in WebSphere V6

The JACC provider in WebSphere Application Server is implemented by both the client and the server pieces of the Tivoli Access Manager server. The client piece of Tivoli Access Manager is embedded in WebSphere Application Server. The server piece is located on a separate installable CD that is shipped as part of the WebSphere Network Deployment (ND) package. The JACC provider is not an out-of-box solution. The following figure represents the high level components of the WebSphere embedded Tivoli Access Manager (Tivoli Access Manager Client) and Tivoli Access Manager Server.

In the following figure, there are conceptually three layers of integration point allowed in the embedded Tivoli Access Manager. It is provided in a hierarchical

fashion so that the function in each layer can become an integration point to fit into different application specific needs. The bottom layer, which is the lowest building block for the other two layers, includes a set of client-server components: the Access Manager Java Runtime (AMJRTE) component and the Tivoli Access Manager Server component. While the AMJRTE component serves the client-side integration point, the Tivoli Access Manager Server component provides the infrastructure for both runtime and management operations.

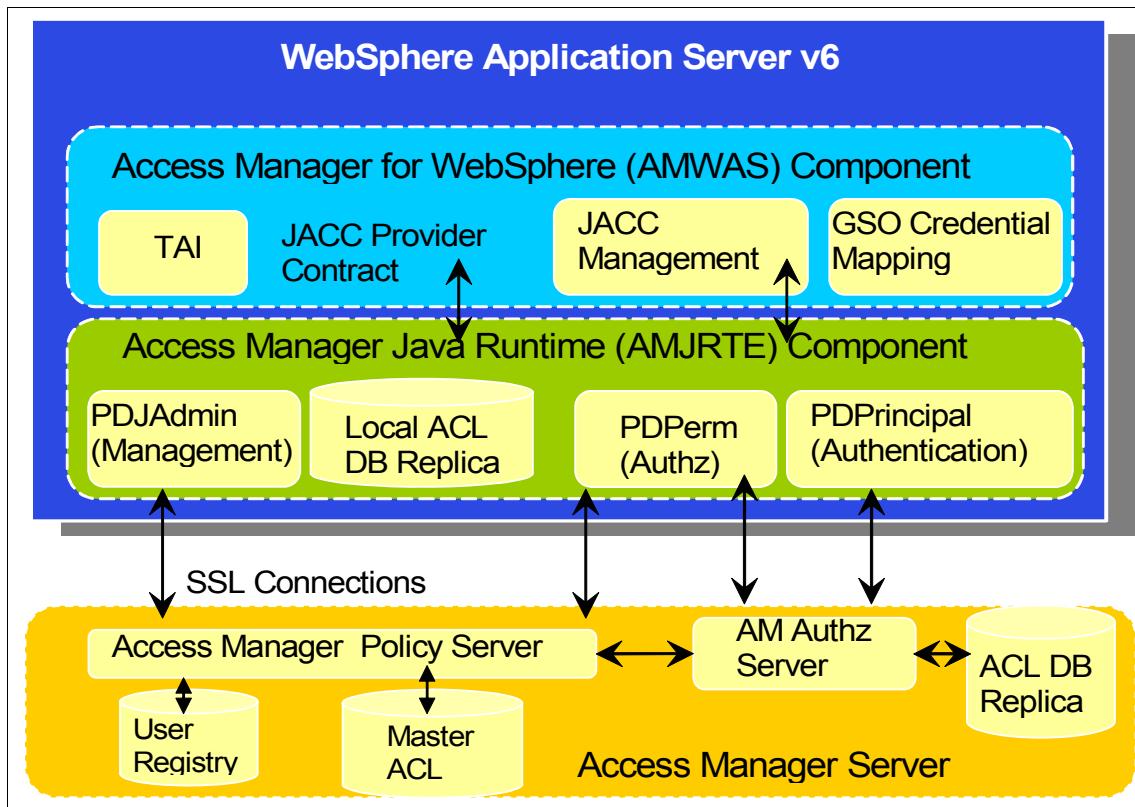


Figure 10-6 WebSphere and Access Manager relations

10.4.1 Embedded Tivoli Access Manager Client architecture

The following diagram depicts the Tivoli Access Manager client architecture in WebSphere Application Server V6.

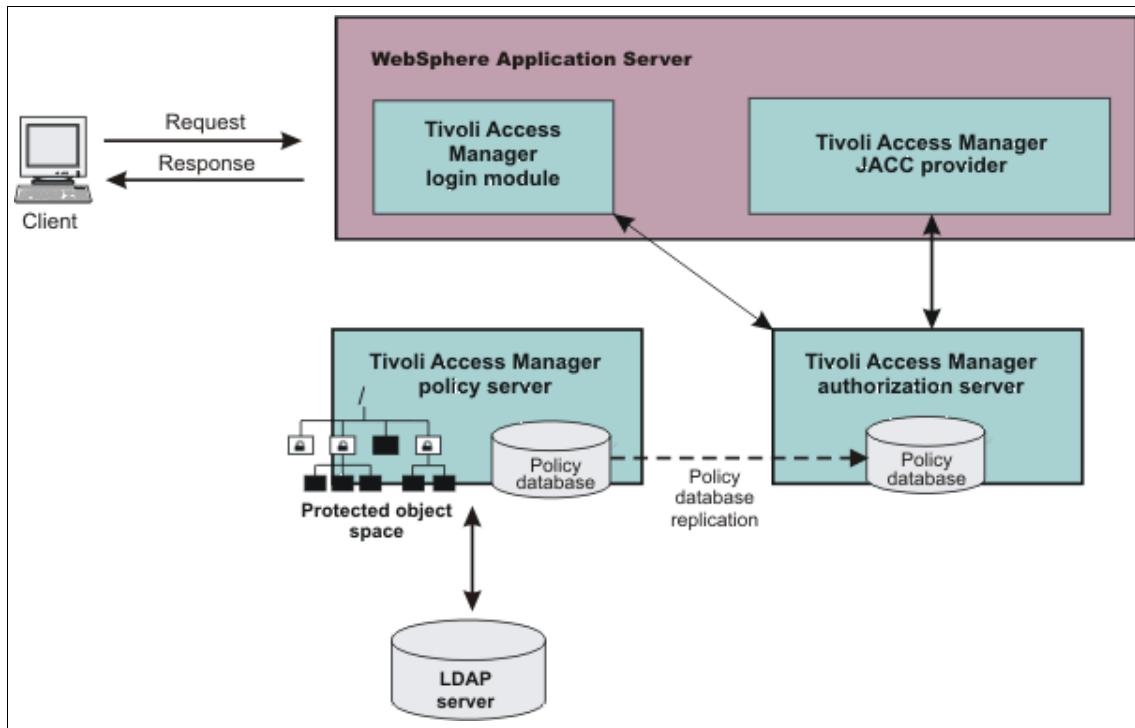


Figure 10-7 Embedded Tivoli Access Manager client architecture in WebSphere V6

1. Users that access protected resources are authenticated using the Tivoli Access Manager login module that is configured for use when the embedded Tivoli Access Manager client is enabled.
2. The WebSphere Application Server container uses information from the J2EE application deployment descriptor to determine the required role membership.
3. WebSphere Application Server uses the embedded Tivoli Access Manager client to request an authorization decision (granted or denied) from the Tivoli Access Manager Authorization server. Additional context information, when present, is also passed to the authorization server. This context information is comprised of the cell name, J2EE application name, and J2EE module name. If the Tivoli Access Manager Policy database has policies that are specified for any of the context information, the authorization server uses this information to make the authorization decision.
4. The authorization server consults the permissions that are defined for the specified user within the Tivoli Access Manager - protected object space. The protected object space is part of the policy database.

5. The Tivoli Access Manager Authorization server returns the access decision to the embedded Tivoli Access Manager client.
6. WebSphere Application Server either grants or denies access to the protected method or resource, based on the decision returned from the Tivoli Access Manager Authorization Server.

The participating WebSphere Application Servers use a local replica of the Tivoli Access Manager Policy database to make authorization decisions for incoming requests. The local policy databases are replicas of the master policy database that are installed as part of the Tivoli Access Manager installation. Having policy database replicas on each participating WebSphere Application Server optimizes performance when making authorization decisions and provides failover capability.

The authorization server can also be installed on the same system as WebSphere Application Server, although this configuration is not illustrated in the diagram.

All instances of Tivoli Access Manager and WebSphere Application Server in the example architecture share the Lightweight Directory Access Protocol (LDAP) user registry on Machine E, see Figure 10-8 on page 227.

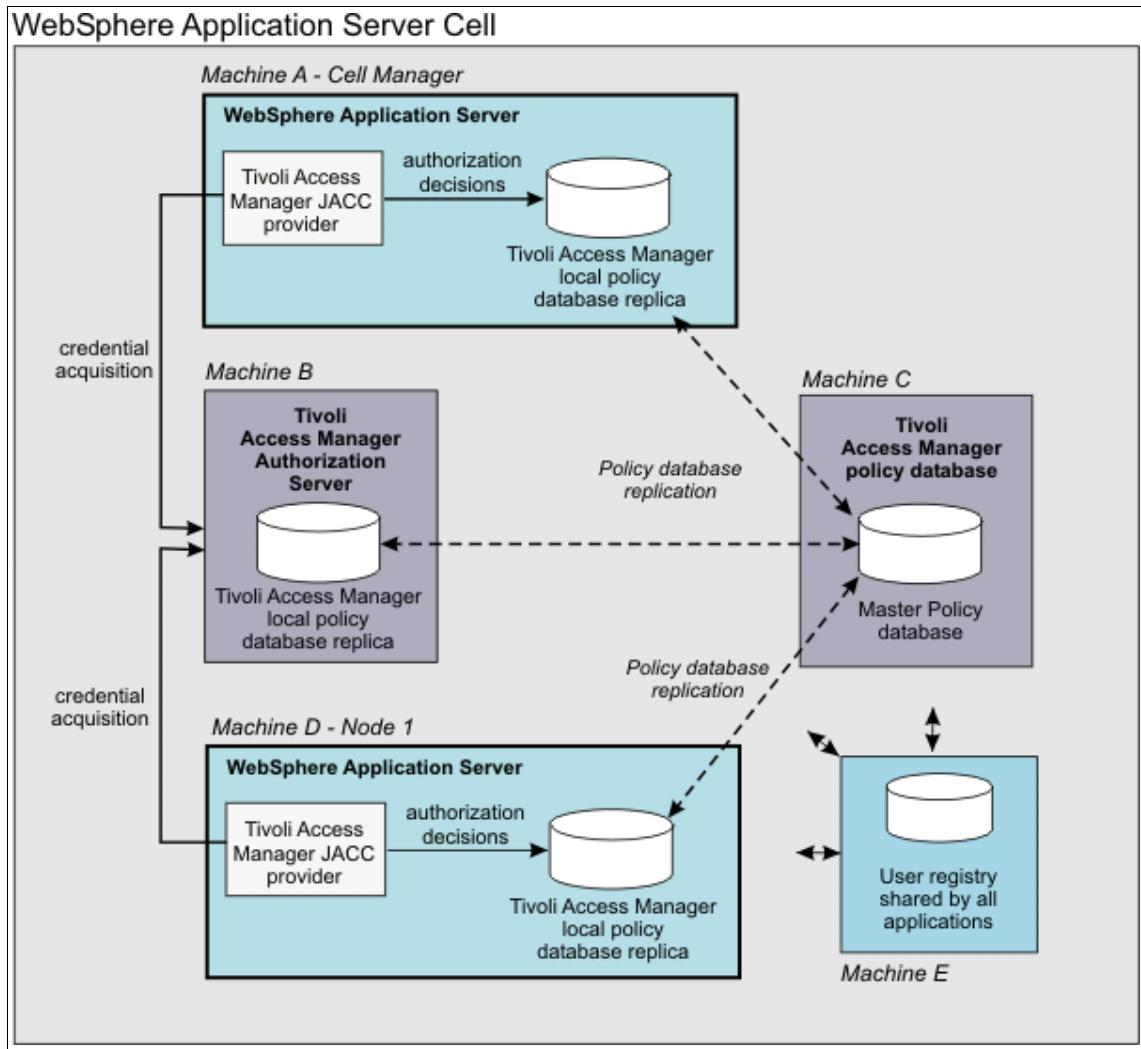


Figure 10-8 WebSphere V6 and Tivoli Access Manager in a sample architecture

The integration of Tivoli Access Manager in WebSphere (Tivoli Access Manager Client) using the JACC model to perform access checks can be divided into the following high level components.

- ▶ Runtime
- ▶ Client configuration
- ▶ Authorization table support
- ▶ Access check
- ▶ Authentication using the PDLoginModule

Tivoli Access Manager runtime support of JACC

Tivoli Access Manager implements the PolicyConfigurationFactory and the PolicyConfiguration interfaces, as required by JACC. During the application installation, the security policy information in the deployment descriptor and the authorization table information in the binding files are propagated to the Tivoli JACC provider using these interfaces. The Tivoli JACC provider stores the policy and the authorization table information in the Tivoli Access Manager policy server by calling the respective Tivoli Access Manager APIs. The information is stored in the security policy database in the Tivoli Access Manager Policy server.

Tivoli Access Manager Client configuration

The Tivoli Access Manager client can be configured using either the administrative console or wsadmin scripting. The administrative console panels for the Tivoli Access Manager Client configuration are located under the Security center panel. The Tivoli client must be set up to use the Tivoli JACC provider. The setup can be done either before (using wsadmin) or during the time of WebSphere Application Server configuration.

Authorization table support

Tivoli Access Manager uses the RoleConfiguration interface to ensure that the authorization table information is passed to the Tivoli Access Manager provider when the application is installed or deployed. When an application is deployed or edited, the set of users and groups for the user or group-to-role mapping are obtained from the Tivoli Access Manager server, which shares the same Lightweight Directory Access Protocol (LDAP) server as WebSphere Application Server. This sharing is accomplished by plugging into the application management users or groups-to-role administrative console panels. The management APIs are called to obtain users and groups rather than relying on the WebSphere Application Server-configured LDAP registry.

Access check

When WebSphere Application Server is configured to use the JACC provider for Tivoli Access Manager, it passes the information to Tivoli Access Manager to make the access decision. The Tivoli Access Manager Policy implementation queries the local replica of the access control list (ACL) database for the access decision.

Authentication using the PDLoginModule module

The custom login module in WebSphere Application Server can do the authentication. This login module is plugged in before the WebSphere Application Server-provided login modules. The custom login modules can provide information that can be stored in the Subject. If the required information is stored, no additional registry calls are made to obtain that information.

As part of the JACC integration, the Tivoli Access Manager-provided PDLoginModule module is also used to plug into WebSphere Application Server for both Lightweight Third Party Authentication (LTPA) and Simple WebSphere Authentication Mechanism (SWAM) authentication. The PDLoginModule module is modified to authenticate with the user ID or password. The module is also used to fill in the required attributes in the Subject so that no registry calls are made by the login modules in WebSphere Application Server. The information that is placed in the Subject is available for the Tivoli Access Manager Policy object to use for access checking.

10.5 WebSEAL authentication

This section will focus on the authentication from a client to the Access Manager WebSEAL reverse proxy. We will describe the configurations available and provide instructions for the most common authentication scenarios.

10.5.1 Basic authentication

By default, WebSEAL is configured for authentication over HTTPS using Basic Authentication (BA). If you want to enable Basic Authentication over HTTP the default below will have to changed accordingly. Edit the *webseald.conf* file and locate the [ba] stanza.

Example 10-1 Basic Authentication

```
# Enable authentication using the Basic Authentication mechanism
# One of <http, https, both, none>
ba-auth = https
```

If you decide to use basic authentication in your configuration you might want to consider changing the security realm displayed in the dialogue window by changing the basic-auth-realm setting, in Figure 10-9 on page 230 the result of changing it to the parameter below can be seen.

Example 10-2 WebSEAL authentication realm

```
# Realm name. This is the text that is displayed in the
# browser's dialog box when prompting the user for login data.
# By default, the string 'Access Manager' is used.
basic-auth-realm = ITS0 Applications
```

Restart the WebSEAL instance to make the changes effective. To test the settings point your browser to the root of your WebSEAL server.



Figure 10-9 Modified Basic Authentication dialogue

Once the user is logged in, the only way to close the WebSEAL session is that the user has to close the browser. The browser caches the credentials and automatically authenticates the user again even if WebSEAL closed the session.

10.5.2 Form-based authentication

In order to configure Form based Authentication in WebSEAL, you need to edit the *webseald.conf* file and then restart WebSEAL. Open *webseald.conf* file and locate the [ba] stanza and set ba-auth = none, then locate the [forms] stanza and change as below:

Example 10-3 Forms Authentication

```
# Enable authentication using the forms authentication mechanism
# One of <http, https, both, none>
forms-auth = https
```

Restart your WebSEAL instance to make changes effective, then you can test the configurations by accessing a protected page.

Once the user is logged in you want to close the WebSEAL session the user will need to close he browser or preferably the application could redirect the user to the *pkmslogout* page. After the user hit this page WebSEAL will destroy the session and displays the logout message.

Tip: If you are going to use Form based authentication you can tailor your login and logout pages to match with your applications design by modifying the login.html and logout.html in the directory:
<webseal_instace_root>/lib/html/C/

10.5.3 Client certificate based authentication

In order to configure certificate based authentication in WebSEAL, you need to edit the *webseald.conf* file and then restart WebSEAL. Using certificates to authenticate clients requires server and client configuration, on the WebSEAL side, open *webseald.conf* file and locate the [ba] stanza and set the ba-auth=none entry, then locate the [certificate] stanza and change as shown in Example 10-4.

Example 10-4 Certificate Authentication

```
# When to accept a certificate from HTTPS clients. Options are:
# never           Never request a client certificate.
# required        Always request a client certificate. Do not accept the
#                  connection if the client does not present a certificate.
# optional         Always request a client certificate. If presented, use it.
# prompt_as_needed Certificates will only be prompted for and processed when
#                  certificate authentication is necessary (due to an ACL or
#                  POP check failure).
# accept-client-certs = required
```

After doing the change, find the [authentication-mechanisms] stanza, uncomment the line and change <cert-ssl-library> for your cert-ssl library. Below you can see the change for AIX, please read the WebSEAL Administration Guide, IBM Tivoli Access Manager V5.1, SC32-1359 for information about the specific libraries for the operating system used.

Example 10-5 Certificate authentication library

```
# Certificates
cert-ssl      = libsslauthn.a
```

The second step is to create and load a client certificate into the client browser. If using self-signed certificates the certificate will also have to be loaded into the WebSEAL keystore as a signer certificate. If using your own Certificate Authority (CA) then the CA public key certificate will be loaded in the WebSEAL keystore as a signer certificate. WebSEAL does a one-to-one DN (Distinguished Name) matching of the certificate with LDAP. In this sample we will use a self-signed certificate. First we create the user for the sample user01, with the **user create** command in **pdadmin**, the Tivoli Access Manager Administration CLI (Command

Line Interface). The user must be made valid with the **user modify** command and then the information about the new user can be seen with the **user show** command in **pdadmin**.

Example 10-6 Access Manager user show command

```
# pdadmin -a sec_master
Enter Password:
pdadmin sec_master> user create -no-password-policy user01
cn=user01,ou=users,o=itso,c=US user01 " " test
pdadmin sec_master> user modify user01 account-valid yes
pdadmin sec_master> user show user01
Login ID: user01
LDAP DN: cn=user01,ou=users,o=itso,c=US
LDAP CN: user01
LDAP SN:
Description:
Is SecUser: Yes
Is GSO user: No
Account valid: Yes
Password valid: Yes
```

The next step is to create a self-signed certificate which exactly matches the following DN:

LDAP DN: cn=user01,ou=users,o=itso,c=US

You can create the certificate using the ikeyman tool, please refer to the Security Fundamentals, REDP3944, for more information about ikeyman and creating a self-signed certificate.

Extract the certificate as Base64-encoded ASCII data (*.arm file) to later import in the WebSEAL keystore. Also Export the certificate as PKCS12 (*.p12 file) to import it into the browser.

Use the ikeyman utility and open the WebSEAL keystore located at:
<webseal_instance_root>\certs\pdsrv.kdb, the default password is *pdsrv*.

Import the certificate you exported above with the .arm extension. Restart the WebSEAL instance to pick-up the changes.

Load the certificate into your browser, use the PKCS12 certificate (*.p12 file).

You can test the configuration by accessing a secured resource with your browser. You should be able to login without entering the username or password. Depending on your security settings and browser you might be presented with a certificate request, that allow you to chose the certificate to use.

10.5.4 Token authentication

Token authentication is used in a two-factor authentication, this is used when the users must provide two forms of identification. For example, a single factor of identification, such as a password, plus a second factor in the form of an authentication token. The two-factor method is based on something the user knows plus something the user possesses. It provides a more reliable level of user authentication than reusable passwords.

Tivoli Access Manager provides a built-in two-factor authentication library, *xtokenauth*. It is a client implementation for the RSA SecurID token authentication server (ACE/Server) and is written against the RSA Authorization API. WebSEAL provides RSA token authentication client (ACE/Agent) functions, and is certified as SecurID Ready.

By default, this built-in shared library for token authentication is hard-coded to map SecurID (RSA) token passcode data. This default token authentication mechanism expects the user name used by the client to map to an existing user account in the Access Manager LDAP registry.

For information about configuring Token Authentication please refer to the *WebSEAL Administration Guide, IBM Tivoli Access Manager V5.1,SC32-135*.

10.5.5 HTTP header authentication

Tivoli Access Manager WebSEAL provides an authentication module that authenticates users based on information obtained from custom HTTP headers supplied by the client or a proxy agent. This module consists of a mapping function that maps header data to an Access Manager identity.

WebSEAL trusts that this custom HTTP header data is the result of a previous authentication. The WebSEAL authentication module is built specifically to map data obtained from Entrust Proxy headers. When you enable HTTP header authentication using the built-in authentication module, you should disable all other authentication methods. You should accept connections only from the Entrust Proxy. By disabling other authentication methods eliminates methods that could be used to impersonate custom HTTP header data.

For information about configuring Token Authentication please refer to the *WebSEAL Administration Guide, IBM Tivoli Access Manager V5.1,SC32-135*.

10.5.6 Kerberos and SPNEGO authentication

WebSEAL supports the SPNEGO protocol and Kerberos authentication for use with Windows clients to achieve Windows desktop Single Sign-On. The

SPNEGO protocol allows for a negotiation between the client (browser) and the server regarding the authentication mechanism to use. The client identity presented by the browser can be verified by WebSEAL using Kerberos authentication mechanisms.

WebSEAL's support for Kerberos authentication has been implemented specifically to support a Windows desktop Single Sign-On solution. This solution requires that the WebSEAL server be configured into an Active Directory domain, and that WebSEAL be able to access a Kerberos Key Distribution Center (KDC). In addition, the Internet Explorer client must be configured to use the SPNEGO protocol and Kerberos authentication when contacting WebSEAL.

For information about configuring Kerberos SPNEGO Authentication please refer to the WebSEAL Administration Guide, IBM Tivoli Access Manager V5.1, SC32-135.

10.6 WebSEAL junctions

The purpose of authenticating to WebSEAL is to access its protected resources, although WebSEAL provides minimum Web server functionality most commonly the resources protected will be on backend servers. WebSEAL's connections with the back-end Web servers have constantly been referred to as *junctions*. All WebSEAL junctions are connections between a front-end WebSEAL server and a back-end Web server which may be another WebSEAL server and may go via another proxy server. Only the HTTP and HTTPS protocols are supported and WebSEAL to WebSEAL connections must have SSL enabled.

A junction is where the back-end server Web space is connected to the WebSEAL server at a specially designated mount point in the Access Manager Web space created in the Policy Server database by appropriate use of the **pdadmin** command.

The junction is then a logical Web object space, normally on another Web server, rather than the physical file and directory structure of the proxied Web server. Junctions define an object space that reflects organizational structure rather than the physical machine and directory structure commonly encountered on standard Web servers. A browser client never knows the physical location of a Web resource as WebSEAL translates the requested URL addresses into the addresses that a back-end server expects without ever exposing them to the client. Web objects can be moved from server to server without affecting the way the client accesses those objects.

WebSEAL attempts to pass the request to the back-end server by referencing the object in Access Manager's protected object space. If it encounters an ACL or Policy of Protection, POP on that object which requires authentication before the request can be authorized, then the client will be challenged. WebSEAL is configurable for several different challenge mechanism including the default of Basic Authentication, forms based logon from a junctioned application and comes with an Application Developers Kit with which to build customized Cross Domain Authentication Services.

WebSEAL junctions can also be configured to enable the creation of Single Sign-On solutions allowing users to access resources, somewhat regardless of what security domain controls those resources, following their initial authentication logging on to through WebSEAL. The Global Sign-On (GSO) junction option allows for a third-party user registry to be referenced to supply that junction with the appropriate user ID and password. Other options involve manipulation and perhaps additions to the underlying Access Manager schema of *inetOrgPerson*, as each junction can be configured to supply all and any attributes from the schema through to the back-end server. If the logon identity and passwords from the user registries of several legacy applications can be migrated into extra attributes then those applications can be accessed through WebSEAL using only one initial login. Any further login requirements from back-end servers are handled as transparent to the user.

There are also Cross Domain Single Sign-On and e-Community Single Sign-On solutions. These solutions allow for the transfer of Access Manager user credentials across multiple security domains. Please reference the Tivoli documentation for more details at:

http://publib.boulder.ibm.com/tividd/td/ITAME/SC32-1359-00/en_US/HTML/am51_webs_eal_guide.html

10.6.1 Simple junctions

pdadmin is a simple, easy to use command line utility for administration. You can also use the Tivoli Access Manager Web Portal Manager which provides a graphical interface. Before creating junctions you need to login to the secure domain using `sec_master` user ID as shown:

```
pdadmin -a sec_master  
password: Enter your password for sec_master  
pdadmin sec_master>
```

You can get a list of configured WebSEAL servers by using the server list command

```
pdadmin sec_master> server list  
default-webseald-m23vnx61
```

```
default-webseald-m10d9ffd  
ivacld-m23vnx61
```

From this server list output, you can choose the server required, for example default-webseald-m23vnx61, for junction creation. There are three required options for creating basic WebSEAL junctions:

- ▶ -h: hostname of the backend junctioned server
- ▶ -t: junction transport type, the options are: tcp, ssl, tcpproxy, sslproxy, local
- ▶ junction point name

The syntax for creating a basic junction is

```
server task webseal-instance_name create -t transport_type -h host_name  
jct_point_name
```

For example:

```
server task default-webseald-m23vnx61 create -t tcp -h bc2srv2.itso.ral.ibm.com  
/test
```

A junction can be configured to insert Tivoli Access Manager specific client identity and group information into the HTTP header by using the -c option. This information can then be passed to the backend servers which can use this information from the HTTP Header. There are four options that you can use with -C:

- ▶ **iv-user** - Passes the short name or the long name. Defaults to unauthenticated if the client is unauthenticated.
- ▶ **iv-user_I** - Passes full Distinguished Name of the user.
- ▶ **iv-groups** - Passes a list of comma separated groups to which the client belongs.
- ▶ **iv-creds** - This is encoded opaque data structure representing an Access Manager credential. This is used by the new TAI to create a PDPrincipal object and insert that object into the Subject.

Note: -c all adds all of the four options mentioned above.

You can also list the options using a comma as a separator.

A junction can be configured to supply client identity in the Basic Authentication (BA) header by using the -b option when creating the junctions. This is different from the -c option discussed earlier. When configuring a junction for use with the Trust Association Interceptor (TAI) of WebSphere application Server, you will need to configure your junction with the -b supply option. This option inserts the *dummy* password configured in the *webseald.conf* file in BA header. This *dummy*

password is used in the WebSEAL TAI to establish trust between the participating WebSEAL servers and WebSphere Application Servers. You can use the -f option to force a new junction to overwrite an existing junction mount point. Here is an example of using the options described above:

```
server task default-webseald-m23vnx61 create -t ssl -h bc2srv2.itso.ral.ibm.com  
-f -c iv_user,iv_creds /test
```

This creates a new junction test that overwrites the existing junction. To view the details of this junction use the server task <webseal server> show <junction-name> command.

Example 10-7 Access Manager show junction command

```
pdadmin sec_master> server task default-webseald-m23vnx61 show /test  
Junction point: /test  
Type: TCP  
Junction hard limit: 0 - using global value  
Junction soft limit: 0 - using global value  
Active worker threads: 0  
Basic authentication mode: supply  
Forms based SSO: disabled  
Authentication HTTP header: insert - iv_user iv_creds  
Remote Address HTTP header: do not insert  
Stateful junction: no  
Boolean Rule Header: no  
Scripting support: yes  
Preserve cookie names: no  
Delegation support: no  
Mutually authenticated: no  
Insert WebSphere LTPA cookies: no  
Insert WebSEAL session cookies: no  
Request Encoding: UTF-8, URI Encoded  
Server 1:  
ID: f33eb906-28f1-11d9-9d0e-0002557c751d  
Server State: running  
Hostname: bc2srv2.itso.ral.ibm.com  
Port: 80  
Virtual hostname: bc2srv2.itso.ral.ibm.com  
Server DN:  
Query_contents URL: /cgi-bin/query_contents  
Query-contents: unknown  
Case insensitive URLs: no  
Allow Windows-style URLs: yes  
Total requests : 2437
```

Note: If the communications channel between WebSEAL and the junctioned backend server is not secured, use SSL junctions to ensure security.

10.6.2 Single Sign-On junctions

For most cases applications in a secured production environment behind several security measures as protected environments and firewalls can safely be configured to trust WebSEAL and get the user identity by using the headers provided, but sometimes there is a need to integrate backend servers that require authentication and cannot or will not be modified to support better methods such as Trust Association Interceptor or LTPA as explained in the previous section. In these cases WebSEAL provides mechanisms to authenticate to the servers on behalf of the users to Web servers or application servers transparently, without them being aware that Access Manager is handling the authentication.

Basic Authentication

You can configure WebSEAL junctions to supply the back-end server with original or modified client identity information. The set of -b options allows you to supply specific client identity information in the HTTP Basic Authentication (BA) headers. After the initial authentication between the client and WebSEAL, WebSEAL can build a new Basic Authentication header. The request uses this new header as it continues across the junction to the back-end server. You use the -b options to dictate what specific authentication information is supplied in this new header. There are four options available:

- ▶ **supply**: the authenticated Tivoli Access Manager user name with a static, generic dummy password will be used. The original client password is not used in this scenario. This is what it is used for TAI junctions as explained in the previous section.
- ▶ **ignore**: passes the original client Basic Authentication (BA) header straight to the back-end server without interference. WebSEAL can be configured to authenticate this BA client information or ignore the BA header supplied by the client and forward the header, without modification, to the back-end server.
- ▶ **filter**: this will cause WebSEAL to remove the BA header from any client requests, thus ensuring that WebSEAL is the single security provider.
- ▶ **gso**: this is the option used when the backend server requires different user name and or passwords to authenticate. In order to use Global Sign-On (GSO), a GSO resource credential database need to be configured. The user registry contains some extra data for each user beyond the Access Manager required data, the GSO data is a list of gso resource-username-password entries. Please refer to the Tivoli Access Manager documentation for more information about GSO.

Configuring a junction to authenticate to a server using Basic Authentication

In the following example the client authenticates to WebSphere Application Server using basic authentication. We will use the snoop application (WebSphere default application sample) to test a junction that requires Basic Authentication and shares the same user registry as the policy server. To create the junction follow the steps below:

1. From the Access Manager server use the **pdadmin** command line tool and log on as the **sec_master**.

2. List the WebSEAL servers to find the one for configuring the junction:

```
pdadmin sec_master> server list
      default-webseald-m10d9ffd
      ba-webseald-m10d9ffd
      forms-webseald-m10d9ffd
```

3. In this case we will use the server **ba-webseald-m10d9ffd** which listens on IP:port 9.42.171.81:444 on HTTPS and is configured to use Basic Authentication, in this sample we will create a junction to the WebSphere Server which listens on IP:port 9.42.171.85:9080 on HTTP. The junction will use the **-b ignore** option to pass on the BA header from WebSEAL to WebSphere transparently. Enter the following command in the pdadmin command line:

```
pdadmin sec_master> server task ba-webseald-m10d9ffd create -t tcp -h
      9.42.171.85 -p 9080 -b ignore /SnoopApp
```

If you access the snoop application directly using the address <http://9.42.171.85:9080/snoop> the Basic Authentication challenge comes from WebSphere. In order to test the junction we will access the same application through the WebSEAL server. Enter the <https://9.42.171.81:444/SnoopApp/snoop> in your browser. You should be presented with the WebSEAL Basic Authentication header as in Figure 10-9 on page 230.

Enter a valid user name and password from the LDAP registry, for example **user01/test**, WebSEAL will authenticate to WebSphere and present the snoop servlet output, as the next figure shows, notice that the address is that of WebSEAL, and the address that was requested to WebSphere is in the response.

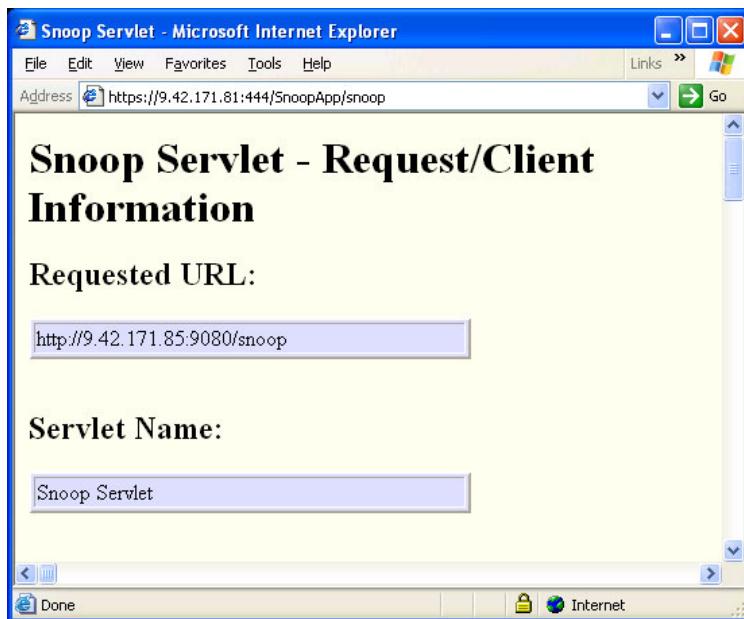


Figure 10-10 WebSEAL Basic Authentication SSO

Form-based authentication

To enable Single Sign-On forms authentication to a back-end application, the Access Manager administrator must perform two tasks. Firstly, a configuration file must be created defining to WebSEAL how to identify a login form when it is received from the back-end application and which fields in the backend server form are relevant for the authentication. Secondly, a junction must be created to the back-end Web server using the `-s` option, which specifies the location of the configuration file. Once this is completed, WebSEAL will provide login support for Access Manager users to the back-end WebSphere application.

For further information about enabling single-sign on forms authentication, refer to the *Access Manager for e-business WebSEAL Administrators Guide*.

Creating the form based authentication configuration file

The purpose of the configuration file for Single Sign-On forms authentication is to define the following to WebSEAL:

- ▶ A pattern which WebSEAL can use to identify the URI which indicates a request to the back-end application for a login form.
- ▶ A pattern which WebSEAL can use to identify the login form with a page returned from the back-end application.

- ▶ A list of fields within the login form which WebSEAL is to provide the values for, and where these values are to be obtained.

The following example is the source for a sample login page for the ITSOBank sample application.

Example 10-8 ITSOBank - login.html

```
<form method="post" action="/itsobank/j_security_check">
<table width="80%">
<tr>
<td width="20%" align="right">UserId:</td><td><input size="20" type="text"
name="j_username" maxlength="25"></td>
</tr>
<tr>
<td align="right">Password:</td><td><input size="20" type="password"
name="j_password" maxlength="25"></td>
</tr>
<tr>
<td></td>
<td>
<input type="submit" name="action" value="Login">&ampnbsp<input type="reset"
name="reset" value="Clear">
</td>
</tr>
</table>
</form>
```

In our form, there are two input fields, *j_username* and *j_password*. These are the two fields which WebSEAL will need to fill in. The following example shows the Single Sign-On forms configuration file.

Example 10-9 Single Sign-On forms authentication configuration file

```
[forms-sso-login-pages]
login-page-stanza = login-itsobank
[login-itsobank]
login-page = /itsobank/login/login.html
login-form-action = *
gso-resource =
argument-stanza = args-for-login-itsobank
[args-for-login-itsobank]
j_username = cred:azn_cred_authzn_id
j_password = string:test
```

In this example, we have configured one login form page, `login-itsobank`. The URI for the login form is `/itsobank/login/login.html`. This entry defines the URI that should be intercepted by WebSEAL. When a request is received for this URI, WebSEAL will intercept the form, and will return to our ITSOBank application the user ID defined for this Access Manager user and the fixed password test.

The forms configuration also allows to use GSO resources, although we did not use it in this example. The users could be easily created in the backend systems with the same password or no-password if the infrastructure and WebSEAL provide a secure environment. This sample is similar to the `-b supply` option discussed earlier. To create the junction, follow the steps below:

1. Create the file `itsobank.fssso.conf` in the `<WebSEAL_Install_dir>/etc` directory. Ensure that the file is readable by the `ivmgr` user.
2. On the Access Manager server launch `pdadmin` and log on as `sec_master`.
3. Find the WebSEAL server we are going to use:

```
pdadmin sec_master> server list
    default-webseald-m10d9ffd
    ba-webseald-m10d9ffd
    forms-webseald-m10d9ffd
```

4. In this case we will use the server `ba-webseald-m10d9ffd` which listens on IP:port 9.42.171.81:444 on HTTPS and is configured to use Basic Authentication. We will create a junction to the WebSphere Application Server which listens on IP:port 9.42.171.85:9082 on HTTP. The junction will use the `-s` option to indicate the forms Single Sign-On file. Enter the following command:

```
pdadmin sec_master> server task ba-webseald-m10d9ffd create -t tcp -h
9.42.171.85 -p 9082 -f -S /opt/pdweb/etc/itsobank.fssso.conf /ITSOBank
```

To test the junction, access the WebSEAL server, the ITSOBank junction and the protected resource `/itsobank/transfer/customertransfer.html` in.

10.7 Trust Association Interceptor (TAI)

In many enterprises there are Web proxy authentication servers like WebSEAL that can be used to perform authentication. When WebSphere is deployed in such scenarios, it is essential to establish a trust relationship between the application server and the third party security server like WebSEAL. This *trusted relationship* between WebSphere Application Server and WebSEAL is established by using a specific Trust Association Interceptor built for WebSEAL. Thus Trust Association Interceptor (TAI) enables the integration of third party security server with WebSphere Application Server.

Trust Association Interceptor is an Interface that is provided by WebSphere Application Server. This interface must be implemented for the specific Web proxy servers by the vendor alone or in conjunction with IBM. The implementation of this interface determines the contract used between WebSphere Application Server and the proxy server to establish trust. In WebSphere Application Server V6 there is a new TAI Interface that has significantly enhanced features.

Note: You can also use the old TAI interface `com.ibm.websphere.security.TrustAssociationInterceptor`. For more information about the old version of TAI, refer to the IBM Redbook *WebSphere Security V5*, SG24-6573.

10.7.1 New, enhanced TAI Interface

The new TAI Interface has been introduced in WebSphere Application Server V5.1.1. This TAI interface, `com.ibm.wsspi.security.tai.TrustAssociationInterceptor`, has been enhanced with the following new features:

- ▶ Support for a multi-phase negotiation during the authentication process.
- ▶ *TAIResult* is returned by the TAI and it indicates if more negotiation is required or the negotiation process is completed.
- ▶ Trust Association Interceptor is capable of asserting the userID and group information to WebSphere Application Server.
- ▶ Custom information may be added to the subject during the TAI processing and these can be returned as a JAAS Subject and can be used in application code.

There are two key methods in the new interface:

- ▶ `public boolean isTargetInterceptor(HttpServletRequest req) throws WebTrustAssociationException`

This method determines if the request originated from one of the proxy servers associated with the Trust Association Interceptor. The code in this method must determine whether the incoming request originated from one of the configured Proxy Servers by examining the request object. The result of this method may be True or False. True value tells WebSphere Application Server to continue the processing of TAI. In case of false, the TAI is ignored.

- ▶ `public TAIResult negotiateValidateandEstablishTrust(HttpServletRequest req, HttpServletResponse res) throws WebTrustAssociationFailedException`

The code in this method must determine whether to trust the proxy server the request originated from. This code is proxy server specific and should

authenticate the proxy server in some meaningful way. Also this method enables the TAI to use a trust negotiation protocols like SPNEGO to provide challenge and responses back to the client.

The return value of negotiateValidateandEstablishTrust is a *TAIResult*. This object indicates the status of negotiation or the final result of negotiation. The TAIResult class has three static methods for creating a TAIResult.

- ▶ `create(int status)`

You can set the int to something other than HttpServletResponse.SC_OK and the HttpServletResponse will be sent back to the client which will make another request to the TAI.

- ▶ `create(int status, String principal)`

You can set the status to HttpServletResponse.SC_OK and then provide the user ID or the unique ID for this user. WebSphere Application Server then queries the registry with this ID to get additional information for in order to create the credentials.

- ▶ `create(int status, String principal, Subject subject)`

You can set the status to HttpServletResponse.SC_OK thus indicating that no further negotiation is needed. WebSphere Application Server will create the Subject using the information provided in principal and subject.

There are a few additional methods on the TrustAssociationInterceptor interface. These additional methods are used for TrustAssociationInterceptor initialization, shut down, and its identification. For more information about these methods refer the WebSphere Application Server V6 Information Center or the WebSphere Application Server Java API documentation.

10.8 IBM WebSphere Application Server and WebSEAL integration

This section discusses the different integration scenarios between WebSphere Application Server and WebSEAL.

10.8.1 Integration options

There are various options to setup Single Sign-On between WebSphere Application Server and Access Manager's WebSEAL.

Using Trust Association Interceptor (TAI)

You can set up the TAI in two ways:

- ▶ With a trusted user
- ▶ With a trusted connection

Trusted user

In this configuration, the TAI identifies the WebSEAL server using the Basic Authentication header. A trusted user is created in LDAP and the TAI is configured with that userID. Only the password (not the userID) is placed on the Basic Authentication header by WebSEAL. This represents a "shared secret" which only the TAI and the WebSEAL server know.

During runtime, the TAI examines the password and validates with the user registry that the password belongs to the trusted user. This procedure enables the TAI to trust that it really is the WebSEAL server asserting the end user's identity, and the TAI can therefore trust it. To set up the WebSEAL junction to use the Basic Authentication header to identify the WebSEAL server, you use the *-b supply* option with the junction creation command. WebSEAL builds the Basic Authentication header using the password, which is specified in the *Webseald.conf* file (*basicauth-dummy-passwd* property). We have described this set up in detail later on.

Trusted connection using mutual SSL

In this configuration, the WebSEAL server identifies and authenticates itself to the Web server using its own client-side certificates. In this case, the TAI will do no further validation of the WebSEAL server hosts. This configuration is set in TAI using the *com.ibm.websphere.security.WebSEAL.mutualSSL=true* setting. For the new TAI interface, also set the *com.ibm.websphere.security.webseal.checkViaHeader=false* property. With these settings the TAI validates the WebSEAL host using the *hostname* property, and does no further validation. It assumes that the connection from WebSEAL to application server is completely trusted.

Note: This setup requires an SSL junction. You set up an encrypted junction using SSL with client certificates.

Using Lightweight Third Party Authentication (LTPA)

With LTPA, you do not have to configure a TAI for the application server. Instead, you configure an LTPA junction.

10.8.2 Configuration for the Trust Association Interceptor approach

This section provides detailed description about the configuration for using Trust Association Interceptors.

Flow of tokens from WebSEAL to WebSphere Application Server

WebSEAL provides authentication and authorization to all requests before passing them to the junctioned Web server. In the figure below WebSEAL is located in the Demilitarized zone.

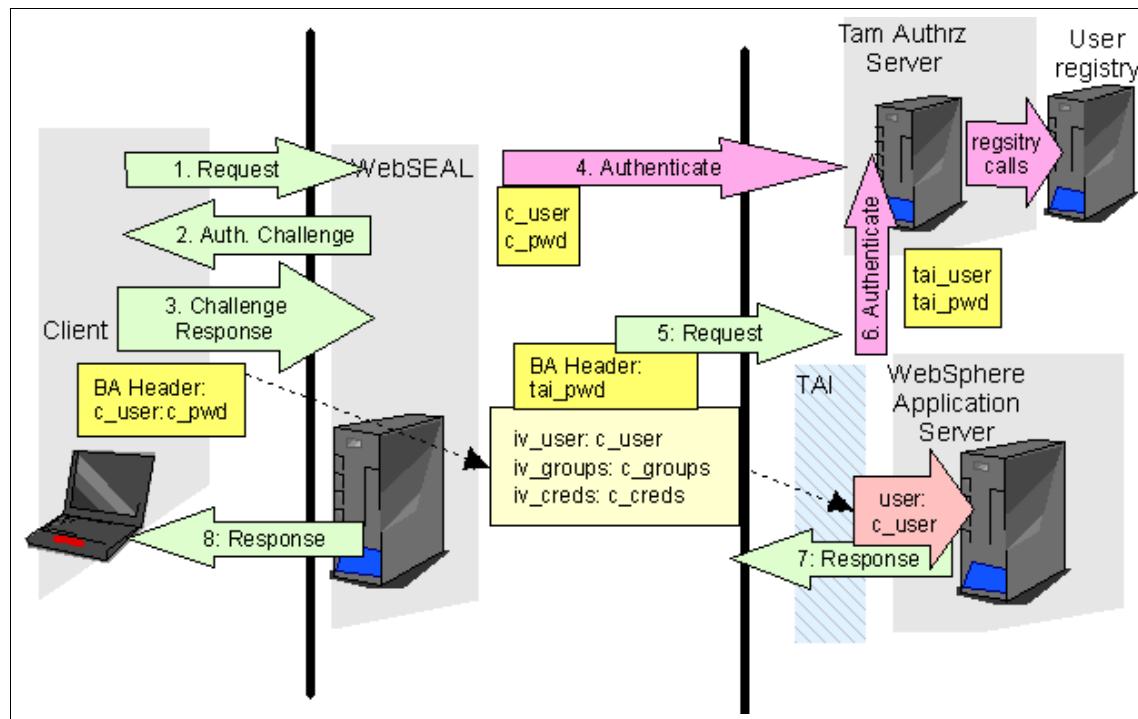


Figure 10-11 Request flow when using the TAI

1. An unauthenticated client issues a request for a secure resource which is intercepted by the reverse proxy (WebSEAL).
2. WebSEAL issues an HTTP authentication challenge to the client. Note that WebSEAL could be configured to provide a login form instead, but the overall flow of information would remain the same.

3. The client responds to the authentication challenge with a new request containing the client's userID (c_user) and password (c_pwd) in the HTTP Basic Authentication (BA) Header.
4. WebSEAL authenticates the user against the user registry using the c_user and c_pwd values. WebSEAL also authorizes the request based on ACL that is configured for the junction.
5. WebSEAL modifies the BA Header so that it includes the TAI password configured in the WebSEAL configuration file. WebSEAL also attaches the client's userID and group membership and credentials into an additional HTTP headers (iv_user, iv_groups, and iv_creds) that are sent along with the request to the application server.
6. The request goes to the application server, where the Trust Association Interceptor (TAI) intercepts the request for further security processing. The TAI extracts the credential information from the incoming request from WebSEAL. The TAI performs the authentication for the configured tai_user (using the configured tai_pwd). This authentication ensures that the TAI, together with the application server, is trusted.
7. The user credentials are extracted from the request by the TAI and used to construct a PDPrincipal object in the application server. A credential object containing user information is constructed from information contained in the PDPrincipal. The Principal and the Credential objects are inserted into a JAAS Subject which is returned from the call.
8. WebSphere sends the output to WebSEAL.
9. WebSEAL dispatches the output to the client.

A few additional comments for the TAI:

- ▶ WebSphere Application Server does not query the registry directly for Trust Association Interceptor processing. The new Interceptor class TAMTrustAssociationInterceptorplus contacts the Tivoli Access Manager Authorization Server which does the check with the user registry. This also indicates that additional configuration is required to ensure that WebSphere Application Server can contact the Authorization server.
- ▶ It is possible to negotiate with the client in a multiphase handshake. Keep in mind that the Tivoli Access Manager TAI does not use the multiphase negotiation nor does it need to. There are security protocols that may require negotiation, for example: SPNEGO. TAI that can handle negotiation can be developed using the new, extended interface.
- ▶ With the new TAI, you can add custom attributes to the Subject in the form of Java sets.
- ▶ You can continue to use the old WebSEAL TAI class called WebsealTrustAssociationInterceptor in WebSphere Application Server V6.

Configuration steps

In this section we will discuss the steps to configure Single Sign-On between WebSphere Application Server and WebSEAL.

Pre-requisites:

- ▶ Ensure that the IBM Tivoli Directory Server V5.2 is installed and configured for both Tivoli Access Manager for e-business and for WebSphere Application Server registry. If you are following the scenario introduced in this book, you can import the import.ldif provided as an additional material.
- ▶ Ensure that Global Security is enabled with LTPA and LDAP.
- ▶ We will use a WebSphere Application Server sample application called Technology Samples. Ensure that you install this application and are able to run it through the Web server.
- ▶ Tivoli Access Manager for e-business V5.1 Fixpack 6 should be installed and configured correctly. You should be able to access the WebSEAL form login page. To configure WebSEAL for form-based authentication do the following changes in webseald.conf file:

```
forms-auth=https  
basic-auth=none
```

The following diagram shows a simple environment we used for this chapter to test the configuration we described here.

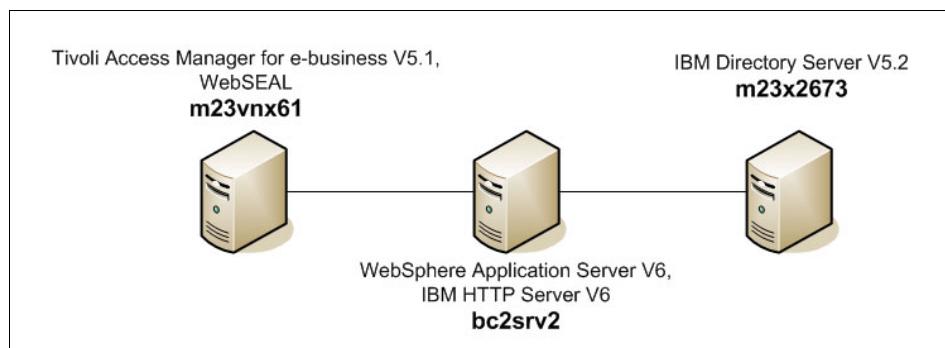


Figure 10-12 Test environment for the configuration

Installing and configuring the base products

To begin, install and configure WebSphere Application Server, IBM Directory Server with DB2, IBM HTTP Server, Tivoli Access Manager for e-business (including WebSEAL). Refer to the WebSphere Application Server product documentation, the Tivoli product documentation for installation and

configuration. For user registry configuration for WebSphere Application Server, you can refer to the Chapter 2., “Configuring the user registry” on page 7.

Creating test users for Tivoli Access Manager

Create two user accounts in LDAP by importing the following text as an .ldif file.

Example 10-10 tai-config.ldif

```
dn: uid=taiuser,ou=users,o=itso,c=us
objectclass: organizationalPerson
objectclass: ePerson
objectclass: top
objectclass: inetOrgPerson
uid: taiuser
userpassword: taiuser1
sn: taiuser
givenname: taiuser
cn: tai
preferredlanguage: en

dn: uid=amy,ou=users,o=itso,c=us
objectclass: organizationalPerson
objectclass: ePerson
objectclass: top
objectclass: inetOrgPerson
uid: amy
userpassword: test
sn: amy
givenname: amy
cn: amy
preferredlanguage: en

dn: uid=john,ou=users,o=itso,c=us
objectclass: organizationalPerson
objectclass: ePerson
objectclass: top
objectclass: inetOrgPerson
uid: john
userpassword: test
sn: john
givenname: john
cn: john
preferredlanguage: en

dn: cn=managers,ou=groups,o=itso,c=us
objectclass: top
objectclass: groupOfUniqueNames
cn: managers
```

```
uniqueMember: uid=amy,ou=users,o=itso,c=us

dn: cn=human_resources,ou=groups,o=itso,c=us
objectclass: top
objectclass: groupOfUniqueNames
cn: hr
uniqueMember: uid=john,ou=users,o=itso,c=us
```

Use the following command to import the file: **ldif2db -i tai-config.ldif**

This will create three users in the directory: taiuser, amy, john.

Start the pdadmin tool and import the users into Tivoli Access Manager using the following commands:

```
user import taiuser uid=taiuser,ou=users,o=itso,c=us
user modify taiuser account-valid yes
user modify taiuser password-valid yes
```

Similarly run all three commands replacing taiuser with amy and john

Also import the managers group:

```
group import managers cn=managers,ou=groups,o=itso,c=us
```

Configuring SSL for the Web server

If you are setting up WebSEAL junction to use SSL, which we recommend, you must perform this step so that the HTTPS traffic uses a self-signed certificate. If you are using TCP, instead of SSL for your WebSEAL junction, skip this configuration step and proceed to the next step. The Web server must have a port defined for SSL (usually 443). You can use the IBM Key Management Utility, ikeyman, to generate a self-signed certificate.

1. Set up the IBM HTTP Server using SSL.
2. Use the keystore name: ihskeys.kdb
3. Extract the self-signed certificate into a file named IHSCertificate.arm.
4. Restart the HTTP Server.
5. Verify the configuration by accessing a Web page using SSL (HTTPS).

Importing the Web server certificate to WebSEAL to establish trust

You need to load the certificate you created in the previous step into the key database of WebSEAL.

1. If your HTTP Server is on a different machine than WebSEAL, copy your certificate from the previous step *IHSCertificate.arm* from the HTTP Server machine to the WebSEAL machine.
2. Start ikeyman on the WebSEAL machine and open WebSEAL's keystore located at
`<Access_Manager_Install_root>|PDWeb|www-default|certs|pdsrv.kdb.`
This is the keyring used by WebSEAL to store acceptable CA certificates for SSL junctions. The password for this keystore is *pdsrv*.
3. Add the IBM HTTP Server certificate *IHSCertificate.arm* to the WebSEAL keystore.

Adding the SSL port to the virtual hosts in WebSphere Application Server

You need to configure the Web server plug-in to forward the HTTPS traffic to the application server. You need to update the virtual host list for WebSphere Application Server to include the correct hostname and port numbers, then regenerate the plug-in configuration.

1. Launch the Administrative Console and login on the WebSphere Application Server machine.
2. Select **Environment** → **Virtual Hosts** → **default_host** → **Host Aliases** → **New**. Add a host alias for the hostname and an SSL port. The hostname may be a single * (asterisk) or a fully-qualified hostname. Usually this would be the hostname of the Web server. The port number for SSL is usually 443.
3. Click **OK**.
4. Save the configuration for WebSphere, then regenerate the plug-in configuration.

Configuring WebSEAL

We will configure a WebSEAL junction from the WebSEAL Server to the Web server. The step will be performed on the WebSEAL machine.

1. On the WebSEAL machine, use the **pdadmin** command line to create a WebSEAL junction. Enter the following command:

```
server task default-webSEAL-<hostname> create -t ssl -h
<webserver_host> -p <SSL_port> -j -b supply -c all -f /ssl1
```

For TCP junctions use **tcp** instead of **ssl**.

In our scenario we ran the command as follows:

```
pdadmin sec_master> server task default-webseald-m23vnx61 create -t ssl -h
bc2srv2 -p 80 -j -b supply -c all -f /wastcp
```

2. Edit the webseald.conf file to configure the dummy password that will be passed in the HTTP Header and for Forms authentication. Open the file at <Access_Manager_install_root>/PDWeb/etc/webseald-default.conf.
3. In the [junction] stanza, change the basic-auth-dummy-password to the user password of the taiuser as shown:

```
basicauth-dummy-passwd = taiuser1
```
4. In the [forms] stanza, enable WebSEAL authentication using forms. If you would like to use only SSL junction then set the forms-auth to https.

```
forms-auth = https
```
5. Because you are using forms-based authentication and not basic authentication, change the ba-auth from https to none:

```
ba-auth = none
```
6. Restart the WebSEAL server, Policy Server and the Authorization Server.

Running the SvrSslCfg utility

Run the **SvrSslCfg** command to configure an SSL connection between Tivoli Access Manager and WebSphere Application Server. This command will create a configuration file called <WebSphere_root>/java/jre.PdPerm.properties and a Java key store file, which securely stores a client certificate. These two files enable WebSphere Application Server to be able to contact the Tivoli Access Manager server.

Note: This command needs to be run for each WebSphere Application Server machine, Java environment. For example, in a network deployment setup, if the deployment manager and a node are installed on the same machine, you must run the SvrSslCfg twice. Once for the deployment manager developer kit and again to configure the node developer kit.

1. Run the setupCmdLine.bat (or ./setupCmdLine.sh) script, located in <WebSphere_root>\bin to set up the environment.
2. Set the WAS_HOME environment variable to reflect the WebSphere Application Server installation root.
3. Run the SvrSslCfg utility as one continuous command line, or enter keywords on several lines using a trailing continuation character (\) as shown:

```
CLASSPATH=${WAS_HOME}/java/jre/lib/ext/PD.jar:${WAS_CLASSPATH}
java \
-cp ${CLASSPATH} \
com.tivoli.pd.jcfg.SvrSslCfg \
-action config \
-admin_id sec_master \
```

```
-admin_pwd password \
-appsvr_id wasuser \
-policysvr tam_policy_server_host:7135:1 \
-authzsvr tam_authorization_server_host:7136:1 \
-mode remote \
-cfg_file configuration_file \
-key_file key_file \
-cfg_action create
```

The explanation of the different switches follows:

- ▶ action - Action to be taken, it can be config or unconfig.
- ▶ admin_id - administrator ID for Tivoli Access Manager, use: *sec_master*
- ▶ admin_pwd - password for the Tivoli Access Manager administrator
- ▶ appsvr_id - The name that is specified here is combined with the host name to create unique names for Tivoli Access Manager objects created for your application. The following names are reserved for Tivoli Access Manager applications: *ivaclid*, *secmgrd*, *ivnet*, and *ivweb*. This is an ID that is created in the registry. This ID is used by WebSphere Application Server to communicate with Tivoli Access Manager.
- ▶ appsvr_pwd - The password for the application server ID (appsvr_id).
- ▶ authzsvr - Access to the authorization server in the format of:
authorization_server_name:port_number:rank
- ▶ policysvr - Access to the policy server in the format of:
policy_server_host_name:port_number:rank
- ▶ cfg_action - *create* specifies to create the configuration and key store files during server configuration. Configuration fails if either of these files already exists.

replace specifies to replace the configuration and key store files during server configuration. Configuration deletes any existing files and replaces them with new ones.

- ▶ cfg_file - Specifies the fully-qualified file name.
- ▶ key_file - Specifies the directory that will contain the key files for the server. Make sure that server user (for example, *ivmgr*) or all users have permission to access the .kdb file
- ▶ host - the hostname for the application server
- ▶ mode - Specifies the mode in which the application operates. This value must be either *local* or *remote*.

The next command is an example for the scenario introduced in this book.

```
<wasroot>\java\jre\bin\java com.tivoli.pd.jcfg.SvrSslCfg -action config  
-admin_id sec_master -admin_pwd password -appsvr_id wsserver1 -appsvr_pwd  
password -port 8888 -mode remote -host m23vnx61 -pollicysvr m23vnx61:7135:1  
-authzsvr m23vnx61:7136:1 -cfg_file <was_root>\java\jre\PdPerm.properties  
-key_file <was_root>\java\jre\lib\security\PdPerm.ks -cfg_action create
```

Configuring TAI in WebSphere Application Server

The next step is to configure the TAI in WebSphere Application Server.

1. Launch the Administrative Console for WebSphere and login.
2. Expand the **Security** item, click **Global security** then click **Authentication Mechanisms → LTPA**.
3. Click **Trust Association**.
4. Click **Interceptors**.
5. Click **com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus**.

Note: If you plan to use the old TAI, then follow the procedures outlined in the following article:

http://www-106.ibm.com/developerworks/websphere/techjournal/0406_singh/0406_singh.html

6. Click **Custom properties**.
7. Click **New** to add new properties.

Enter the following property name:

com.ibm.websphere.security.webseal.loginId. Enter the following property value: taiuser.

The taiuser must be the shortname of the user in LDAP. This user's password is configured in the webseald.conf file as the dummy password. This user must exist in LDAP in the directory tree that WebSphere application and TAM can search.

8. Click **OK** to continue.
9. Add the following properties to the TAI the same way as described above.
 - com.ibm.websphere.security.webseal.id = iv-user

This is not required, by default it is set to iv-creds, since we have created the junction with -c all we can add the iv-user as an additional check for WebSEAL validation.
 - com.ibm.websphere.security.webseal.hostnames - set the hostname to access for the WebSEAL machine.

- com.ibm.websphere.security.webseal.ports - set the port to access for the WebSEAL machine.
- com.ibm.websphere.security.webseal.checkViaHeader = true.
Default value is false.

When it is set to false, the webseal hostnames in the Via Header are ignored, and the hostnames and port properties do not need to be set. The only mandatory property when checkViaHeader is false is com.ibm.websphere.security.webseal.loginId.

When set to true, the TAI is going to validate the hostnames set in TAI with the ones in the ViaHeader set by WebSEAL.

- com.ibm.websphere.security.webseal.hostnames = m23vnx61
It is not required if CheckViaHeader is set to false or not set. This is a comma separated list of WebSEAL hostnames to be trusted and it is case sensitive. If using the Tivoli Access Manager plugin for Web servers, this property is not set.
- com.ibm.websphere.security.webseal.ports = 443,80
It is not required if CheckViaHeader is set to false or not set. This is a comma separated list of WebSEAL ports to be trusted. If using the Tivoli Access Manager plugin for Web servers, this property is not set.

10. Save the configuration for WebSphere, then restart the application server.

10.8.3 Configuration for the LTPA approach

LTPA refers to Lightweight Third Party Authentication which is an IBM proprietary technology used in IBM products like WebSphere Application Server, Tivoli Access Manager and Lotus Domino. The LTPA token is an encrypted string that contains a user ID, expiration time and a digital signature. Let's look at a scenario in which WebSphere issues the LTPA token. In this case WebSphere Application Server authenticates the user and issues an LTPA token. This LTPA token can be passed to another WebSphere Application Server instance which can read this LTPA token and determine the authenticated user ID. The basis for reading and trusting the LTPA token is that the two WebSphere Application Server instances share the same LTPA keys for token generation and they must also share the same user registry.

Note: The LTPA keys are automatically generated the first time security is enabled. After security is enabled, WebSphere Application Server can generate a new set of keys in two ways:

1. When the LTPA password is changed, a new set of keys are generated.
2. When the *Generate Keys* tab is clicked. The new set of keys are not used until they are saved.

However, in most real world scenarios, the authentication is done by a third party security server like Tivoli Access Manager which is also capable of issuing LTPA tokens. Both Tivoli Access Manager and WebSphere Application Server are configured with the same LTPA encryption key. WebSphere Application Server receives the LTPA Token, decrypts it and determines the authenticated user ID and does not challenge the user again thus providing Single Sign-On from Tivoli Access Manager to WebSphere Application Server. This scenario is shown in figure below.

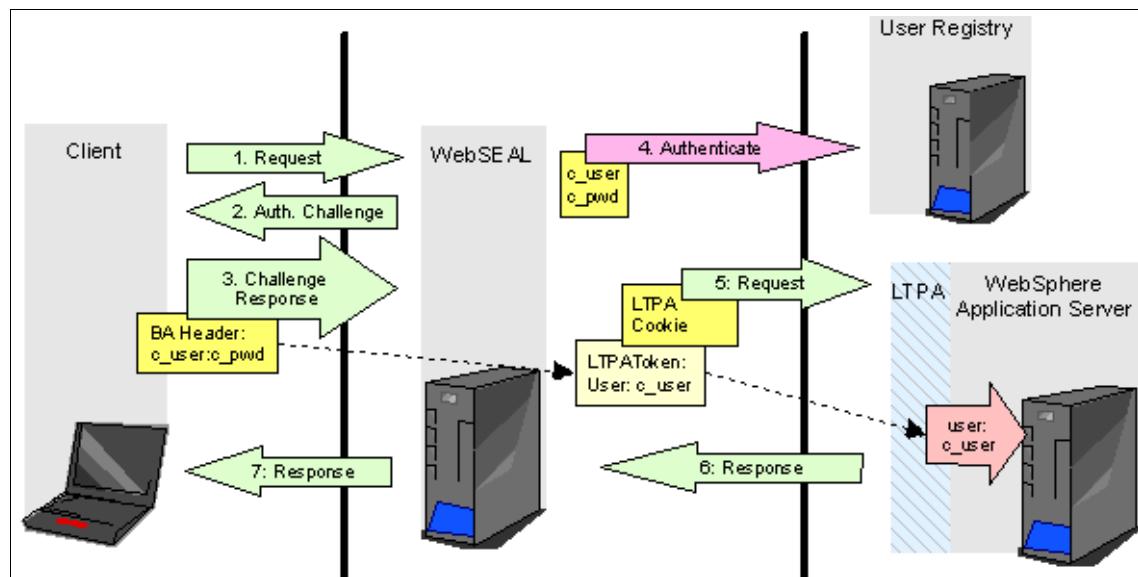


Figure 10-13 Request flow when using LTPA

The following steps describe the flow when using LTPA.

1. A client requests a secured resource.
2. WebSEAL is the Web proxy that intercepts the request and challenges the client.
3. Client supplies the credentials in a new request.

4. WebSEAL authenticates the user against the user registry and constructs an LTPA Token and attaches it to an LTPA cookie.
5. Request is passed to the backend junctioned Web server with the WebSphere Application Server plugin.
6. WebSphere Application Server receives the request. WebSphere Application Server looks for the LTPA token and finds it in the cookie. WebSphere Application Server decrypts the LTPA token and verifies that the signature is correct. From here on WebSphere Application Server trusts the identity of the user as specified in the LTPA Token.
7. WebSphere Application Server sends output to WebSEAL and WebSEAL sends the output to the client.

Note: WebSEAL does not send the LTPA cookie to the client, but rather the cookie is stored in WebSEAL's LTPA cache. WebSEAL uses a different cookie to identify the session with the client, then the actual LTPA token is mapped to the session.

This approach provides higher security since the LTPA token could be captured on the network.

Configuring LTPA

The following steps describe how to integrate WebSEAL and WebSphere Application Server to use LTPA.

1. Launch the Administrative Console for WebSphere Application Server and login.
2. Select **Global Security**, expand **Authentication mechanisms** and click **LTPA** to see the LTPA configuration panel.

Change the password if needed. The first time that security is enabled with LTPA authentication mechanism, LTPA keys are automatically generated with the password entered in the panel. In this procedure, however, LTPA keys will be generated manually so that they can be immediately exported and copied to the WebSEAL server. You cannot export the LTPA keys until the changed password is saved in the WebSphere Application Server repository. However its not mandatory to change the password to export the keys.

In the Key File name field, enter the full path of a file on the WebSphere Application Server machine where the key file should be placed.

Click **Export Keys** to create the exported key file.

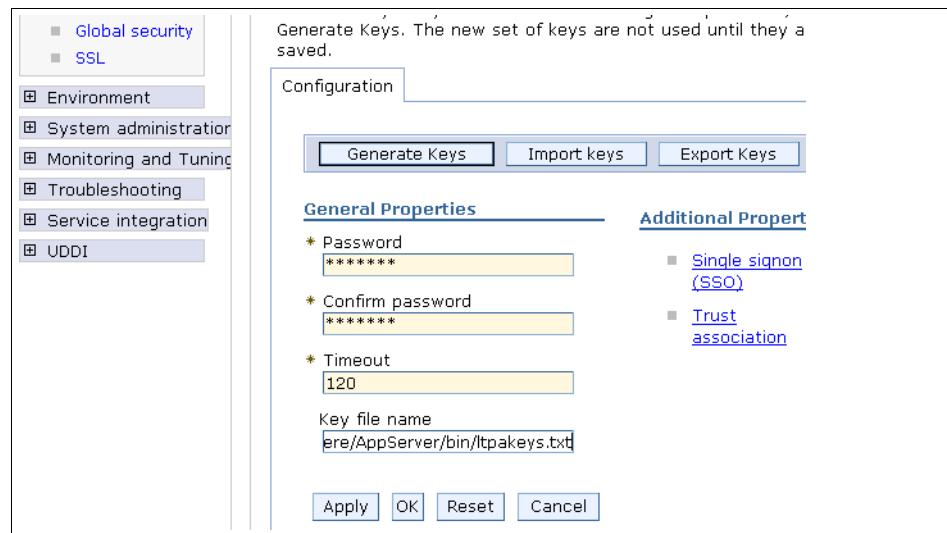


Figure 10-14 LTPA configuration

3. Save the configuration for WebSphere.
4. Click **Single signon (SSO)** and check the **Enabled** box. Also enter the SSO Domain name as shown below in the figure.



Figure 10-15 SSO Domain parameters

5. On the Global Security Panel, Ensure that the WebSphere Application Server is set up with the registry and Global Security is enabled. Change the Active Authentication Mechanism to **LTPA (Light weight Third Party Authentication)**.

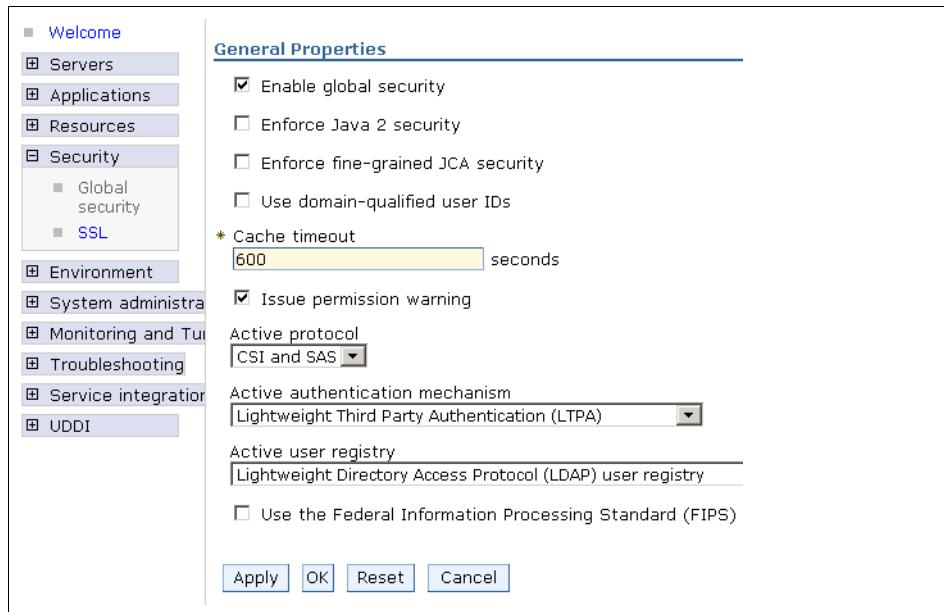


Figure 10-16 Global security settings

6. Click **Apply** to accept the changes then save the configuration for WebSphere.
7. Copy the LTPA key file to the WebSEAL server. Note that this file should be kept very secure, otherwise the LTPA trust relationship may be compromised.
8. In order to set up an SSL junction, you need to enable the Web server to use SSL and exchange certificates between the Web server and WebSEAL.
9. Next step is to create the junction on the WebSEAL server. For junction creation, it is required to specify three options:
 - A - Enables LTPA cookies.
 - F <full_path_to_ltpa_keys_file> - specifies the full path name and location (on the webseal host machine) of ltpa key file exported from the WAS machine. This shared ltpakeys.txt file was originally created on the WebSphere Application Server host and copied to the webseal machine.
 - Z <keyfile_password> - specifies the password required to open the keyfile for LTPA, it is defined in the WebSphere Application Server Administrative Console.

Using pdadmin on the WebSEAL server, execute the following commands:

```
server task default-webseald-m23vnx61 create -t SSL -A -F
“c:\ltpakeys\ltpakeys.txt” -Z “password” -h bc2srv2 -p 443 /ltpa
```

10. Test the junction by accessing the snoop servlet, in our example:
<https://m23vnx61/ltpa/snoop>.

Configuring LTPA cache for WebSEAL

Since LTPA creation, encryption, decryption introduces processing overhead, LTPA cache processing allows you to improve the performance of LTPA junctions in a high-load environment. The LTPA cache is enabled by default. To configure the ltpa cache settings open the *webseald.conf* file and locate the [ltpa-cache] stanza. The following settings are available:

- ▶ ltpa-cache-enabled - default value is “yes”. It enables and disables the LTPA cache.
- ▶ ltpa-cache-size - default value is “1096”. It defines the maximum number of entries allowed in the cache hashtable. Higher value sets more memory and results in faster information access.
- ▶ ltpa-cache-entry-lifetime - default is 3600 seconds. It is the lifetime of a cache entry.
- ▶ ltpa-cache-entry-idle-timeout - default value is 600 seconds. It defines the maximum time an inactive cache entry can remain in cache.

For more information about tuning these values, please refer to the WebSEAL administration guide found here:

http://publib.boulder.ibm.com/tividd/td/ITAME/SC32-1359-00/en_US/PDF/am51_webseal_guide.pdf

10.9 Integration of IBM WebSphere Application Server and Tivoli Access Manager

If we want to integrate WebSphere Application Server applications with Tivoli Access Manager we have to distinguish between:

- ▶ Integration of new applications that are to be developed or existing applications that will be changed.
- ▶ Integration of existing applications without any changes.

For Java applications Access Manager provides a pure Java version of the Authorization API (aznAPI) providing the following classes: PDPermission, PDPrincipal, and PDLoginModule. PDPermission is usable in both a Java Authentication and Authorization Services (JAAS) and non-JAAS environment. These methods can be used for securing new applications or to adjust existing applications. Often, there are already existing J2EE applications secured by WebSphere declarative security also using J2EE security methods alternatively.

When the embedded Tivoli Access Manager is enabled in WebSphere Application Server, it imports WebSphere security definitions into the Access Manager's object space. The function that determines whether a user is granted any permitted roles is then handled by Tivoli Access Manager.

10.9.1 aznAPI

aznAPI is an API specifically designed for Access Manager. It has been approved by the OpenGroup as the standard implementation of the Authorization Model. Access Manager provides a C and a Java version of the API. The aznAPI Java classes are basically Java wrappers for the original C API. WebSphere applications may use the aznAPI to retrieve fine-grained authorization information about a user. The authorization API consists of a set of classes and methods that provide Java applications with the ability to interact with Access Manager to make authentication and authorization decisions.

The aznAPI classes are installed as part of the Tivoli Access Manager Java runtime component which comes with WebSphere Application Server V6. These classes communicate directly with the Tivoli Access Manager authorization server by establishing an authenticated SSL session with the authorization server process.

Note: For more detailed information about Java development with Tivoli Access Manager security and administration, refer to the following documents:

- ▶ *IBM WebSphere Application Server Integration Guide, IBM Tivoli Access Manager V5.1* product guide, SC32-1368.
- ▶ *Authorization Java Classes Developer Reference, IBM Tivoli Access Manager V5.1* product guide, SC32-1350.
- ▶ *Administration Java Classes Developer Reference, IBM Tivoli Access Manager V5.1* product guide, SC32-1356.

10.9.2 Tivoli Access Manager and J2EE Security

The Java security is policy based. This means that authorization to perform an action is not hard coded into the Java run time or executables. Instead, the Java environment consults policy external to the code to make security decisions, in the simplest case, this policy is implemented in a flat file, which somewhat limits its scalability and also adds administrative overhead.

To overcome the flat file implementation of Java 2 policy, and to converge to a single security model, the authorization framework provided by Access Manager can be leveraged from inside a normal Java security check. As mentioned

earlier, the most natural and architecturally pleasing implementation of this support is the JAAS framework.

Support for this standard provides the flexibility for Java developers to leverage fine-grained usage of security and authorization services as an integral component of their application and platform software. The Tivoli Access Manager provided *PLoginModule* login module is enabled when the embedded Tivoli Access Manager is enabled in WebSphere V6. With the Java 2 and JAAS support delivered with the embedded Tivoli Access Manager, Java applications can:

- ▶ Use the Tivoli Access Manager to acquire authentication and authorization credentials from Access Manager.
- ▶ Use the PDPermission class to request authorization decisions.

This offers Java application developers the following advantages:

- ▶ The security of Java applications that use PDPermission is managed using the same, consistent model as the rest of the enterprise.
- ▶ Java developers do not need to learn anything beyond Java 2 and JAAS.
- ▶ Updates to security policy involve Tivoli Access Manager-based administrator actions, rather than any code updates.

Today, JSPs, servlets, and EJBs can take direct advantage of these services. When WebSphere containers support Java 2 security, EJB developers can avoid the need to make security calls by having the containers handle security while they focus on business logic.

There are two options for implementing fine-grained authorization (at the level of actions on objects) within servlets and EJBs today:

- ▶ Given the Access Manager credential information (EPAC) passed in the HTTP header, the servlet or the EJB would have to use the PDPermission class extensions directly to query Access Manager for access decisions. The access enforcement is still the responsibility of the application (servlet or EJB).
- ▶ Develop a proxy bean (a session bean) within an EJB. This proxy bean will intercept all method invocations and communicate with Access Manager (using the PDPermission class) to obtain the access decision and enforce it.

10.9.3 Embedded Tivoli Access Manager in WebSphere Application Server V6

If the application is designed as a J2EE application, it would rely on the J2EE security methods to get a user ID and role. Tivoli Access Manager for WebSphere Application Server provides container-based authorization and centralized policy management for WebSphere Application Server Version 5.1. Tivoli Access Manager for WebSphere Application Server is implemented as an Access Manager aznAPI application running on the WebSphere Application Server instance.

Access Manager for WebSphere Application Server supports applications that use the J2EE Security Classes without requiring any coding or deployment changes to the applications. Tivoli Access Manager for WebSphere Application Server is used to evaluate access requests from a user to protected resources based on the following tasks:

- ▶ Authentication of the user.
- ▶ Determination of whether the user has been granted the required role by examining the WebSphere deployment descriptor.
- ▶ The WebSphere container using Tivoli Access Manager to perform role membership checks for security code added directly into an application (programmatic security).

Enabling the Embedded Tivoli Access Manager

To configure the WebSphere to use the Tivoli Access Manager APIs and the Access Manager JACC implementation you have to enable the embedded Tivoli Access Manager in WebSphere Application Server V6.

Note that for this scenario, Tivoli Access Manager and WebSphere Application Server has to share the same user registry.

1. Create a Tivoli Access Manager user, for example: wstam. Enter the following command in pdadmin:

```
pdadmin sec_master> user create wstam -no-password-policy  
cn=wstam,ou=users,o=itso,c=us wstam wstam test
```

```
user modify wstam account-valid yes
```

```
user modify wstam description "Access Manager user ID for WebSphere"
```

2. Enable Tivoli Access Manager JACC provider, please refer to the 11.7, “Integrating Tivoli Access Manager as an external JACC provider” on page 284. Make sure Global Security is enabled, save your settings and restart WebSphere. After enabling Tivoli Access Manager, you should be able

to see the following see three new servers in the pdadmin utility, in our example:

```
pdadmin sec_master> server list  
Authn_418532961-bc2srv1  
Authz_293977456-bc2srv1  
JACC_293977456-bc2srv1
```

Migration of roles and principals to groups

When the embedded Tivoli Access Manager included in WebSphere Application Server is installed only the Administration Console roles and resources are migrated to Tivoli Access Manager.

In WebSphere Application Server V5 there was a migration utility to migrate the application roles to the Tivoli Access Manager object space. In WebSphere V6 if JACC is enabled, every time a new application is deployed all the required objects and mappings will be created. To create the resources required for the applications already installed in your application server, you must reinstall the applications. WebSphere will create the required objects in the Tivoli Access Manager object space during deployment, you will also have the chance to assign Application Roles to Tivoli Access Manager users or groups.

Deployment descriptor mapping in Tivoli Access Manager

WebSphere maps the application deployment descriptor in a somewhat peculiar way, that might not be obvious at first look. To show how this work we will use the some of the objects created after installing the Itsohello application. The following diagram is a representation of a part of the Tivoli Access Manager object space.

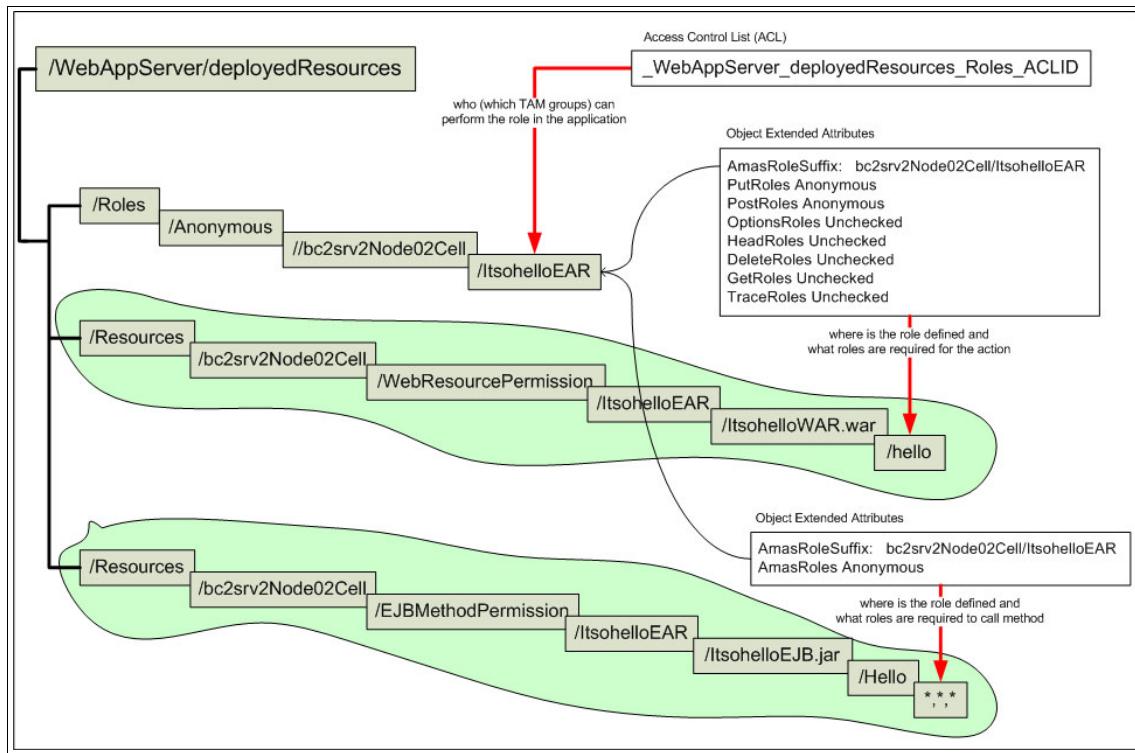


Figure 10-17 Deployment descriptor and J2EE roles in TAM object space.

When the application is deployed all the application roles will be created under `/webSppServer/deployedResources/Role`, the mapping is of the form `/<role_name>/<server_node_cell>/<application_name>`. Although there are some extra objects under that, the Access Manager ACL that govern the role, in other words the Tivoli Access Manager users and groups assigned to the role, is attached at the `<application_name>` level. In our example the ACL attached to `/WebAppServer/deployedResources/Roles/Anonymous/bc2srv2Node02Ce11/ItsohelloEAR` will change to reflect the Tivoli Access Manager users and groups mapped to the role in the Administration Console. Also the Administration console will read the mappings by looking at the ACLs attached to that object.

Note: If you update the ACLs using the tivoli tools, the WebSphere server will need to be restarted to re-read the ACLs attached to the J2EE roles.

During deployment WebSphere translates the each Web resource in the `web.xml` descriptor to an object under `/webSppServer/deployedResources/Resources`, the mapping is of the form:

```
<server_node_cell>/<application_name>/<??>/<application_name>/<war_filename>/<resource>
```

In the sample, the Itsohello application the information from the web.xml file. The relevant information in the web.xml file, in Figure 10-18, translates to the object:

```
/WebAppServer/deployedResources/Resources/bc2srv2Node02Cell/WebResourcePermission/ItsohelloEAR/ItsohelloWAR.war/hello
```

The Web deployment descriptor lists each action allowed in the Web objects according to the HTTP Action, in our case the descriptor allows the Anonymous role *PUT* and *POST* to the /hello Web resource and in Tivoli Access Manager the object is created with extended attributes to represent that as described in Figure 10-17 on page 266.

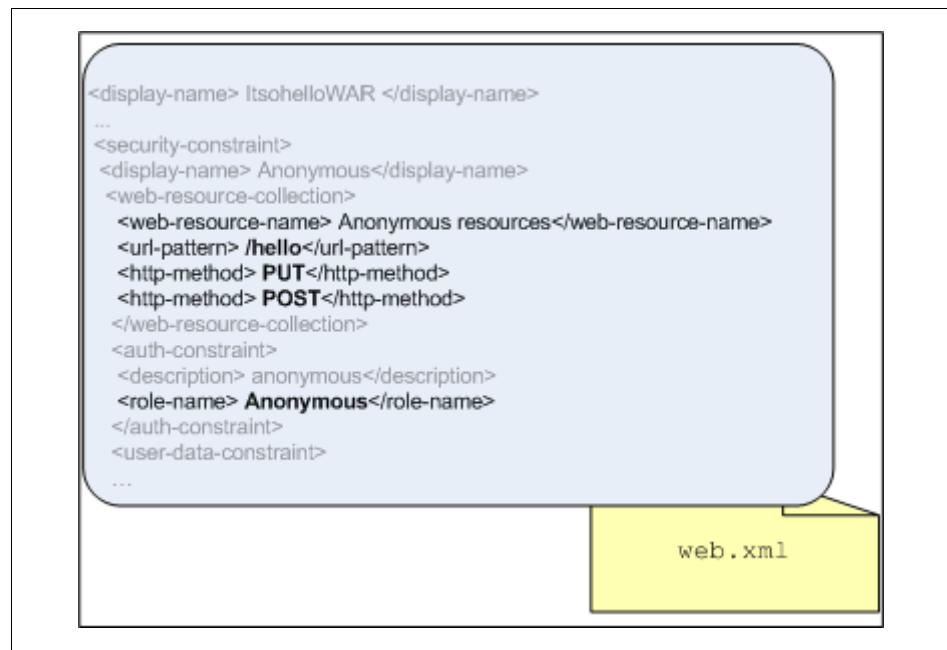


Figure 10-18 web.xml deployment descriptor

Also for the EJB methods, WebSphere translates each method signature in the EJB as described in the ejb-jar.xml descriptor to an object under /webSppServer/deployedResources/Resources, the mapping is of the form:

```
<server_node_cell>/<EJBMETHODPERMISSION>/<application_name>/<ejb_filename>/<ejb_name>/<bean_name>/<method_signature>
```

In the sample, the Itsohello application the information from the ejb-jar.xml file. The relevant information in the ejb-jar.xml file the can be seen in Figure 10-18 on page 267. As you can see in Figure 10-17 on page 266 translates to:

```
/WebAppServer/deployedResources/Resources/bc2srv2Node02Cell/EJBMethodPermission  
/ItsohelloEAR/ItsohelloEJB.jar>Hello/*,*,*
```

The EJB deployment descriptor lists the roles allowed to invoke the methods in the bean, in our case the descriptor allows the Anonymous role to invoke any method in the *Hello* bean, this is both shown in the descriptor in Figure 10-19 and the Tivoli Access Manager object space in Figure 10-17 on page 266.

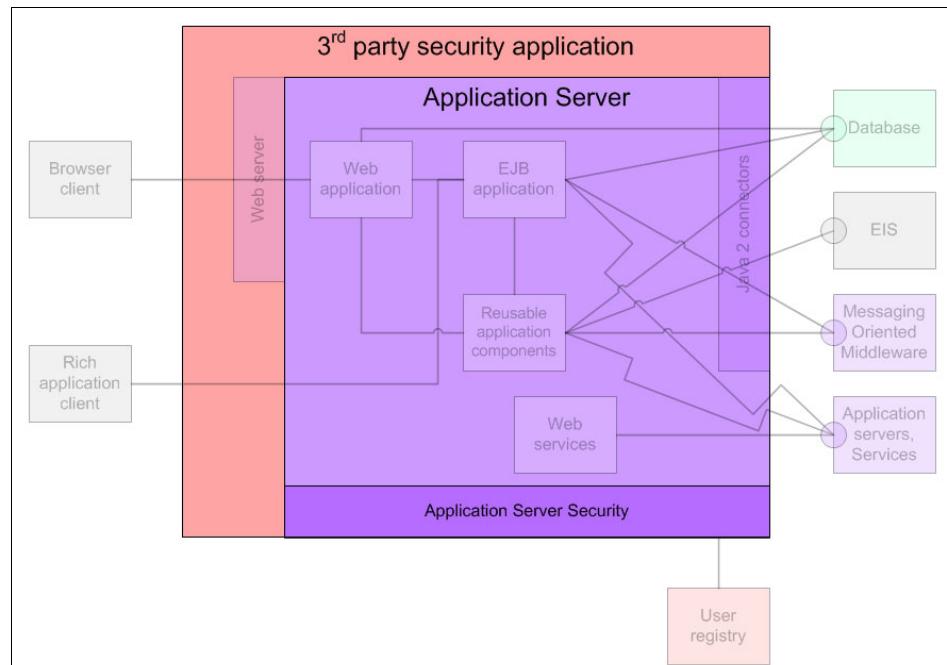


Figure 10-19 ejb-jar deployment descriptor



Chapter 11

Externalizing authorization with JACC



11.1 The specification

The Java Authorization Contract for Containers (JACC) is a new specification introduced in J2EE 1.4 through the JSR 115 process. This specification defines a contract between J2EE containers and authorization providers. This enables any third-party authorization providers to plug into any J2EE 1.4 application servers such as WebSphere to make authorization decisions when a J2EE resource is being accessed. The access decisions will be made through the standard `java.security.Policy` object.

You can find more information about JACC under JSR 115 at:

<http://www.jcp.org/en/jsr/detail?id=115>

The specification defines new `java.security` Permission classes to satisfy the J2EE authorization model. The specification defines the binding of container access decisions to operations on instances of these permission classes. The specification defines the semantics of policy providers that employ the new permission classes to address the authorization requirements of J2EE, including the following:

- ▶ The definition of roles as named collections of permissions
- ▶ The granting to principals of permissions corresponding to roles
- ▶ The determination of whether a principal has been granted the permissions of a role (for example, `isCallerInRole`)
- ▶ The definition of an identifier to role mappings that bind application-embedded identifiers to application scoped role names

The specification defines the installation and configuration of authorization providers for use by containers. The specification defines the interfaces that a provider must make available to allow container deployment tools to create and manage permission collections corresponding to roles.

There are three primary components that the JACC Specification defines:

- ▶ Deployment tools contract
- ▶ Container contract
- ▶ Provider contract

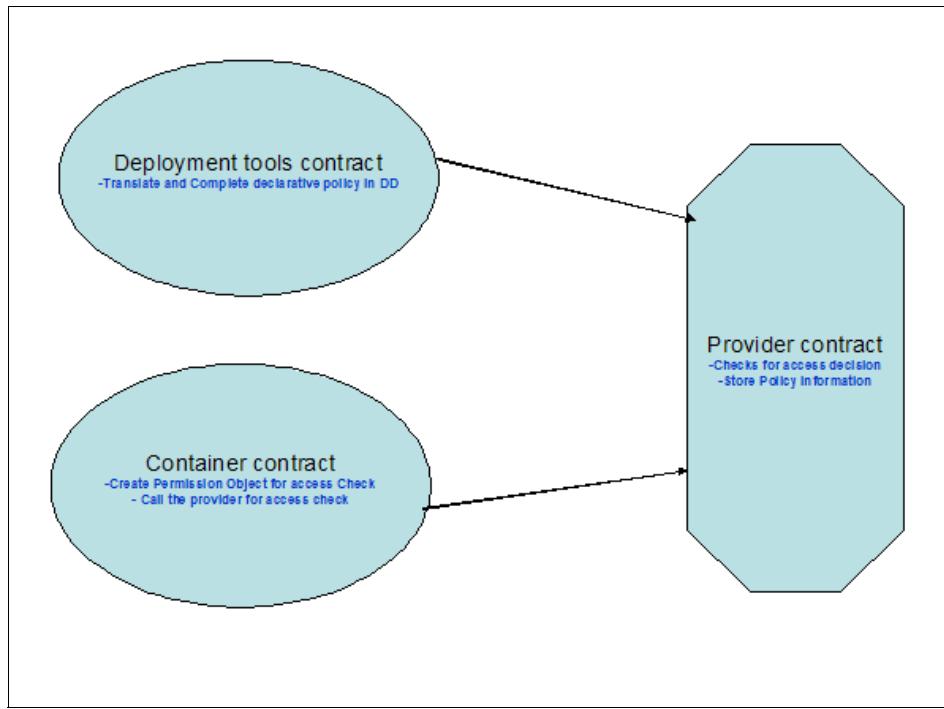


Figure 11-1 Security contracts in JACC

11.2 Deployment tools contract

J2EE deployment tools must translate and complete the declarative policy statements appearing in deployment descriptors into a form suitable for securing applications on the platform. The resulting policy statements may differ in form from the policy statements appearing in the deployment descriptors. The specification requires that the policy information in the deployment descriptor be propagated to the container during the application install time. The policy information contains the security-related information in the deployment descriptor. Specifically, the security_constraint information in the web.xml and the method_permission information in the ejb-jar.xml along with security_role_ref information in both these files need to be propagated to the provider in the format specified by the contract. The format is different for Web and EJB modules and is governed by the rules specified in the contract. The deployment tools contract defines the following key components:

► **Policy contexts and policy context identifiers**

The JACC Specification states that deployment tools of contract must define the separate authorization policy contexts corresponding to each deployed instance of a J2EE module. Deployment tools of contract must provide the per module scoping of policy context necessary to provide for the independent administration of policy contexts corresponding to individual application modules (perhaps deployed multiple times) within a common policy provider.

► **Servlet policy context identifiers**

The JACC Specification states that deployment tools of contract must define servlet policy context identifiers sufficient to differentiate all instances of a Web application deployed on the logical host or on any other logical host that may share the same policy statement repository. One way to satisfy this requirement is to compose policy context identifiers by concatenating the hostname with the context path (as defined in the Servlet specification) identifying the Web application at the host.

► **Translating servlet deployment descriptors**

The JACC Specification states that deployment tools of contract must translate the security-constraint and securityrole-ref elements in the deployment descriptor into permissions and add them to the PolicyConfiguration object.

► **EJB policy context identifiers**

The JACC Specification states that the EJB policy context identifiers are sufficient to differentiate all instances of the deployed EJB JARs on every application server.

► **Translating EJB deployment descriptors**

If the method-permission element contains the unchecked element, then the deployment tools must call the addToUncheckedPolicy method to add the permissions resulting from the translation to the PolicyConfiguration object. Alternatively, if the method-permission element contains one or more role-name elements, then the deployment tools must call the addToRole method to add the permissions resulting from the translation to the corresponding roles of the PolicyConfiguration object.

► **Deploying an application or module**

The application server's deployment tools must translate the declarative authorization policy appearing in the application or module deployment descriptor(s) into policy statements within the Policy providers used by the containers to which the components of the application or module are being deployed.

► **Undeploying an application or module**

To ensure that there is not a period during undeployment when the removal of policy statements on application components renders what were protected components unprotected, the application server must stop dispatching requests for the application's components before undeploying an application or module.

► **Deploying to an existing policy configuration**

To associate an application or module with an existing set of linked policy contexts, the identifiers of the existing policy contexts must be applied by the relevant containers in fulfilling their obligations as defined in the Policy Decision and Enforcement Subcontract. The policy contexts should be verified for existence, by calling the `inService` method of the `PolicyConfigurationFactory` of the relevant containers' Policy providers. The deployment tools must call `Policy.refresh` on the Policy provider of each of the relevant containers, and the containers must not perform pre-dispatch decisions or dispatch requests for the deployed resources until these calls have completed.

► **Redeploying a module**

Containers are not required to implement redeployment functionality.

11.3 Container contract

The container contract of the JACC Specification specifies how the container creates the permission objects during access checks and calls the provider with appropriate information to help make the access decision. When a resource is being accessed, the container is expected to create the appropriate permission object and call the provider's `Policy.implies` method. The container is also expected to register what are called the policy context handler objects that contain additional information to make the access decision. The following handlers are required to be registered by the containers. The container contract defines the following components:

- Policy Enforcement by Servlet Containers which includes the Evaluation of Transport Guarantees, Pre-dispatch Decision and Application Embedded Privilege Test.
- Provider Support for Servlet Policy Enforcement which includes Servlet Policy Decision Semantics, Matching Qualified URL Pattern Names, WebResourcePermission Matching Rules, WebRoleRefPermission Matching Rules and WebUserDataPermission Matching Rules.
- Policy Enforcement by EJB Containers which includes the EJB Pre-dispatch Decision and EJB Application Embedded Privilege Test.

- ▶ Provider Support for EJB Policy Enforcement which includes EJB Policy Decision Semantics, EJBMethodPermission Matching Rules and EJBRoleRefPermission Matching Rules.
- ▶ Component runAs Identity
- ▶ Setting the Policy Context
- ▶ Checking AccessControlContext Independent Grants
- ▶ Checking the Caller for a Permission
- ▶ Missing Policy Contexts
- ▶ Default Policy Context
- ▶ Policy Compatibility Requirements
- ▶ Optimization of Permission Evaluations

11.4 Provider contract

The provider contract in the JACC Specification specifies that each JRE of an application server should be provided with classes that implement the PolicyConfigurationFactory class and the PolicyConfiguration interface. The classes are used by the container to propagate the security information to the provider. The provider is also expected to provide the implementation for the java.security.Policy object. This Policy object must assume responsibility for performing all access decisions within the JRE in which it is installed. The Policy object can delegate the non-javax.security.jacc access decisions to the corresponding default system Policy implementation class. The Provider contract defines the following components:

- ▶ Policy Implementation Class
- ▶ Policy Configuration Interface
- ▶ PolicyContext Class and Context Handlers
- ▶ What a Provider Must Do
- ▶ Optional Provider Support for JAAS Policy Object
- ▶ What the Application Server Must Do

11.5 Why JACC?

In the J2EE 1.3 Specification, there is no specification address; the access decisions that are made by the Application Server vendor implementations and proprietary interfaces are used for third party vendor product integration. There is no standard way for third party authorization providers such as Tivoli Access Manager to plug in to application servers to make the decisions. There is no

standard way for the third party providers to collect the security policy information from the application or from the application servers. In order to address these issues, the JAAC is introduced in the J2EE 1.4 Specification.

11.6 JACC in WebSphere V6

WebSphere Application Server 6.0 supports JACC and provides several key components to support the provider contract, container contract and deployment tool contract. The JACC Specification only specifies a contract to propagate the policy information to the provider. There is no contract specified to propagate the authorization table information to the provider. It is the responsibility of the provider to present some kind of management interface to handle principals (users/groups) to roles. In order to propagate the authorization table information, WebSphere Application Server provides interfaces RoleConfigurationFactory and RoleConfiguration. The implementation of these interfaces is optional.

Figure 11-2 shows the WebSphere support for the Deployment tools Contract, Provider Contract and Container Contract as specified by the JACC Specification.

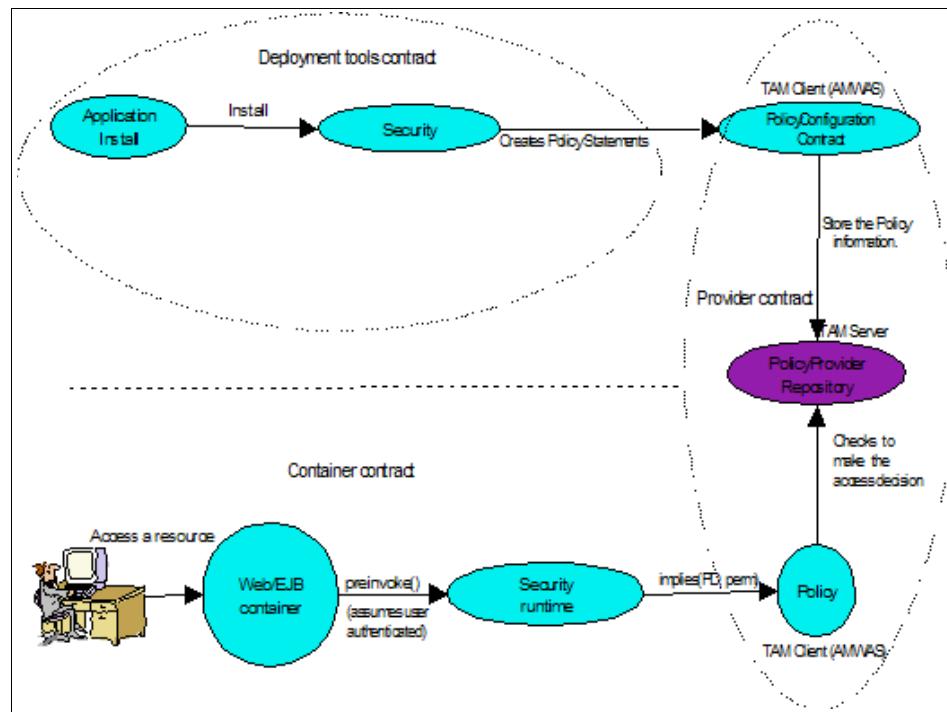


Figure 11-2 JACC support in WebSphere V6

Deployment tools contract

The following is the sequence of steps to take for deployment tools contract components:

1. Create a PolicyContext identifier (contextID) for the module.
2. Get the PolicyConfiguration for the contextID.
3. Translate the declarative policy in DD into appropriate permission classes.
4. Create Policy Statements in the PolicyConnfiguration objects using the permission classes.
5. Commit the changes and refresh the Policy.

Container contract

The following is the sequence of steps to take for deployment tools contract components:

1. Create the PolicyContext identifier for the module.
2. Register the various PolicyContextHandlers.
3. Create the Protection Domain (PD) and the appropriate Permission object (perm).

Provider contract

The provider makes the access decision based on the permission object.

11.6.1 JACC access decisions in WebSphere V6

The authenticated user makes a request to the Web or the EJB resource; the security runtime makes the decision whether to allow the access or not. This is called an access decision. The following figure shows the generic flow of the access decision for protected resources under the WebSphere environment, where external authorization is enabled through JACC.

Based on JACC, the appropriate permission object is created, the appropriate policy context handlers are registered, and the appropriate policy context identifier (contextID) is set. A call is made to the `java.security.Policy` object method implemented by the provider to make the access decision.

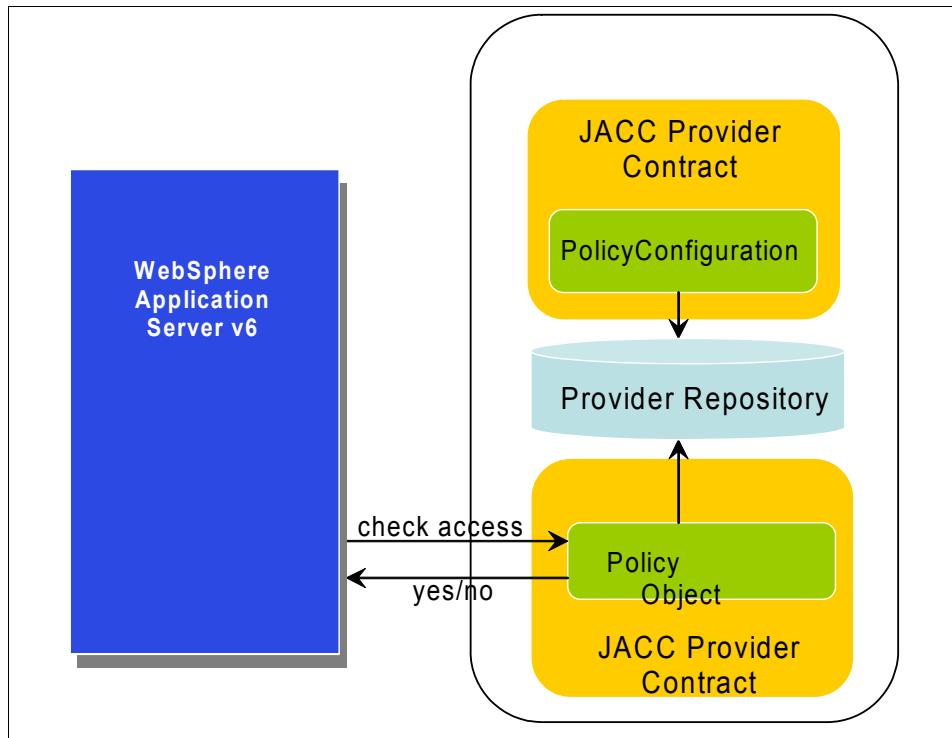


Figure 11-3 Externalized decisions making with JACC in WebSphere V6

Access decisions for enterprise beans

The authenticated user makes a request to the protected EJB resource; WebSphere security runtime delegates the authorization check to the security runtime. Figure 11-4 shows the flow steps that take place when the JACC is enabled for external authorization.

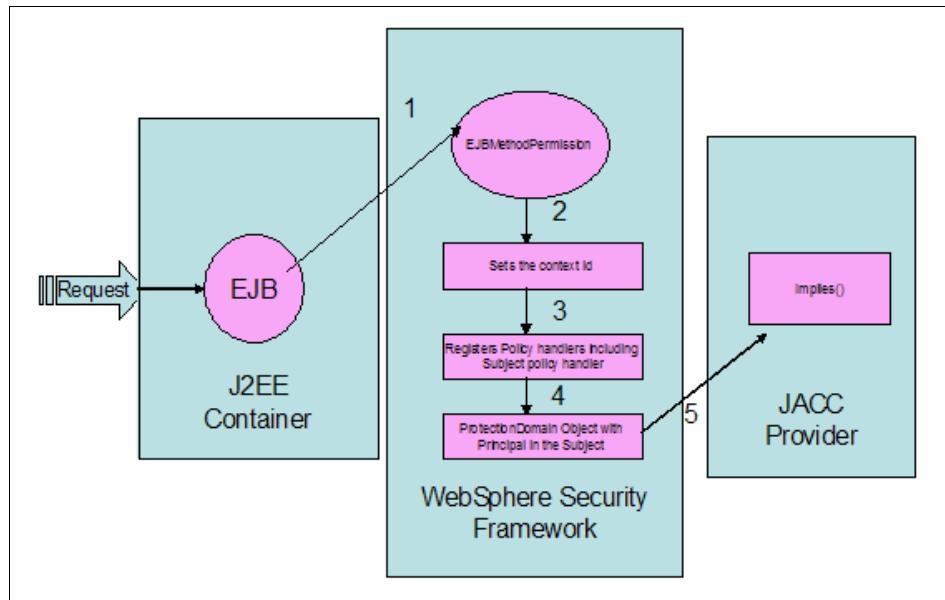


Figure 11-4 Logical steps for decision making

1. It creates the EJBMethodPermission object using the bean name, method name, interface name and the method signature.
2. It creates the contextID and sets it on the thread by using the PolicyContext.setContextID(contextID) method.
3. It registers the required policy context handlers, including the Subject policy context handler.
4. It creates the ProtectionDomain object with principal in the subject. If there is no principal, null is passed for the principal name.
5. The access decision is delegated to the JACC provider by calling the implies() method of the Policy object, which is implemented by the provider. The EJBMethodPermission and the ProtectionDomain objects are passed to this method.
6. The isCallerInRole() access check also follows the same process, except that an EJBRoleRefPermission object is created instead of an EJBMethodPermission.

Access decisions for Web resources

The authenticated user makes a request to the protected Web resource; the WebSphere security runtime delegates the authorization check to security

runtime. Following are the steps that take place when the JACC is enabled for external authorization.

Flow for Everyone subject

1. The WebResourcePermission is constructed with urlPattern and the HTTP method accessed.
2. A ProtectionDomain with a null principal name is created.
3. The JACC provider's Policy.implies() method is called with the permission and the protection domain. If the URI access is unchecked (or given access to the Everyone subject), the provider should permit access (return true) in the implies() method. Access is then granted without further checks.

Using HTTPS protocol

1. The WebUserDataPermission is constructed with the urlPattern accessed, along with the HTTP method invoked and the transport type of the request. If the request is over HTTPS, the transport type is set to CONFIDENTIAL; otherwise, null is passed.
2. ProtectionDomain with a null principal name is created.
3. The JACC provider's Policy.implies() method is called with the permission and the protection domain. If the request is using the HTTPS protocol and the implies returns false, the HTTP 403 error is returned to imply excluded/precluded permission and no further checks are performed. If the request is not using the HTTPS protocol, and the implies returns false, the request is redirected over HTTPS.

The provider's implies() method is called using the Permission object and the ProtectionDomain created previously. If the user is granted permission to access the resource, the implies() method should return true. If the user is not granted access, the implies() method should return false.

11.6.2 JACC policy context identifiers in WebSphere V6

JACC Specification defines that "*It must be possible to define separate authorization policy contexts corresponding to each deployed instance of a J2EE module. This per module scoping of policy context is necessary to provide for the independent administration of policy contexts corresponding to individual application modules (perhaps multiply deployed) within a common Policy provider. Each policy context contains all of the policy statements (as defined by this specification) that affect access to the resources in one or more deployed modules. At policy configuration, a PolicyConfiguration object is created for each policy context, and populated with the policy statements (represented by permission objects) corresponding to the context. Each policy context has an associated policy context identifier.*"

A policy context identifier is defined as a unique string that represents a policy context. WebSphere Application Server makes the contextID unique by using the string href:cellName/appName/moduleName as the contextID format for the modules. The href part of the string indicates that a hierarchical name is passed as the contextID.

11.6.3 WebSphere Extensions to the JACC Specification

WebSphere provides three extension interfaces to the JACC Specification; they are InitializeJACCProvider, RoleConfiguration and RoleConfigurationFactory.

The JACC Specification only specifies a contract to propagate the policy information to the provider. There is no contract specified to propagate the authorization table information to the provider. It is left to the provider to present some kind of management interface to handle principals (users/groups) to roles. In order to propagate the authorization table information, WebSphere Application Server provides interfaces RoleConfigurationFactory and RoleConfiguration. The implementation of these interfaces is optional. In some cases, the JACC provider requires initialization during server startup so that it can communicate with the server during startup. WebSphere provides the InitializeJACCProvider interface for this reason. When this interface is implemented, it is called during server startup. Any custom properties in the JACC configuration model are propagated to the initialize method of this implementation. The custom properties can be entered either using the administrative console or by scripting.

During server shutdown, the cleanup method is called for any clean-up work that a provider requires. Implementation of this interface is strictly optional, and should be used only if the provider requires initialization during server startup.

The RoleConfiguration interface is used to propagate the authorization information to the provider. This interface is similar to the PolicyConfiguration interface found in Java Authorization Contact for Containers (JACC).

The RoleConfigurationFactory interface is similar to the PolicyConfigurationFactory interface introduced by JACC, and is used to obtain RoleConfiguration objects based on the contextIDs.

11.6.4 JACC policy propagation in WebSphere V6

The policy propagation between the WebSphere application server and JACC Provider is handled in the following ways.

- ▶ A new application is installed and the configuration is saved.
- ▶ An application is uninstalled and the configuration is saved.

- ▶ There is an update to an existing application either with a new module or an update to an existing module with security policy changes.

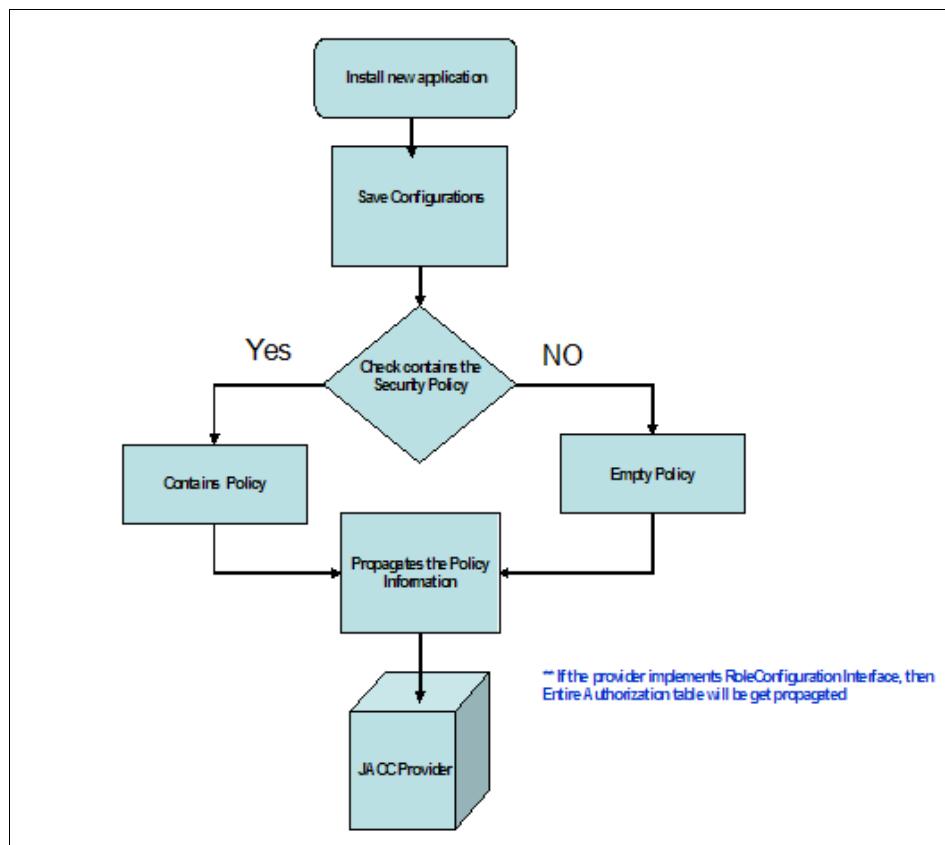


Figure 11-5 JACC policy propagation during application install

When an application is installed or deployed in the WebSphere Application Server, the security policy information in the application is propagated to the provider when the configuration is saved. The contextID for that application is saved in its application.xml file, used for propagating the policy to the JACC provider, and also for access decisions for J2EE resources.

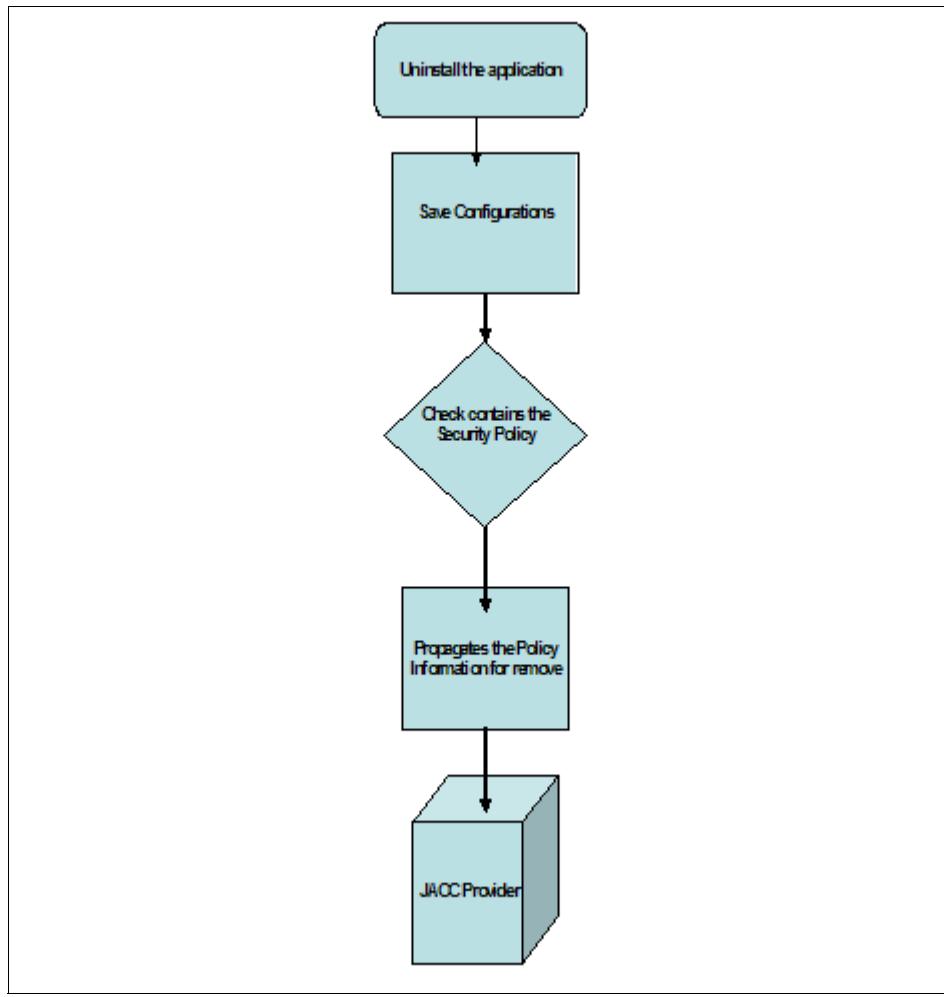


Figure 11-6 JACC policy removal during application uninstall

When an application is uninstalled, the security policy information in the application is removed from the provider when the configuration is saved.

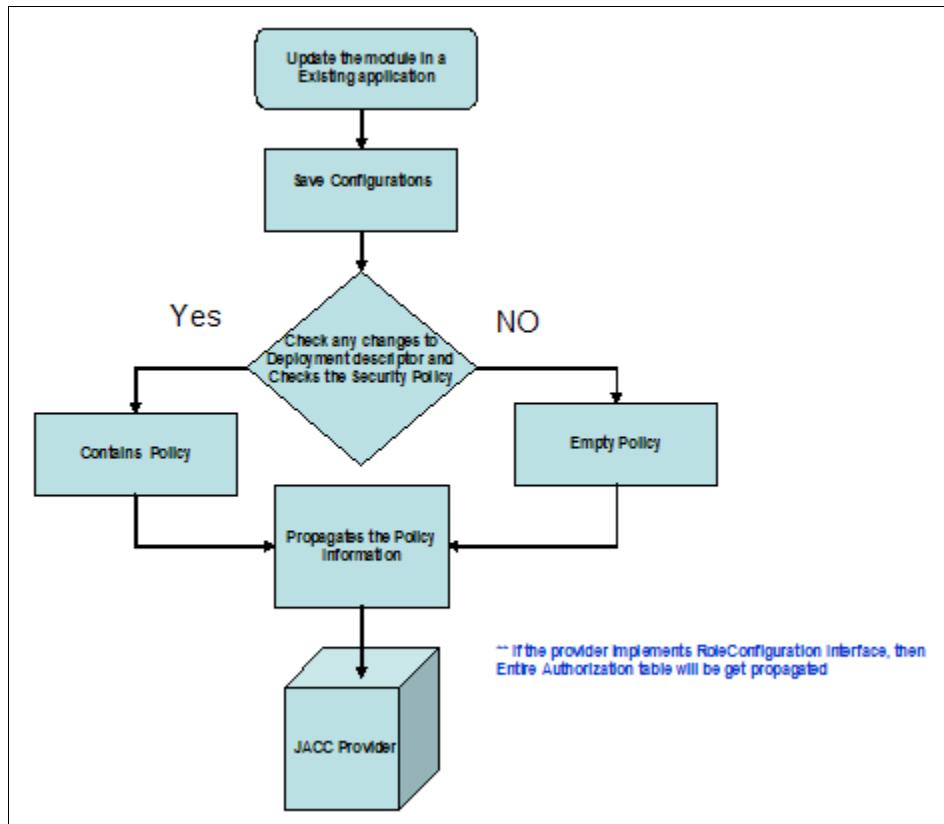


Figure 11-7 JACC policy update during application update

If you update the existing application or add a new module to an existing application, the information in the impacted modules is propagated to the provider by default. A module is impacted when the deployment descriptor of the module has changed as part of the update. If the provider supports the `RoleConfiguration` interfaces, the entire authorization table for that application is propagated to the provider.

If, for some reason, the security information should not be propagated to the provider during application updates, you can set the JVM property `com.ibm.websphere.security.jacc.propagateonappupdate` to `false` in the deployment manager or the unmanaged base application server. If this property is set to `false`, then none of the updates to an existing application in the server is propagated to the provider. You also can set this property on a per-application basis using the custom properties of an application. The `wsadmin` tool can be used to set the custom property of an application. If this property is set at the application level, none of the updates to that application is propagated to the

provider. If the update to an application is a full update, for example a new application .ear file is used to replace the existing one, the provider is then refreshed with the entire application security policy information.

In the network deployment (ND) environment, when an application is installed and saved, the security policy information in that application is updated in the provider from the deployment manager (dmgr or cell). However, the application is not propagated to its respective nodes until the synchronization command is issued and completed. Also, in the ND setup, when an application is uninstalled and saved at the deployment manager, the policy for that application is removed from the JACC provider. However, unless the synchronization command is issued and completed from the deployment manager to the nodes hosting the application, the applications are still running in the respective nodes. In this instance, any access to this application should be denied since the JACC provider does not contain the required information to make the access decision for that application. Note that any updates to the application already installed as described above are also propagated to the provider from the deployment manager. The changes in the provider are not in sync with the applications in the nodes until the synchronization is completed.

11.6.5 Dynamic module updates in WebSphere V6 for JACC

WebSphere handles the dynamic module update with respect to JACC for Web modules. When the Web module is updated, only that particular application needs to be restarted in native authorization mode. In the case of JACC enabled, it depends on the provider support to handle the dynamic module updates very specific to the security modules. There is a dynamic module check box that needs to be checked in order to take effect.

11.7 Integrating Tivoli Access Manager as an external JACC provider

The following steps will guide you through the configuration of WebSphere Application Server to use Tivoli Access Manager as the external authorization engine.

1. Start the WebSphere Application Server Administrative Console, then log in.
2. Click **Security** → **Global Security** from the left navigation menu.
3. Under Authorization, click **Authorization Providers**.
4. Under General Properties, click **External JACC provider**.

If the Tivoli Access Manager Properties are not pre-filled, specify the following properties:

- J2EE policy class name: com.tivoli.pd.as.jacc.TAMPolicy
- Policy configuration factory class name:
com.tivoli.pd.as.jacc.TAMPolicyConfigurationFactory
- Role configuration factory class name:
com.tivoli.pd.as.jacc.TAMRoleConfigurationFactory
- JACC provider initialization class name:
com.tivoli.pd.as.jacc.cfg.TAMConfigInitialize
- Requires the EJB arguments policy context handler for access decisions:
false
- Supports dynamic module updates: true

5. Under Additional Properties, click **Tivoli Access Manager Properties**.

6. Enter the following information:

- *Enable embedded Tivoli Access Manager* - Select this option to enable the Tivoli Access Manager.
- *Ignore errors during embedded Tivoli Access Manager Disablement* - Select this option when you want to unconfigure the JACC provider. Do not select this option during configuration.
- *Client listening point set* - WebSphere Application Server must listen using a TCP/IP port for authorization database updates from the policy server. More than one process can run on a particular node or machine. Enter the listening ports used by Tivoli Access Manager Clients, separated by a comma. If a range of ports is specified, separate the lower and higher values by a colon (for example, 7999, 9990:999).
- *Policy server* - Enter the name of the Tivoli Access Manager Policy server and the connection port. Use the form policy_server:port. The policy communication port is set at the time of the Tivoli Access Manager configuration, and the default is 7135.
- *Authorization servers* - Enter the name of the Tivoli Access Manager Authorization server. Use the form auth_server:port:priority. The authorization server communication port is set at the time of the Tivoli Access Manager configuration, and the default is 7136. More than one authorization server can be specified by separating the entries with commas. Specifying more than one authorization server at a time is useful for reasons of failover and performance. The priority value is determined by the order of the authorization server use (for example, auth_server1:7136:1, and auth_server2:7137:2). A priority value of 1 is required when configuring against a single authorization server.

- *Administrator user name* - Enter the Tivoli Access Manager Administrator user name that was created when Tivoli Access Manager was configured (it is usually sec_master).
- *Administrator user password* - Enter the Tivoli Access Manager administrator password.
- *User registry distinguished name suffix* - Enter the distinguished name suffix for the user registry that is shared between Tivoli Access Manager and WebSphere (for example, dc=raleigh,dc=ibm,dc=com).
- *Security domain* - You can create more than one security domain in Tivoli Access Manager, each with its own administrative user. Users, groups and other objects are created within a specific domain, and are not permitted to access resource in another domain. Enter the name of the Tivoli Access Manager security domain that is used to store WebSphere Application Server users and groups. If a security domain has not been established at the time of the Tivoli Access Manager configuration, leave the value as Default.
- *Administrator user distinguished name* - Enter the full distinguished name of the WebSphere security administrator ID (for example, cn=wasdmin, dc=raleigh,dc=ibm,dc=com). The ID name must match the Server user ID on the LDAP User Registry panel in the administrative console. To access the LDAP User Registry panel, click **Security** → **Global Security**. Under User registries, click **LDAP**.

11.7.1 Disabling the embedded Tivoli Access Manager

In a Network Deployment architecture, ensure all managed servers, including node agents, are started, then perform the following process once on the deployment management server. Information from the unconfigure operation is forwarded to managed servers, including node agents, when the server is restarted. The managed servers then require a restart for changes to take effect.

Disabling using the Administrative Console

To unconfigure the Tivoli Access Manager JACC provider using the WebSphere Application Server Administrative Console, complete the following steps.

1. Disable Global Security by clicking **Security** → **Global security** and deselect the **Enable global security** option.
2. Restart the application server.
3. Select **Security** → **Global security**.
4. Under Authorization, click **Authorization Providers**.
5. Under Related items, click **External JACC provider**.

6. Under Additional properties, click **Tivoli Access Manager Properties**. The configuration screen for the Tivoli Access Manager JACC provider is displayed.
7. Deselect the **Enable embedded Tivoli Access Manager** option. If you want to ignore errors when unconfiguring, select the **Ignore errors during embedded Tivoli Access Manager Disablement** option. Select this option only when the Tivoli Access Manager domain is in an irreparable state.
8. Click **OK**.
9. (Optional) If you want security enabled, without Tivoli Access Manager, re-enable Global Security.
10. Restart all WebSphere Application Server instances for the changes to take effect.

Disabling using wsadmin

To unconfigure the Tivoli Access Manager JACC provider, follow these steps.

1. Disable Global Security by clicking **Security** → **Global security** and deselect the **Enable global security** option.
2. Restart the server or, in a network deployment architecture, restart the deployment manager process.
3. Start the wsadmin command line utility.
4. From the wsadmin prompt, enter the following command:
`$AdminTask unconfigureTAM -interactive`
5. When all information is entered, enter F to save the properties (or C to cancel the unconfiguration process and discard entered information).
6. (Optional) If you want security enabled, not using Tivoli Access Manager, re-enable Global Security.
7. Restart all WebSphere Application Server instances for the changes to take effect.

Reconfiguring using wsadmin

Reconfigure the Java Authorization Contract for Containers (JACC) provider using the `$AdminTask reconfigureTAM interactive` wsadmin command. Enter all new and existing options.

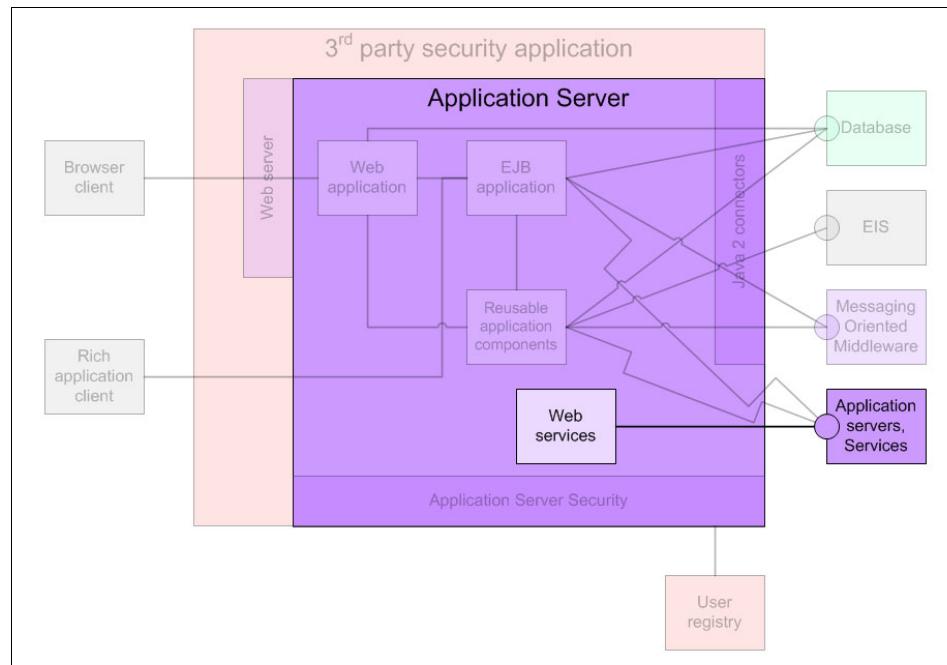
11.8 Sample application for JACC

You can find the details about the sample application in “Sample application for testing JACC” on page 391.



Chapter 12

Web services security



12.1 Security overview

Since the early days of the Internet as a universal network open to anyone, information exchange has been a concern. Although it is worth noting that there is no absolute security, developments in this field have been very quick and fruitful because they were driven by urgent business needs. Without an appropriate level of security, the commercial exploitation of the Internet would not be feasible.

Networks must be designed to provide a high level of security for information that travels across the Internet or privately managed intranets or extranets.

Algorithms such as third-party authentication, public key encryption, and digital signature can provide a sufficient level of security. However, security not only depends on algorithms, standards, and products. Companies are required to follow security best-practice recommendations.

Note: A company's security policy should reasonably cover the most important procedures, such as user management (adding/removing users and granting their rights and access levels), network use guidelines (private mail, Web site access policy, and antivirus protection), user authentication procedure (user ID/password, key cards), system monitoring procedures, and procedures in case of attack.

A general security framework should address the following requirements:

- ▶ **Identification:** the party accessing the resource is able to identify itself to the system.
- ▶ **Authentication:** Authentication is the process of validating the user, whether a client is valid in a particular context. A client can be either an end user, a machine or an application.
- ▶ **Authorization:** authorization is the process of checking whether the authenticated user has access to the requested resource.
- ▶ **Integrity:** ensure that information will not be changed, altered, or lost in an unauthorized or accidental manner.
- ▶ **Confidentiality:** no unauthorized party or process can access or disclose the information.
- ▶ **Auditing:** all transactions are recorded so that problems can be analyzed after the fact.
- ▶ **Non-repudiation:** both parties are able to provide legal proof to a third party that the sender did send the information, and the receiver received the identical information. Neither involved side is “unable to deny.”

Some classifications also include *availability* to be a part of the above schema, meaning that hostile attack cannot achieve denial-of-service by allocating too many system resources. In this chapter, we do not deal with this security aspect.

12.2 Web service security exposures

Web services security is one of the most important Web services subjects. When using Web services, similar security exposures exists as for other Internet, middleware-based applications and communications.

To explain the Web service security exposures, let us use a bank teller scenario as an example, as shown in Figure 12-1. The bank teller (Web service consumer) connects over the Internet to the bank's data center (Web service provider). We assume there is no security applied at all, which is not realistic, but necessary for the example.

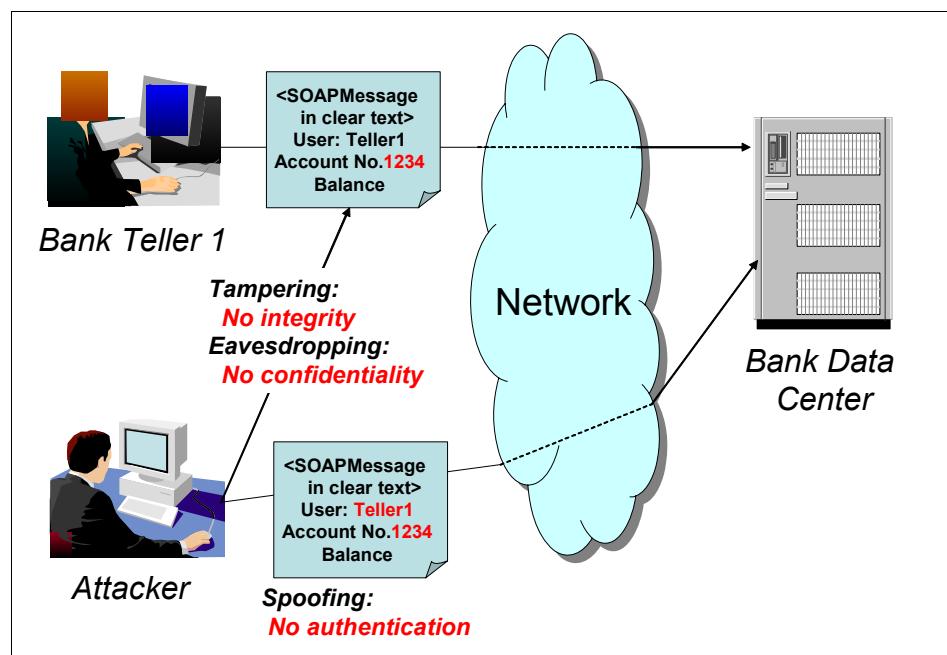


Figure 12-1 Common security exposures in a sample bank teller application based on Web services

The three major risk factors in this example are:

► **Spoofing: no authentication**

An attacker could send a modified SOAP message to the service provider, pretending to be a bank teller, to get confidential information, or to withdraw money from another customers account.

Applying authentication to the Web service, this security exposure can be eliminated.

► **Tampering: no integrity**

The SOAP message is intercepted between the Web service client and server. An attacker could modify the message, for example, deposit the money into another account by changing the account number. As there is no integrity constraint, the Web service server does not check if the message is valid, and will accept the modified transaction.

Applying integrity mechanism to the Web service, this security exposure can be eliminated.

► **Eavesdropping: no confidentiality**

An attacker can intercept the SOAP message, and read all contained information. Since the message is not encrypted, confidential customer or bank information can end up in the wrong hands.

This exposure exists because the account and balance information is sent over the network in plain text.

Applying a confidentiality mechanism to the Web service, this security exposure can be eliminated.

To prevent the described security exposures, the following mechanisms can be applied to secure a Web services environment (also shown in Figure 12-2 on page 293):

- Message level security: Web services security (WS-Security)
- Transport level security: TLS/SSL

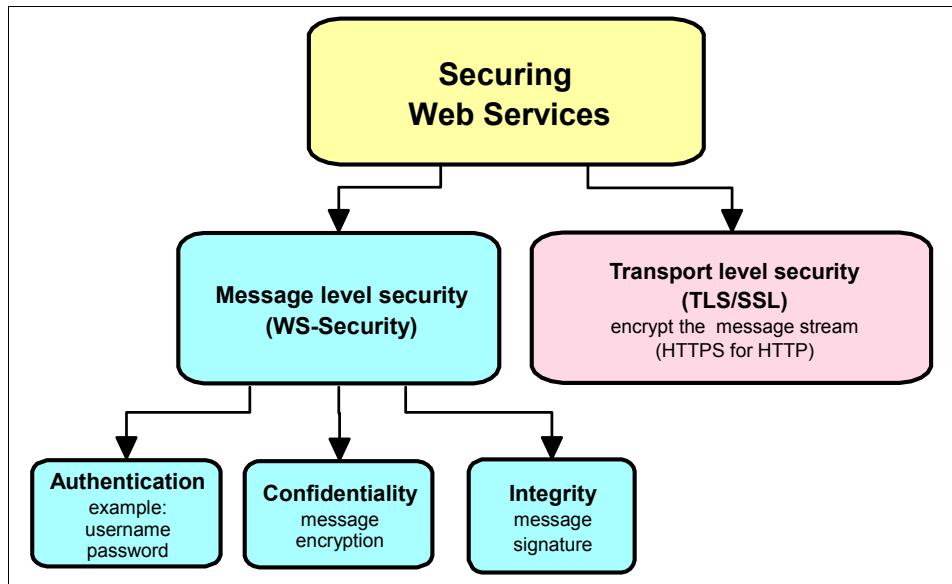


Figure 12-2 Securing Web services

Depending on the demanded level of application security, one or more of these security mechanisms can be applied.

Also, a combination of message-level security and transport-level security can be implemented.

The more security mechanism are implemented, which increases the security effect, the more influence on other non-functional requirements is given. So, when designing a Web services security solution, one has to keep in mind that security has an impact on the following non-functional requirements:

- ▶ **System capacity**

Any applied security mechanism has an impact on system resource usage (for example CPU and memory usage). So, when planning a Web service environment, the required security *overhead* must be considered in the system capacity and volume planning.

The non-functional requirements, capacity and volume, cover for example the number of concurrent users, and the number of transactions per second. This has influence on the required system infrastructure (hardware, network).

- ▶ **Performance**

Security mechanisms and functions also impact the applications response time. When defining the Web service system response time requirements,

one has to mind that the response times will be affected when applying security.

The performance requirement for a system defines the response time for a main application operation (for example: less than 1 second for 90 % of all transactions).

Note: Applying security is not only a question of feasibility; the additional system resources and the influence on the response time must also be considered.

The WS-Security specification, and SSL mechanism will be covered in detail in the next sections.

12.3 WS-Security

This section introduces WS-Security concepts. You can find more information about the various WS-Security specifications in 12.3.3, “WS-Security Roadmap” on page 297.

12.3.1 WS-Security concepts

The WS-Security specification provides a message-level security which is used when building secure Web services to implement message content integrity and confidentiality. The advantage of using WS-Security over SSL is that it can provide End-to-End Message Level security. This means that the message security can be protected even if the message goes through multiple services; so called intermediaries. Additionally, WS-Security is independent of the transport layer protocol, it can be used for any SOAP binding (for example HTTP, SOAP, RMI). Using WS-Security, End-to-End security can be obtained (see Figure 12-3).

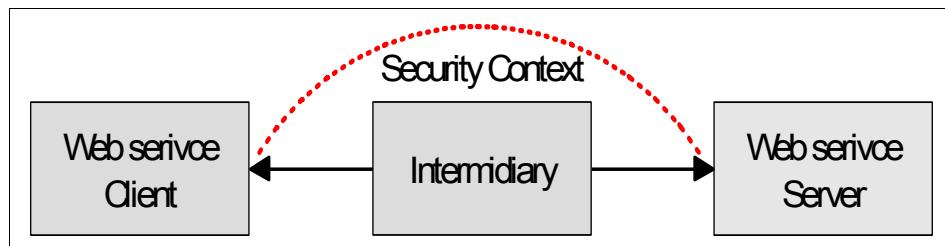


Figure 12-3 End to end security with message level security

The WS-Security specification, Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), is proposed by the OASIS WSS Technical Committee. This specification proposes a standard set of SOAP extensions. This specification is flexible and is designed to be used as the basis for securing Web services within a wide variety of security models including PKI, Kerberos, and SSL. It provides support for multiple security token formats, multiple trust domains, multiple signature formats, and multiple encryption technologies based on XML Signature and XML Encryption to provide integrity or confidentiality.

The specification includes security token propagation, message integrity, and message confidentiality. However, these mechanisms by themselves do not address all the aspects of complete security solution. Therefore, WS-Security represents only one of the layers in a complex secure Web services solution design.

Important: With WS-Security 1.0, the wire format changed in a way which is not compatible with previous WS-Security drafts. Also, interoperability between implementations based on previous drafts and version 1.0 is not possible.

The WS-Security specification defines the usage of XML Signature and XML Encryption:

- ▶ Message integrity is provided by XML Signature in conjunction with security tokens to ensure that modifications to messages are detected. See:
<http://www.w3c.org/Signature>
- ▶ Message confidentiality leverages XML Encryption in conjunction with security tokens to keep portions of a SOAP message confidential. See:
<http://www.w3c.org/Encryption>

12.3.2 Evolution of the WS-Security specification

The WS-Security support is provided in WebSphere 5.0.2 and later; each version of WebSphere is based on different versions of the Web services security language.

The first version of the WS-Security specification was proposed by IBM, Microsoft, and Verisign in April 2002. After the formalization of the April 2002 specifications, the specification is transferred to OASIS consortium; see:

<http://www.oasis-open.org>

In OASIS activities, core specification and many profiles which describe the use of a specific token framework in WS-Security have been discussed. The latest

specification and profiles of WS-Security were proposed in March 2004 as the OASIS Standard. The latest core specification, Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) was standardized in March 2004. The two profiles, Web Services Security UsernameToken Profile 1.0 and Web Services Security X.509 Certificate Token Profile 1.0, were standardized at the same time.

There are other token profiles that OASIS is currently working on: Web Services Security: SAML Token Profile, Web Services Security: Rights Expression Language (REL) Token Profile, Web Services Security: Kerberos Token Profile, Web Services Security Minimalist Profile (MProf) and Web Services Security: SOAP Message with Attachments (SwA) Profile.

The support of the April 2002 specification is provided in WebSphere 5.0.2 and 5.1. WebSphere Application Server version 6.0 supports the WS-Security 1.0 specification and two profiles (UserName-Token 1.0, x.509 Token 1.0).

Figure 12-4 shows the evolution of WS-Security.

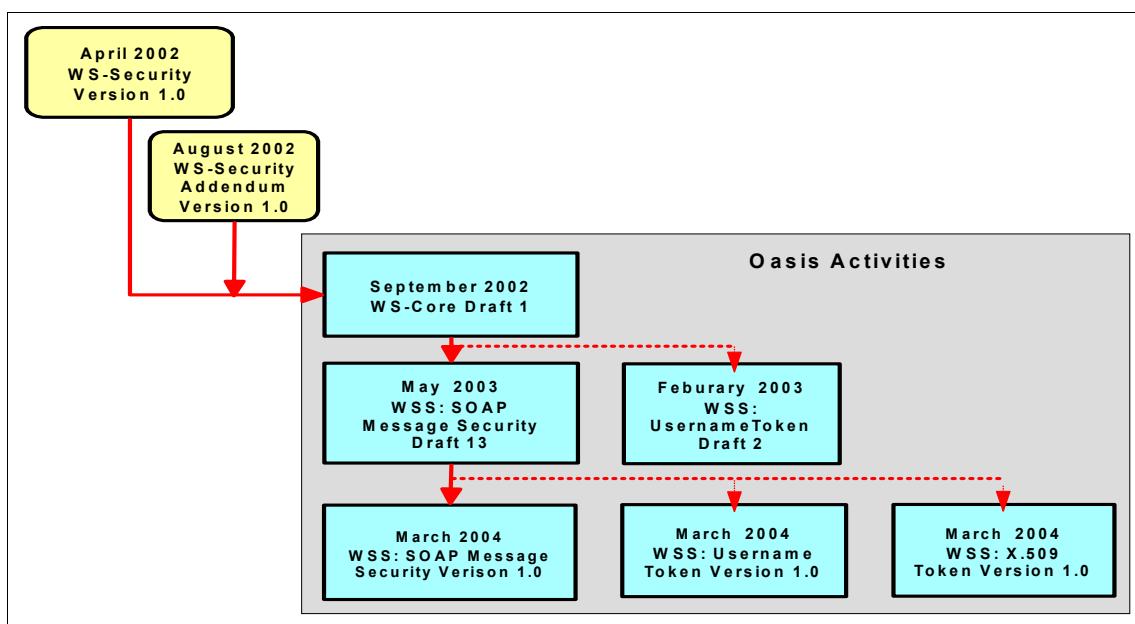


Figure 12-4 Evolution of Web services security

To read more about these standards, refer to:

- ▶ Specification: Web Services Security (WS-Security) Version 1.0 (April 2002):
<http://www-106.ibm.com/developerworks/webservices/library/ws-secure/>
- ▶ Web Services Security Addendum (August 2002):

- ▶ <http://www-106.ibm.com/developerworks/webservices/library/ws-secureadd.html>
- ▶ Web Services Security: SOAP Message Security V1.0 (March 2004):
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>
- ▶ Web Services Security: UsernameToken Profile V1.0 (March 2004):
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf>
- ▶ Web Services Security: X.509 Token Profile V1.0 (March 2004):
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf>

12.3.3 WS-Security Roadmap

As mentioned, the WS-Security specification addresses only a subset of security services for all security aspects. A more general security model is needed to cover other security aspects, such as logging and non-repudiation. The definition of those requirements is given in a common Web services security model framework, a security white paper of Web Services Security Roadmap proposed by IBM and Microsoft. We describe this Roadmap in the following section.

Web services security model framework

The Web services security model introduces a set of individual interrelated specifications to form a layering approach to security. It includes several aspects of security: Identification, authentication, authorization, integrity, confidentiality, auditing, and non-repudiation. It is based on the WS-Security specification, co-developed by IBM, Microsoft, and VeriSign.

The Web services security model is schematically shown in Figure 12-5 on page 298.

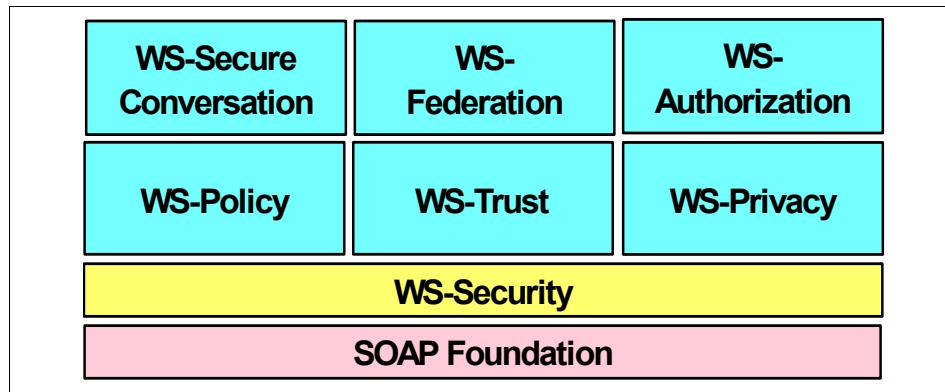


Figure 12-5 WS-Security Roadmap

These specifications include different aspects of Web services security:

- ▶ **WS-Policy**

Describes the capabilities and constraints of the security (and other business) policies on intermediaries and endpoints (for example, required security tokens, supported encryption algorithms, and privacy rules).

- ▶ **WS-Trust**

Describes a framework for trust models that enables Web services to securely interoperate. This specification is responsible for managing trusts and establishing trust relationships.

- ▶ **WS-Privacy**

Describes a model for how Web services and requestors state privacy preferences and organizational privacy practice statements.

- ▶ **WS-Federation**

Describes how to manage and broker the trust relationships in a heterogeneous federated environment, including support for federated identities.

- ▶ **WS-Authorization**

Describes how to manage authorization data and authorization policies.

- ▶ **WS-SecureConversation**

Describes how to manage and authenticate message exchanges between parties, including security context exchange and establishing and deriving session keys.

The combination of these security specifications enables many scenarios that are difficult or impossible to implement with today's more basic security mechanisms such as transport securing or XML document encryption.

12.3.4 Example of WS-Security

Here we provide some examples of SOAP messages with WS-Security. Using WS-Security, authentication mechanism, integrity and confidentiality can be applied in the message level. In WebSphere V6.0, there are many options to apply these security mechanisms. In this section, the most typical scenarios of each mechanism are shown as an introduction.

As an overview, Figure 12-6 shows the elements Web service security elements which are added to the SOAP message.

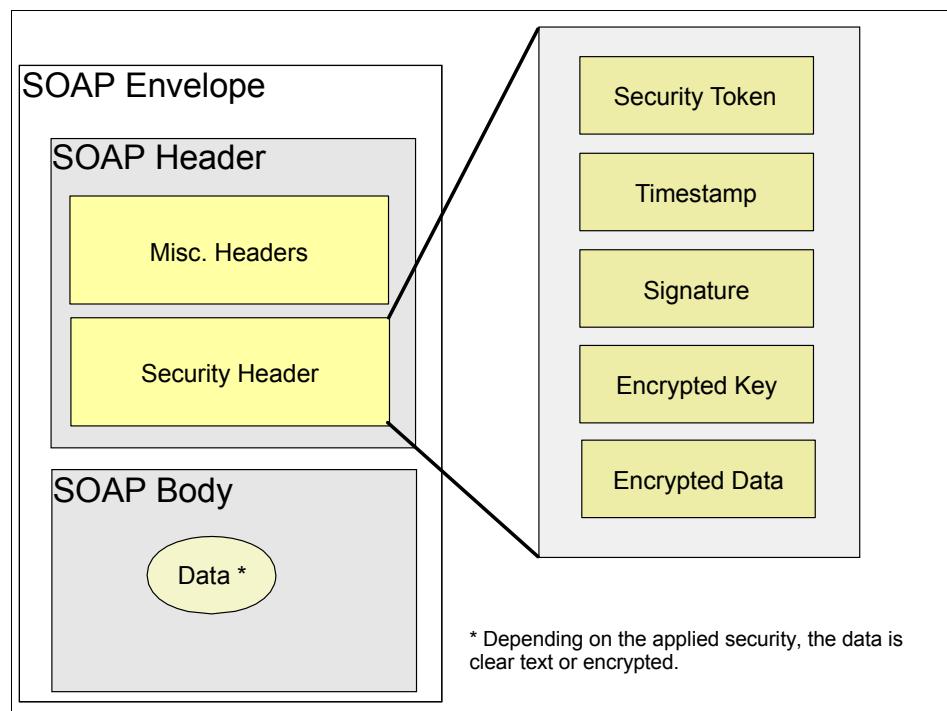


Figure 12-6 SOAP message security with WS-Security

Applying WS-Security, the SOAP security header will be inserted under the SOAP envelope.

Authentication

The username and password information as a Username Token is stored in the message. When the Username Token is received by the Web service server, the username and password are extracted from the Username Token and they are verified. Only when both the username and password are valid, the message will be accepted and processed at the server.

Using Username Token is just one of the ways of implementing authentication. This mechanism is also known as *basic authentication*. Other forms of authentication are digital signature, ID Assertion, LTPA, and custom tokens. These other mechanisms can be configured using Application Developer.

Steps to enable a basic authentication in your application

The way to configure authentication is shown simply in the following steps.

1. Client side

To insert the Username Token to a SOAP message, a security token and its token generator should be specified in the client's WS-Security configuration.

- a. Specify a security token at Request Generator configuration. In case of using *basic authentication*, the security token type should be Username. This security token is sent inside the SOAP message to the server.
- b. Specify a token generator for Username Token at Request Generator configuration. The role of the token generator is to get the username and password from the configuration file and generate the Username Token with the username and password. The token generator class for Username Token, UsernameTokenGenerator, is provided by the WebSphere Web services security runtime as a default implementation.

2. Server side

To receive the client's Username Token, a security token which is required by the server and a token consumer should be specified in the server's WS-Security configuration.

- a. Specify a security token which is required by the server, in case of *basic authentication*, the required security token type is a Username as for a client's configuration.
- b. Specify a token consumer at Request Consumer configuration. The token consumer receives a security token in the request message and validates it. The token consumer class for Username Token, UsernameTokenConsumer, is provided by the WebSphere Web services security runtime as a default implementation.
- c. Turn on Global Security in the WebSphere Application Server where the application is deployed.

Integrity

Integrity is applied to the application to ensure that no one illegally modifies the message while it is in transit. Essentially, integrity is provided by implementing an XML digital signature on the contents of the SOAP message. If the message data changes illegally, the signature would no longer be valid.

In WebSphere V6.0, multiple and arbitrary parts of the message can be signed, for example a message body, security token and time stamp.

A signature is created based on a key that the sender is authorized to have. Unauthorized sniffers do not have this key. When the receiver gets the message, it too creates a signature using the message contents. Only if the two signatures match does the receiver honor the message. If the signatures are different, an error is returned to the sender.

Steps to enable integrity in your application

The way to configure integrity is shown in the following steps.

1. Client side

To specify the integrity of part of a SOAP message, you need to specify the part which should be signed and the way of signing in the client's WS-Security configuration.

- a. Specify the parts of the message that have to be signed at Request Generator configuration. The message parts can be specified by the predefined keyword or XPath expression. Also, you can specify multiple parts which needs a signature.
- b. Specify key-related information which includes the location of the client's key, a type of key and a password for protecting the key.
- c. Specify signing information which defines how to sign to the specified part. You need to specify some options for signature such as a signature method algorithm or key-related information. Application Developer helps you to specify these options.
- d. In a most typical integrity example, a security token is inserted in the SOAP message, which will be used as signature verification by the server. In such an example, a token generator should be specified at Request Generator configuration. This token generator's role is to generate a token for signature verification. In this case, a token generator for X.509 certificate token, X509TokenGenerator, should be specified, which is provided by the WebSphere Web services security runtime as a default implementation.
- e. If a client expects a response that includes integrity information by the server, then the client also has to be configured to validate the integrity of the response message at Response Consumer configuration.

2. Server side

To specify required integrity for part of a SOAP message, you need to specify the part which should be signed and the way of verifying of the signature in the server's WS-Security configuration.

- a. Specify the parts of the message which require a signature at Request Consumer configuration. The message parts can be specified by the predefined keyword or XPath expression. Also, you can specify multiple parts which need a signature.
- b. Specify key-related information which includes the location of the server's key, a type of key and a password for protecting the key.
- c. Specify signing information which defines how the specified part is to be signed. You need to specify some options for signature, such as a signature method algorithm or key-related information. Application Developer helps you to specify these options.
- d. In a most typical integrity example, a security token is inserted into the SOAP message, which will be used as signature verification by the server. In such an example, a token consumer should be specified at Request Consumer configuration. This token consumer's role is to receive the token for signature verification. In this case, a token consumer for X.509 certificate token, X509TokenConsumer, should be specified; it is provided by the WebSphere Web services security runtime as a default implementation.
- e. If a server needs a response that includes integrity information by the server, then the server also has to be configured to sign the response message at Response Generator configuration.

Confidentiality

In WebSphere V6.0, multiple and arbitrary parts of the message can be encrypted, for example a message body, security token and so on.

Confidentiality is the process by which a SOAP message is protected so that only authorized recipients can read it. Confidentiality is provided by XML encryption of the contents of the SOAP message. If the SOAP message is encrypted, only one who knows the key for confidentiality can decrypt and read the message.

Steps to enable confidentiality in your application

The simplified steps to enable confidentiality follow.

1. Client side

To specify confidentiality of part of a SOAP message, you need to specify the part which should be encrypted and the manner of encryption in the client's WS-Security configuration.

- a. Specify the parts of the message that have to be encrypted at Request Generator configuration. The message parts can be specified by the predefined keyword or XPath expression. Also, you can specify multiple parts that need encryption.
 - b. Specify key-related information which includes the location of the client's key, type of key and a password for protecting the key.
 - c. Specify encryption information which defines how to encrypt the specified part. You need to specify some options for encryption such as an encryption method algorithm, and key-related information. Application Developer helps you to specify these options.
 - d. If a client expects a response that includes confidentiality by the server, then the client also has to be configured to decrypt the server's encryption of the response message at Response Consumer configuration.
2. Server side

To specify required confidentiality for part of a SOAP message, you need to specify the part which should be encrypted and the way of decrypting the encryption in the server's WS-Security configuration.

- a. Specify the parts of the message which require decryption at Request Consumer configuration. The message parts can be specified by the predefined keyword or XPath expression. Also, you can specify multiple parts that need a signature.
- b. Specify key-related information, including the location of the server's key, a type of key and a password for protecting the key.
- c. Specify encryption information which defines how to decrypt the specified part. You need to specify some options for encryption such as an encryption method algorithm and key-related information. Application Developer helps you to specify these options.
- d. A token consumer should be specified at Request Consumer configuration. This token consumer's role is to receive information for message decryption. In this case, a token consumer for X.509 certificate token, X509TokenConsumer, should be specified; it is provided by the WebSphere Web services security runtime as a default implementation.
- e. If a server needs a response that includes confidentiality by the server, then the server also has to be configured to encrypt the response message at Response Generator configuration.

12.4 Transport-level security

HTTP, the most used Internet communication protocol, is currently also the most popular protocol for Web services. HTTP is an inherently insecure protocol, because all information is sent in clear text between unauthenticated peers over an insecure network. It belongs to the group of protocols, such as SMTP, telnet, and FTP, that were designed in the earlier stages of the Internet, when security seemed not to be an issue, and will eventually be replaced by transfer protocols that allow authentication and encryption.

To secure HTTP, transport-level security can be applied. Transport-level security is a well-known and often used mechanism to secure HTTPS inter- and intranet communications. Transport-level security is based on Secure Sockets Layer (SSL) or Transport Layer Security (TLS) that runs beneath HTTP.

HTTPS allows client and server-side authentication through certificates, which have been either self-signed or signed by a certification agency.

HTTPS can be assigned in any combination with any parts of message level security (WS-Security).

Unlike message-level security, HTTPS encrypts the *entire* HTTP data packet. There is no option to apply security selectively only on certain parts of the message. SSL and TLS provide security features including authentication, data protection, and cryptographic token support for secure HTTP connections.

We will not cover HTTPS in more detail; please refer to the points in “More information” on page 305.

12.4.1 SOAP/HTTP transport-level security

Although HTTPS does not cover all aspects of a general security framework, it provides a security level regarding party identification and authentication, message integrity, and confidentiality. It does not provide authentication, auditing, and non-repudiation. SSL cannot be applied to other protocols, such as JMS. To run HTTPS, the Web service port address must be in the form `https://`.

Even with the WS-Security specification, SSL should be considered when thinking about Web services security. Using SSL, a so-called point-to-point security can be achieved (see Figure 12-7 on page 305).

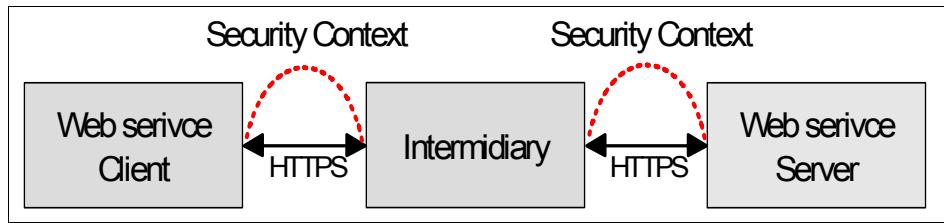


Figure 12-7 Point-to-point security with HTTPS

12.5 Summary

Web services technology enables a loosely coupled, language-neutral, platform-independent way of linking applications within organizations, across enterprises, and across the Internet. To achieve the target, however, it is essential for Web services to provide a sufficient level of security to support business transactions. Ensuring the integrity, confidentiality, and security of Web services through the application of a comprehensive security model is critical, both for organizations and their customers.

With WebSphere Application Server V6.0, Web services security can be applied as transport-level security and as message-level security. Highly secure client-server designs can be architected using these security levels.

More information

Because Web services security is a quickly evolving field, it is essential for developers and designers to regularly check for recent updates. In this section, we provide some of the most important entry points for your exploration.

- ▶ XML Signature Workgroup home page can be found at:
<http://www.w3.org/Signture/>
- ▶ XML Encryption Workgroup home page can be found at:
<http://www.w3.org/Encryption/>
- ▶ WS-Security specification 1.0 can be found at:
<http://www.ibm.com/developerworks/library/ws-secure/>
- ▶ The whitepaper of Web services security Roadmap can be found at:
<http://www.ibm.com/developerworks/webservices/library/ws-secmap/>
- ▶ OASIS WS-Security 1.0 and token profiles can be found at:
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

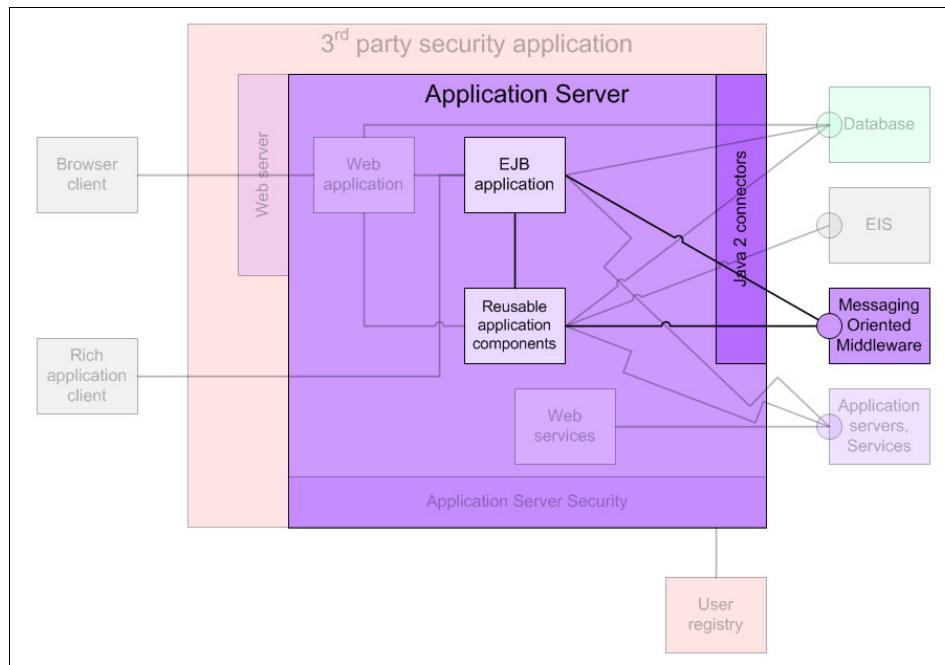
- ▶ For information about IBM Tivoli® Access Manager for Business Integration, refer to this Web site:
<http://www.tivoli.com/products/index/access-mgr-bus-integration/>
- ▶ For information about IBM WebSphere MQ, refer to the standard proposition overview on developerWorks:
<http://www.ibm.com/software/ts/mqseries/messaging/>

There are several commercial and non-commercial information sources that cover more general subjects, such as SSL encoding and HTTPS protocol.



Chapter 13

Securing messaging oriented middleware



13.1 Java Messaging Service API

Java Messaging Service (JMS) is a Java API that allows applications to create, send, receive, and read messages. As part of the J2EE 1.4 Specification, WebSphere Application Server V6 supports the JMS 1.1 API. JMS API in the J2EE platform has the following features.

- ▶ Application clients, Enterprise JavaBean components and Web components can send or synchronously receive a JMS message. Application clients can in addition receive JMS messages asynchronously. (Application clients refer to the application running on the client side in the J2EE client container.)
- ▶ The message-driven bean enables the asynchronous consumption of messages. A JMS provider may optionally implement concurrent processing of messages by message-driven beans.
- ▶ Messages sent and received can participate in distributed transactions.

The JMS specifications do not discuss the security and encryption of the message that is getting transferred using the JMS provider. Instead, specifications leave the security implementation to the JMS provider. We are going to discuss WebSphere MQ and default messaging as JMS providers in this chapter.

Security considerations

The five security considerations for messaging are:

- ▶ Authentication
- ▶ Authorization
- ▶ Confidentiality
- ▶ Data integrity
- ▶ Non-repudiation

Find more details about these topics below.

- ▶ **Authentication** is a mechanism used to check whether the application or the user is genuine. In a WebSphere MQ context, when a message channel starts, it is possible for the message channel agent (MCA) at each end of the channel to authenticate its partner, known as *mutual authentication*. For the sending MCA, this provides assurance that the partner it is about to send messages to is genuine. And for the receiving MCA, there is a similar assurance that it is about to receive messages from a genuine partner.

The application that handles the messaging has to perform the authentication; for example, when a servlet sends a message, WebSphere has to authenticate the user if he/she can run the servlet. Since there is no message-level security (who can send what type of message), the message level should be considered during application design.

- ▶ **Authorization** for the WebSphere MQ objects is stored in MQ (actually in a special queue). WebSphere MQ uses normal operating system user name and group authorizations to protect WebSphere MQ applications and WebSphere MQ Administration.

Access Control (ACL) can be defined for each object. This Access Control service protects critical resources in a system by limiting access only to authorized users and their applications. It prevents the unauthorized user of an object. For example, you can define Access Control so that it only allows that particular application to connect to a queue manager if the user ID associated with the application is authorized to do so.

- ▶ **Confidentiality:** many times you will need to protect the message from unauthorized disclosure and you do not want to ignore the message content confidentiality when the message is travelling over an insecure network such as the Internet. In such cases, there is no help that we can get from Access Control definitions. What we need here is message encryption. For example, after sending the message MCA gets from the transmission queue, the message is encrypted before it is sent over the network to the receiving MCA. At the other end of the channel, the message is decrypted before the receiving MCA puts it on its destination queue.
- ▶ **Data integrity** service describes more about unauthorized modification of the data. Such modification of data is possible in two different cases, through hardware and transmission errors or because of a deliberate attack.

Many hardware products and transmission protocols now have a mechanism to detect and correct hardware and transmission errors. So, for our messaging security, this may not be a threat or concern. But this is not the same with deliberate attacks.

Access control mechanism can contribute to data integrity to an extent since data cannot be modified if access is denied. So the Data Integrity service can be used to detect whether the contents of the message have been modified while it was travelling over the network. This can also be helpful while messages are stored in a local queue; the access control mechanism provided by WebSphere MQ might be sufficient to prevent deliberate modification of the contents of the message. However, for a greater level of security, a data integrity service can be used to detect whether the contents of a message have been deliberately modified between the time the message was put on the queue and the time it was retrieved from the queue.

- ▶ **Non-repudiation** is more about providing with a proof of origin that the message was sent by one particular individual and providing a proof of delivery that can provide the sender with undeniable evidence that the message was received by that particular individual.

For implementation, IBM WebSphere MQ does not provide non-repudiation as part of its base function. However, this can be achieved by writing your own exit programs within the WebSphere MQ environment. The following sections will discuss how Authentication, Authorization and Confidentiality are handled in IBM WebSphere MQ and IBMdefault provider.

13.2 Default JMS provider

WebSphere Application Server V6 includes a 100% Java embedded JMS provider. This messaging system is based on the *Service Integration Bus (SIB)* which is installed by default with the application server. Once configured, the messaging system can be exposed to client applications as a queue manager like in WebSphere Application Server V5 or clients can take advantage of the Service Integration Bus for communicating with the messaging system. For information about defining a *MQ client link* to expose the messaging engine as a queue manager to clients, refer to the WebSphere Application Server V6 Information Center. Communicating via the Service Integration Bus to the embedded JMS provider will be discussed in this section.

13.2.1 Messaging components of the Service Integration Bus

The following sections will discuss the various pieces of the Service Integration Bus and service oriented architecture that work together to provide applications with JMS messaging services. Figure 13-1 on page 311 and Figure 13-2 on page 312 depict a simple messaging infrastructure using WebSphere Application Server Embedded Messaging in a single server and multi-node WebSphere Application Server Network Deployment installation.

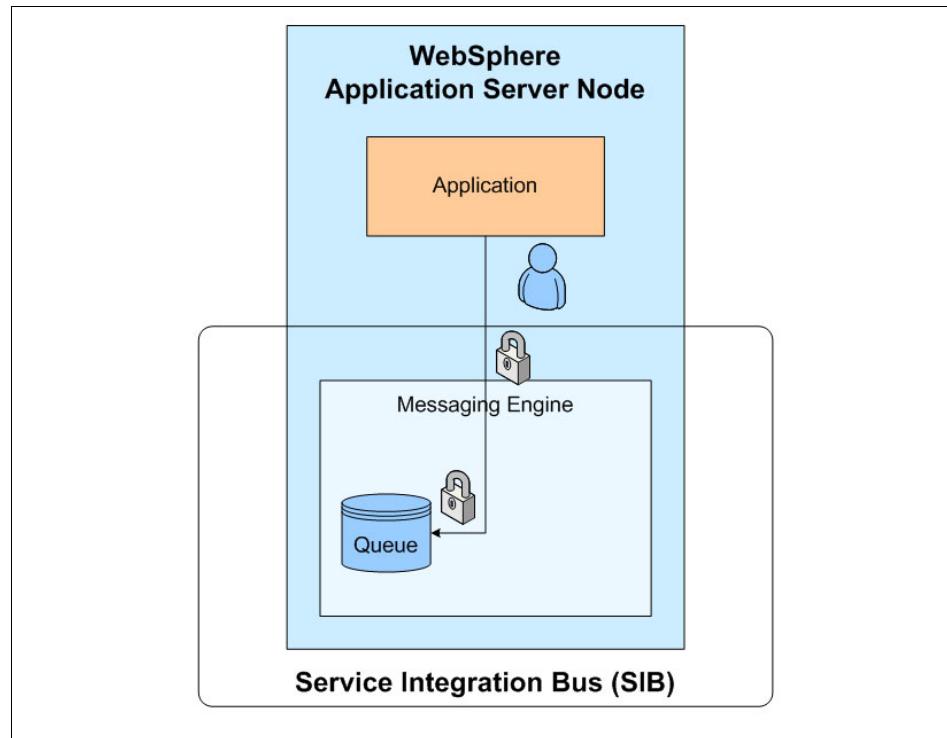


Figure 13-1 Single server messaging

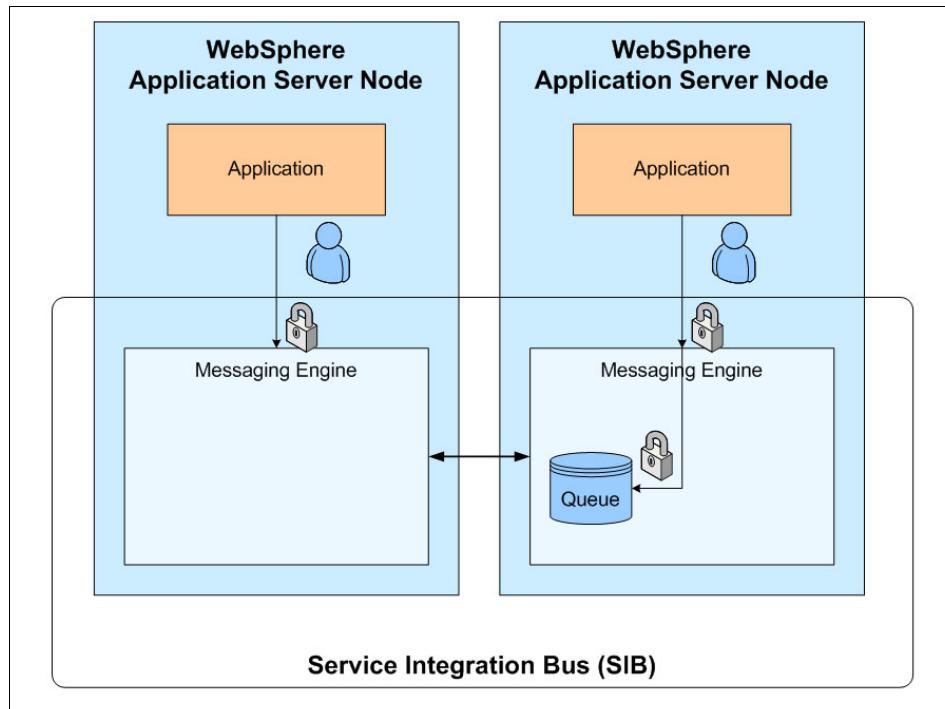


Figure 13-2 Multi-node messaging

Service Integration Bus

The Service Integration Bus provides the basic framework for the application server to participate in a service oriented architecture and to provide JMS messaging services to applications. Using this framework, it is possible to connect multiple JMS providers into a messaging fabric. A bus can be defined as a group of interconnected messaging engines.

Applications connect to the bus at specific points to send messages which are then routed among the servers and clusters connected to the bus. By interconnecting buses, messages can be sent to servers and clusters that are not directly connected to the local Service Integration Bus. These are called foreign buses.

Messaging engine

A messaging engine is the JMS provider running in an application server. The messaging engine hosts the bus destinations that applications send and receive messages from. Applications connect to the local messaging engine sending or receiving messages. Messages sent via the JMS provider are then routed over the bus to the appropriate bus members where the message destination resides.

When a server or cluster is added to the bus, the messaging engine is automatically created for that server. In a single server environment, the engine runs inside the application server.

Foreign bus

A foreign bus is another Service Integration Bus that the local bus can communicate with. A foreign bus can be another WebSphere Application Server Service Integration Bus or a bus defined to communicate with WebSphere MQ. Messages can be routed to the foreign bus directly through a link between the buses or indirectly through one or more intermediary buses. For communication with WebSphere MQ, see 13.3, “Application server and WebSphere MQ” on page 321.

Bus destination

A destination is a location within the Service Integration Bus that applications send messages to or receive messages from. Destinations can be either permanent or temporary. Temporary destinations are used for the application's session only. Destinations are either local to the Service Integration Bus or located on another bus. The main types of destinations are:

- ▶ Queue - Used for point to point messaging.
- ▶ Topic Space - Used for publish subscribe based messaging.
- ▶ Alias - An alternate name that can be used in place of the name of another destination. The actual destination that the alias maps to can either be on the local bus or a foreign bus.
- ▶ Foreign - Used to identify a destination on another bus. This allows the application to access the destination on the foreign bus.

Note: An application can only receive messages from a destination on the bus to which it is connected. An application that subscribes to a local topic space can receive messages from a foreign topic space if topic space names have been mapped between the buses.

13.2.2 Embedded messaging security overview

This section will discuss the three main topics of security as related to embedded messaging in WebSphere Application Server V6. Security can be enabled on the bus if Global Security has been enabled for the application server. When the bus is initially created, a default set of roles is also created. Access to the bus and resources on the bus is role-based and administered through the WebSphere Application Server wsadmin tool. The default roles allow all authenticated users to access the bus and local bus destinations.

Note: Security checks are not done on Service Integration Bus resources if Global Security is not enabled or if Global Security is enabled but *Secure* is not checked on the Bus properties page. To enable Global Security, see Chapter 3, “Global Security” on page 31.

Authentication

In order to access a secured bus, and the resources on the bus, a set of credentials has to be supplied. Once the credentials are verified, authorization to access the bus is checked. Authentication is checked between client and messaging server, as well as between messaging servers on the bus. The credentials are checked against the user registry defined during the Global Security setup for the application server. For Global Security information, see Chapter 3, “Global Security” on page 31.

Authorization

Once authentication has taken place, authorization to access a resource is checked. In order to access a destination on the bus, the user must first be authorized to access the bus. Membership in the BusConnector role determines access to the bus, refer to 13.2.3, “Administering Service Integration Bus security” on page 317 for details on changing role membership. If the user is not assigned role membership then connection is denied. Once connected to the bus, the messaging engine will check roles for the destination being accessed.

Access to bus destinations are based on role membership and the action being performed. The bus destinations each have a set of roles which are checked based on the type of action(s) available for the destination. The Service Integration Bus also has a default set of roles that may apply to all destinations on the bus. The default set of roles and the roles defined on the destination work together to define who can perform what action on the bus destination. For example, to send a message to a queue endpoint the user would need to be a member of the Sender role for the queue endpoint or a member of the DefaultSender role for the bus. The following table, taken from the *WebSphere Application Server V6 Information Center*, lists the bus destination types and the available roles for that destination.

Table 13-1 Destination role names

Destination type	Role names
queue	Sender, Receiver, Browser, Creator, IdentityAdopter
port	Sender, Receiver, Browser, Creator, IdentityAdopter
webService	Sender, Receiver, Browser, Creator, IdentityAdopter

Destination type	Role names
topicSpace	Sender, Receiver, IdentityAdopter
foreignDestination	Sender, IdentityAdopter
alias	Sender, Receiver, Browser, IdentityAdopter

The default roles defined on the bus destinations are as follows.

- ▶ Sender
- ▶ Receiver
- ▶ Browser
- ▶ Creator
- ▶ IdentityAdopter

If Topic access check required is specified on a Topic Space destination then topic level security is enabled. Once topic level security is enabled, additional authorization checks will be performed when users perform actions on topics. By default, everyone has full access to topics within a topic space. Authorization roles for topics are inherited from parent topics in the topic tree. Roles defined for topics are:

- ▶ Sender
- ▶ Receiver
- ▶ IdentityAdopter (only for root topic)

For the commands to administer bus destination authorization see 13.2.4, “Administering destination security” on page 318. For the commands to administer topic level roles, see 13.2.5, “Administering topic space roles” on page 320.

The following table gives a brief description for each of the roles available.

Table 13-2 SIB roles

Role	Capability
Sender	Send a message to the destination.
Receiver	Receive (consume) a message from the destination.
Browser	Browse messages on the destination.
Creator	Create a temporary destination based on the permanent destination.
IdentityAdopter	Role to send message under a different user identity. Does not apply to JMS.

Role	Capability
BusConnectorRole	Those who have a role to connect to the bus.

Transport security: confidentiality

Providing credentials to the messaging engine and gaining access to the bus destination is only half the security battle. In order to secure the connection between the client and the messaging server, or between messaging servers, it is important to encrypt the connection. This is done via SSL. What transport security is available on the messaging server for JMS connections is determined by the messaging engine inbound transports defined for the application server where the messaging engine runs. The list of transports defined for an application server can be seen by opening the application server properties and clicking **Messaging engine inbound transports**. By default, both SSL and non-SSL enabled transports are defined as can be seen in Figure 13-3.

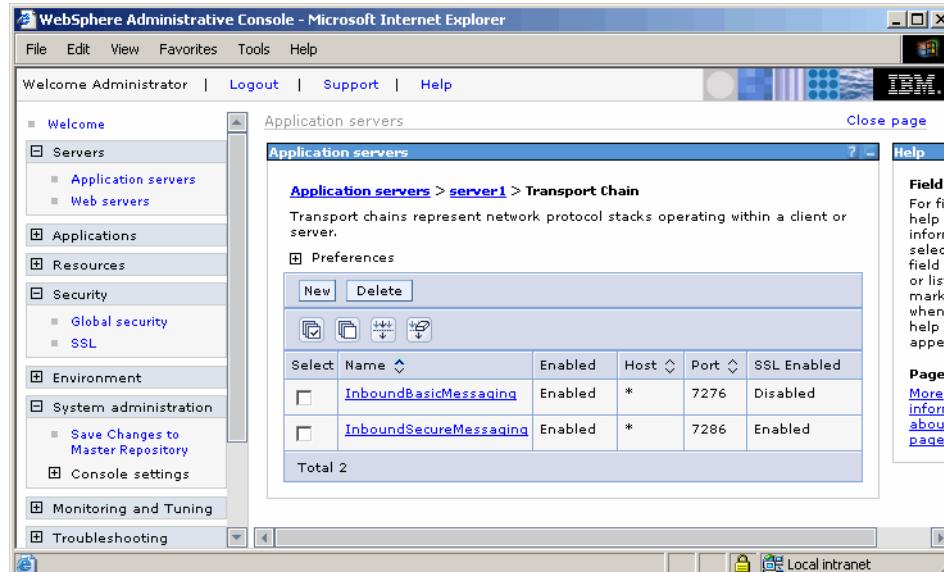


Figure 13-3 Inbound Transports

InboundSecureMessaging is the SSL-enabled transport. To require SSL connections, disable the *InboundBasicMessaging* transport chain. For communication between messaging engines on the Service Integration Bus, the default transport *InboundBasicMessaging* is used. To change transports, that is, require SSL, the transport name to use must be entered in the Inter-engine transport field on the bus's properties page as shown in Figure 13-4 on page 317. To use SSL, enter *InboundSecureMessaging* in the field.

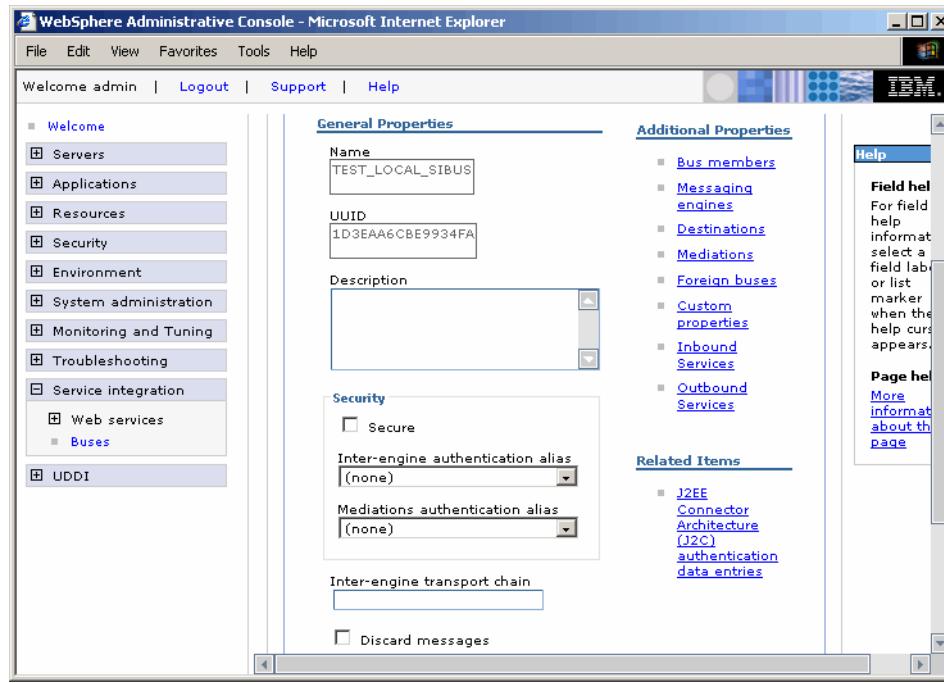


Figure 13-4 Inter-Engine transport

13.2.3 Administering Service Integration Bus security

Access to the Service Integration Bus is determined by user or group membership in the BusConnectorRole. When Global Security is enabled and the bus security has been enabled, access to the bus is checked when a connection to a bus resource is opened. By default, the special group AllAuthenticated is a member of this role. This allows all logged in users to access the Service Integration Bus. Any User or group can be assigned to this role as needed. Changing membership in the BusConnectorRole is accomplished using the wsadmin tool. Following are the wsadmin commands to view and modify membership in this role.

- ▶ List Users in BusConnector role:

```
$AdminTask listUsersInBusConnectorRole {-bus busname}
```

- ▶ List Groups in BusConnector role:

```
$AdminTask listGroupsInBusConnectorRole {-bus busname}
```

- ▶ Add user to BusConnectorRole:

```
$AdminTask addUserToBusConnectorRole {-bus busname -user username}
```

- ▶ Add group to BusConnectorRole:
`$AdminTask addGroupToBusConnectorRole {-bus busname -group groupname}`
- ▶ Remove a user from BusConnectorRole:
`$AdminTask removeUserFromBusConnectorRole {-bus busname -user username}`
- ▶ Remove a group from BusConnectorRole:
`$AdminTask removeGroupFromBusConnectorRole {-bus busname -group groupname}`

13.2.4 Administering destination security

Access to bus destinations is based on user or group membership in the various roles defined on the bus destination and the default roles defined on the Service Integration Bus. These two sets of roles are combined to determine if the action is authorized for the user. The section lists the commands used to modify user and group membership in the default roles defined on the Service Integration Bus for local bus destinations and the roles defined on the bus destination itself.

Default roles for bus destinations

The available role names for bus destinations are:

- ▶ Sender
- ▶ Receiver
- ▶ Browser
- ▶ Creator
- ▶ IdentityAdopter

The following commands work with the default roles for destinations on the Service Integration Bus.

- ▶ List users in default role:
`$AdminTask listUsersInDefaultRole {-bus busname -role roleName}`
- ▶ List groups in default role:
`$AdminTask listGroupsInDefaultRole {-bus busname -role roleName}`
- ▶ Add user to default role:
`$AdminTask addUserToDefaultRole {-bus busname -role roleName -user userName}`
- ▶ Add group to default role:
`$AdminTask addGroupToDefaultRole {-bus busname -role roleName -group groupName}`

- ▶ Remove user from default role:
`$AdminTask removeUserFromDefaultRole {-bus busname -role roleName -user userName}`
- ▶ Remove group from default role:
`$AdminTask removeGroupFromDefaultRole {-bus busname -role roleName -group groupName}`

Destination specific roles

The available role names for bus destinations are:

- ▶ Sender
- ▶ Receiver
- ▶ Browser
- ▶ Creator
- ▶ IdentityAdopter

The following commands set the roles for a specific local bus destination. For information about working with foreign destinations refer 13.3.5, “Administering foreign Service Integration Bus security” on page 326 and the WebSphere Application Server V6 Information Center.

- ▶ List users in role:
`$AdminTask listUsersInDestinationRole {-type destinationType -bus busname -destination destinationName -role roleName}`
- ▶ List groups in role:
`$AdminTask listGroupsInDestinationRole {-type destinationType -bus busname -destination destinationName -role roleName}`
- ▶ Add user to role:
`$AdminTask addUserToDestinationRole {-type destinationType -bus busname -destination destinationName -role roleName -user userName}`
- ▶ Add group to role:
`$AdminTask addGroupToDestinationRole {-type destinationType -bus busname -destination destinationName -role roleName -group groupName}`
- ▶ Remove user from role:
`$AdminTask removeUserFromDestinationRole {-type destinationType -bus busname -destination destinationName -role roleName -user userName}`
- ▶ Remove group from role:
`$AdminTask removeGroupFromDestinationRole {-type destinationType -bus busname -destination destinationName -role roleName -user userName}`

13.2.5 Administering topic space roles

The roles available for topic level security are:

- ▶ Sender
- ▶ Receiver
- ▶ IdentityAdopter

The following commands set roles for the topic space root topic, based on topic space root roles, set roles for individual topics and commands to change the inheritance of roles from parent topics.

- ▶ List users in TopicSpaceRootRole:

```
$AdminTask listUsersInTopicSpaceRootRole {-bus busname-topicSpace  
topicSpaceName -role roleName}
```

- ▶ List groups in TopicSpaceRootRole:

```
$AdminTask listGroupsInTopicSpaceRootRole {-bus busname-topicSpace  
topicSpaceName -role roleName}
```

- ▶ Add a user to TopicSpaceRootRole:

```
$AdminTask addUserToTopicSpaceRootRole {-bus busname-topicSpace  
topicSpaceName -role roleName -user userName}
```

- ▶ Add a group to TopicSpaceRootRole:

```
$AdminTask addGroupToTopicSpaceRootRole {-bus busname-topicSpace  
topicSpaceName -role roleName -group groupName}
```

- ▶ Remove a user from TopicSpaceRootRole:

```
$AdminTask removeUserFromTopicSpaceRootRole {-bus busname-topicSpace  
topicSpaceName -role roleName -user userName}
```

- ▶ Remove a group from TopicSpaceRootRole:

```
$AdminTask removeGroupFromTopicSpaceRootRole {-bus busname-topicSpace  
topicSpaceName -role roleName -group groupName}
```

- ▶ List users in topic role:

```
$AdminTask listUsersInTopicRole {-bus busname-topicSpace topicSpaceName  
-topic topicName -role roleName}
```

- ▶ List groups in topic role:

```
$AdminTask listGroupsInTopicRole {-bus busname-topicSpace topicSpaceName  
-topic topicName -role roleName}
```

- ▶ Add user to topic role:

```
$AdminTask addUserToTopicRole {-bus busname-topicSpace topicSpaceName  
-topic topicName -role roleName -user userName}
```

- ▶ Add group to topic role:
`$AdminTask addGroupToTopicRole {-bus busname-topicSpace topicSpaceName -topic topicName -role roleName -group groupName}`
- ▶ Remove user from topic role:
`$AdminTask removeUserFromTopicRole {-bus busname-topicSpace topicSpaceName -topic topicName -role roleName -user userName}`
- ▶ Remove group from topic role:
`$AdminTask removeGroupFromTopicRole {-bus busname-topicSpace topicSpaceName -topic topicName -role roleName -group groupName}`
- ▶ List sender role inheritance for a topic:
`$AdminTask listInheritSenderForTopic {-bus busname-topicSpace topicSpaceName -topic topicName}`
- ▶ List receiver role inheritance for a topic:
`$AdminTask listInheritReceiverForTopic {-bus busname-topicSpace topicSpaceName -topic topicName}`

13.3 Application server and WebSphere MQ

If the Embedded Messaging server does not meet your needs, WebSphere Application Server can use WebSphere MQ as a Messaging engine. Applications running on WebSphere Application Server V6 can communicate with WebSphere MQ using standard JMS API objects and methods.

13.3.1 WebSphere MQ messaging components

WebSphere Application Server applications can communicate with WebSphere MQ in two ways. WebSphere MQ can be connected to the Service Integration Bus as a foreign bus and the Service Integration Bus handles communication with WebSphere MQ, or applications can interact directly with WebSphere MQ. When connected to the Service Integration Bus, the WebSphere MQ Queue Manager appears to be another Messaging Engine on a foreign bus and all communication is handled via TCP/IP. If MQ is not linked to the Service Integration Bus, the applications interact directly with the WebSphere MQ server via inter-process communication if the WebSphere MQ server is local, or via TCP/IP. Regardless of which method is configured, the WebSphere application uses the same JMS API to interact with the JMS provider. Only the underlying implementation changes. Choosing the appropriate architecture depends on application and architectural requirements and will not be discussed here.

Direct communication with WebSphere MQ

Direct communication between WebSphere applications and IBM WebSphere MQ is handled via a pool of connection objects. See Figure 13-5 on page 323. The JMS Queue Connection Factory or JMS Topic Connection Factory specifies how to contact the Queue Manager and is administered by the WebSphere Administrative Console. Definition of this type of JMS resources is done by clicking **Resources** → **JMS Providers** → **WebSphere MQ**. In this configuration, WebSphere Application Server does not handle any of the authentication or authorization tasks other than passing a set of credentials to the WebSphere MQ server. The credentials can either be supplied by the application or the container where the request is made. Transport Security (Confidentiality) is specified on the Queue or Topic Connection Factory. The WebSphere MQ server handles all of the security tasks. IBM WebSphere Application Servers prior to V6 can only interact with WebSphere MQ in this manner.

Integrating MQ onto the Service Integration Bus

Integrating WebSphere MQ onto the Service Integration Bus follows the Service Oriented Architecture (SOA) model. The WebSphere MQ queue manager will appear to the local Service Integration Bus as a *Foreign Bus*. Communication parameters for the foreign bus are defined by an *MQLink* definition on the messaging engine used to communicate with MQ. Figure 13-5 on page 323 shows a WebSphere MQ queue manager connected to the Service Integration Bus via an MQ Link. Applications that need to send messages to MQ connect directly to their local messaging engine and put messages to a destination defined on the messaging engine hosting the MQ Link. The messaging engine that hosts the destination then takes care of transporting the messages to the appropriate MQ queue manager via the MQ Link. As far as the application is concerned the destination is local to the messaging engine.

Note: It is not possible to read from a queue on a WebSphere MQ server that has been integrated onto the Service Integration Bus. Messaging engines on the Service Integration Bus appear to MQ as another MQ Queue Manager. Therefore, queues on an MQ server are considered remote queues from the messaging engine's perspective and cannot be read from.

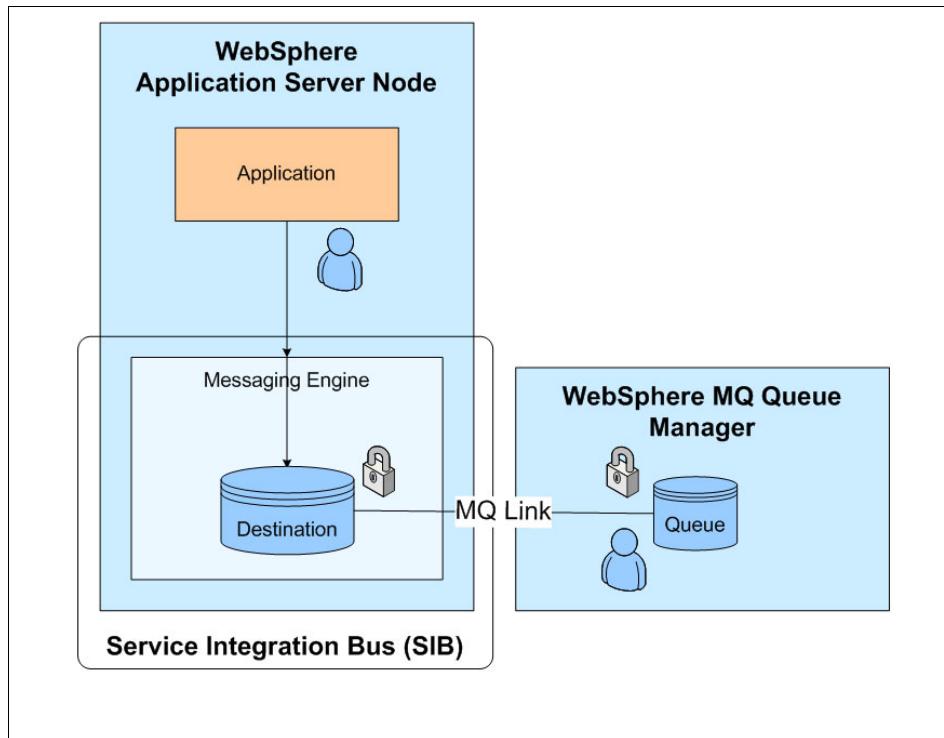


Figure 13-5 WebSphere MQ Service Integration Bus Integration

The following sections will discuss the three security points, Authentication, Authorization and Transport Security and what roles WebSphere Application Server and WebSphere MQ play.

13.3.2 Authentication

This section describes authentication for WebSphere Application Server and WebSphere MQ.

WebSphere Application Server

When WebSphere MQ is integrated onto the Service Integration Bus as a *Foreign Bus*, appearing as another messaging engine to WebSphere, WebSphere Application Server can handle a portion of the authentication. In order to access the Foreign Bus, applications connect to the local messaging engine and then put messages to either alias destinations or foreign destinations. In order to access any resources on the local bus, proper credentials must be supplied to the local messaging engine by either application

or the application container if security is enabled for the application server and bus. These credentials are validated against the global user registry.

For more information about local Service Integration Bus security see 13.2.2, “Embedded messaging security overview” on page 313.

WebSphere MQ

When a JMS client, WebSphere Application Server in this instance, connects to WebSphere MQ, the credentials supplied are checked against the local operating system user registry where the MQ server is installed.

13.3.3 Authorization

This section describes authorization for WebSphere Application Server and WebSphere MQ.

WebSphere Application Server

Authorization to access Service Integration Bus resources, including alias destinations, foreign destinations and connecting to the bus has already been discussed in Authorization in 13.2.2, “Embedded messaging security overview” on page 313. Alias destinations and foreign destinations are used to access resources on the MQ server, so access to these local resources is required.

WebSphere MQ

Access to WebSphere MQ resources is handled by the *Object Authority Manager (OAM)* on the WebSphere MQ server. The OAM is automatically enabled for each queue manager. If authorization checking is not required the OAM may be disabled.

The OAM maintains an Access Control List (ACL) for each WebSphere MQ object it is controlling access to. On UNIX systems only group IDs can appear in an ACL. This means that all members of a group have the same authority. On Windows, both user IDs and group IDs can appear in an ACL. This means that authorities can be granted to individual users as well as groups. The control command `setmqaut` grants and revokes authorities and is used to maintain the ACLs. Some of the authorizations that can be granted, or revoked, are:

- ▶ get
- ▶ browse
- ▶ put
- ▶ connect

In order to connect to the queue manager the user must have connect authorization. Once connected, when a request is made to a queue, process or namelist, the OAM checks the users authorization for that resource. For

example, if a user wants to put a message on the queue, they would need “put” authorization for the queue. You can find all details about WebSphere MQ messaging security in the IBM WebSphere MQ V5.3 Security product documentation.

13.3.4 Transport security

Communication between WebSphere Application Server and WebSphere MQ can be accomplished via TCP/IP, or if the queue manager and application server are on the same machine, via interprocess communication. When communication via a foreign bus, MQ Link is used, communication is always done via TCP/IP and the following two sub-sections apply.

If the applications communicates directly with the MQ server, not over the Service Integration Bus, then either communication path can be used. Transport security is not required when inter-process communication is used as transport of messages is done in memory. Transport level security, SSL parameters, for direct connections is defined on the JMS Queue or Topic Connection Factories. The SSL settings on the WebSphere side and MQ side must match.

WebSphere Application Server

When defining the MQ Link between a WebSphere Application Server messaging engine and the WebSphere MQ foreign bus, two MQ style channels, a sender and receiver, are defined in the local messaging engine. These two channels handle all communication between the messaging engine and the queue manager.

The sender channel, as the name implies, is used when messages are sent from the application server to MQ. The sender channel has an option for *Transport chain* which determines if SSL is used when transporting messages to MQ. The two default options for Transport chain are *OutboundBasicMQLink* and *OutboundSecureMQLink*. *OutboundSecureMQLink* uses SSL when connecting to WebSphere MQ.

When MQ needs to route a message to the embedded messaging engine on the application server, a connection is opened on the receiver channel. The port, or ports, that the local messaging engine accepts MQ Link inbound requests on are defined by the available transports in *WebSphere MQ link inbound transports* from the application server properties page. By default two transports are defined, *InboundBasicMQLink* and *InboundSecureMQLink*. *InboundSecureMQLink* is SSL enabled while *InboundBasicMQLink* is not. If SSL is required then disable the *InboundBasicMQLink* and restart the application server or node where the messaging engine is. Additional transports may be defined as needed.

Note: The default InboundBasicMQLink port is 5558. The default InboundSecureMQLink port is 5578.

Note: When communication with WebSphere MQ is done across the Service Integration Bus, the name of the Sender channel defined on the MQ link must match the name of a receiver channel on the MQ queue manager. The same is true for the sender channel defined on the MQ queue manager matching the name of the receiver channel on the MQ Link.

Important: It is strongly recommended that channel names not use lowercase letters as special consideration must be used when using lowercase letters and defining the channels on WebSphere MQ.

WebSphere MQ

Clients and queue managers communicate with WebSphere MQ over channels defined on the MQ server. When WebSphere Application Server applications use direct JMS connections to an MQ server, not over the Service Integration Bus, the applications appear as clients. When communication is done via the Service Integration Bus, the MQ Link appears as a queue manager to WebSphere MQ. Regardless of what method is used, communication is done over one or more channels. SSL properties on the channel allow for selection of which Cipher Spec to use and which clients, based on DN, to accept connections from. Enabling SSL on the channel is as simple as selecting the Cipher Spec and restarting the channel.

13.3.5 Administering foreign Service Integration Bus security

Security on the Service Integration Bus and resources within the Service Integration Bus will be checked if Global Security has been enabled on the application server and the Secure option as been checked for the Service Integration Bus. Assuming security is enabled for the Service Integration Bus, access to foreign destinations defined on the local Service Integration Bus first requires access to the local Service Integration Bus. The commands to modify the BusConnector role, the role that controls access to the Service Integration Bus, have already been detailed in 13.2.3, “Administering Service Integration Bus security” on page 317. Once the application accesses the local Service Integration Bus, additional checks are made to verify sender rights on the destination and foreign bus objects.

The following commands can be used to modify the Sender or IdentityAdopter roles on a foreign bus:

- ▶ List users in role for foreign bus

```
$AdminTask listUsersInForeignBusRole {-bus busname -foreignBus  
foreignBusName -role rolename}
```

- ▶ List groups in role for foreign bus:

```
$AdminTask listGroupsInForeignBusRole {-bus busname -foreignBus  
foreignBusName -role rolename}
```

- ▶ Add a user to a role for foreign bus:

```
$AdminTask addUserToForeignBusRole {-bus busname -foreignBus foreignBusName  
-role rolename -user username}
```

- ▶ Add a group to a role for a foreign bus:

```
$AdminTask addGroupToForeignBusRole {-bus busname -foreignBus  
foreignBusName -role rolename -group groupname}
```

- ▶ Remove a user from a role for foreign bus:

```
$AdminTask removeUserFromForeignBusRole {-bus busname -foreignBus  
foreignBusName -role rolename -user username}
```

- ▶ Remove a group from a role for a foreign bus:

```
$AdminTask removeGroupFromForeignBusRole {-bus busname -foreignBus  
foreignBusName -role rolename -group groupname}
```

The following commands can be used to modify the Sender and IdentityAdopter roles for the foreign destination:

- ▶ List users in role:

```
$AdminTask listUsersInDestinationRole {-type destinationType -bus busname  
-foreignBus foreignBusName -destination destinationName -role roleName}
```

- ▶ List groups in role:

```
$AdminTask listGroupsInDestinationRole {-type destinationType -bus busname  
-foreignBus foreignBusName -destination destinationName -role roleName}
```

- ▶ Add user to role:

```
$AdminTask addUserToDestinationRole {-type destinationType -bus busname  
-foreignBus foreignBusName -destination destinationName -role roleName  
-user userName}
```

- ▶ Add group to role:

```
$AdminTask addGroupToDestinationRole {-type destinationType -bus busname  
-foreignBus foreignBusName -destination destinationName -role roleName  
-group groupName}
```

- ▶ Remove user from role:

```
$AdminTask removeUserFromDestinationRole {-type destinationType -bus busname -foreignBus foreignBusName -destination destinationName -role roleName -user userName}
```
- ▶ Remove group from role:

```
$AdminTask removeGroupFromDestinationRole {-type destinationType -bus busname --foreignBus foreignBusName destination destinationName -role roleName -user userName}
```

13.3.6 Administering WebSphere MQ security

Access to WebSphere MQ objects is controlled by the OAM. To **setmqaut** command is used to grant and revoke authorities and maintain the ACL contained in the OAM. Only a user with administration authority can run this command. Running the **setmqaut** command without parameters will display the command usage as follows:

```
setmqaut -m QMgrName [-n ObjName] -t ObjType [-p Principal | -g Group] [-s ServiceName] Authorizations
```

Some of the authorizations that can be granted, or revoked, are:

- ▶ get
- ▶ browse
- ▶ put
- ▶ connect

Note: Prepend the authorization with + to grant and - to revoke. For example, use +connect to grant connect authorization and -get to revoke get authorization.

Multiple authorizations can be specified at one time by separating them with a space. For example, +browse -get - put

Note: In a Unix environment, authorizations can only be granted on groups.

For example, to grant browse rights to user janedoe on a queue name default and queue manager named QM_klchm8p the following command would be executed and result shown.

```
C:\PROGRA~1>setmqaut -m QM_klchm8p -n default -t queue -p janedoe +browse  
The setmqaut command completed successfully.
```

The **dspmqaut** command can be used to display authorizations from the OAM. For an in depth details on administering MQ security see the IBM WebSphere MQ V5.3 Security product documentation.

13.3.7 Connecting WebSphere MQ to the Service Integration Bus

In order to connect WebSphere MQ to the Service Integration Bus, the local Service Integration Bus must be defined. Refer to “Configuring Embedded Messaging” on page 393 for the steps to define the Service Integration Bus and add an application server to the bus. Once completed, the following steps detail defining the foreign bus and the MQ Link for use in communicating with WebSphere MQ.

Defining the foreign bus

1. Open the Administrative Console and log in.
2. Click + next to Service integration.
3. Click **Buses** to display the list of defined buses.
4. Click the previously defined bus where the foreign bus will be defined to display the bus properties page.
5. Click **Foreign buses** to display the list of foreign buses defined on this Service Integration Bus
6. Click **New** to add a new foreign bus.
7. Enter a name and description for the foreign bus.

Note: If foreign destinations will be used to communicate with MQ queues, enter the name of the MQ queue manager here. Otherwise any descriptive name will be fine.

8. If applications are not allowed to send messages to the foreign bus, uncheck the **Send allowed** check box.
9. Click **Next**.
10. Select the Routing type. For communication to WebSphere MQ from the local bus via an MQ link select **Direct, WebSphere MQ link**.
11. Click **Next**.
12. Enter the name to use to authenticate inbound message flows in the Inbound user ID field.

Note: All messages received from the foreign bus will appear to come from the id entered in the Inbound user ID field. This is important when the sender of the message is not authorized to access the Service Integration Bus or put messages to objects on the local Service Integration Bus.

When a message enters the bus, the user ID that is stored in the message is checked against the local security. Initially the user ID is the user ID assigned to the message by the sending bus. That user ID may not have access to the local Service Integration Bus or resources, and may not even be known to the local User Registry. By entering a locally known user ID in the Inbound user ID field, the message can transmit into the local Service Integration Bus and all message coming from the foreign bus will appear to come from this local id.

13. Enter the name to use to authenticate outbound message flows in the Outbound user ID field.

Note: All messages sent to the foreign bus will appear to come from the id entered in the Outbound user ID field. On WebSphere MQ, this value will be in the UserIdentifier field of the message context.

When a message enters a secure foreign bus, the user ID that is stored in the message. The user ID is initially set to the user ID of the message sender. This may not be appropriate, by entering a user ID here all messages will appear to come from the new user ID.

14. Click **Next**.
15. Click **Finish** to return to the foreign bus list page.
16. Save the configuration for WebSphere.

Defining the MQ link

1. Open the Administrative Console and login.
2. Click + to expand Service integration.
3. Click **Buses** to display the Service Integration Bus list page.
4. Click the Service Integration Bus where the foreign bus for MQ is defined to display the Service Integration Bus properties page.
5. Click **Messaging engines** to display a list of the messaging engines on the Service Integration Bus.
6. Click the messaging engine where the MQ link will be defined.

7. Click **WebSphere MQ links**.

Note: *WebSphere MQ client links* are used to define a MQ link that will make the messaging engine appear as a queue manager to JMS clients connecting to it. Security for client links is handled by the roles discussed for embedded messaging communication in 13.2, “Default JMS provider” on page 310.

8. Click **New** to create a new WebSphere MQ link.
9. Enter a name and description for the link.
10. Select the foreign bus from the *Foreign bus name* list. This is the foreign bus defined above.
11. Enter a queue manager name. This is the name that the MQ link will appear to the MQ queue manager as and it is recommended not to use lowercase letters. This name will be used when defining the sender channel on the WebSphere MQ queue manager.
12. Modify other properties as needed.
13. Click **Next**.
14. Enter a name for the sender channel. This must match a receiver channel defined on the MQ server.
15. Enter the hostname and port of the MQ server queue manager.
16. Select the Transport chain to use.

Note: The transport chain determines the transport level security. Select *OutboundSecureMQLink* to use SSL when connecting to WebSphere MQ.

17. Modify other properties as needed.
18. Click **Next**.
19. Enter a name for the receiver channel. This must match a sender channel defined on the WebSphere MQ queue manager.
20. Modify other properties as needed.
21. Click **Next**.
22. Click **Finish** to return to the WebSphere MQ links list page.
23. Save the configuration for WebSphere.

Defining a foreign destination

In order for an application to put a message on a queue on the MQ queue manager the local Service Integration Bus needs a Foreign destination definition defining the foreign bus, MQ queue manager, and destination, queue. The following steps detail how to define the foreign destination:

1. Open the Administrative Console and login.
2. Click + to expand Service integration.
3. Click **Buses** to display the Service Integration Bus list page.
4. Click the Service Integration Bus where the foreign destination will be defined to display the Service Integration Bus properties page.
5. Click **Destinations** to display the destinations list page.
6. Click **New** to create a new destination.
7. Select **Foreign** as the destination type and click **Next**.
8. Enter an Identifier for the destination.

Note: The Identifier must match the name of the destination on the foreign bus. In the case of WebSphere MQ, this is the name of the queue defined on the queue manager.

9. Enter a description for the destination.
10. Select the foreign bus from the bus list where this destination links to. For an MQ queue this is the name of the foreign bus defined for the MQ link.
11. Modify other properties as needed.
12. Click **Next**.
13. Click **Finish** to return to the Destinations list page.
14. Save the configuration for WebSphere.

Defining the JMS objects

Defining a JMS Connection Factory and JMS queue for queue destinations on the local Service Integration Bus has already been covered in , “Configuring Embedded Messaging” on page 393. When sending messages to a Foreign destination, the application connects to the local messaging engine. The connection factory used to connect to the local Service Integration Bus messaging engine can be reused to here. The steps to configure the JMS Connection Factory can be found in “Define the JMS Objects” on page 400.

The steps to create a JMS queue destination for a foreign destination are identical to the steps detailed in “Defining the JMS objects” on page 332 with one

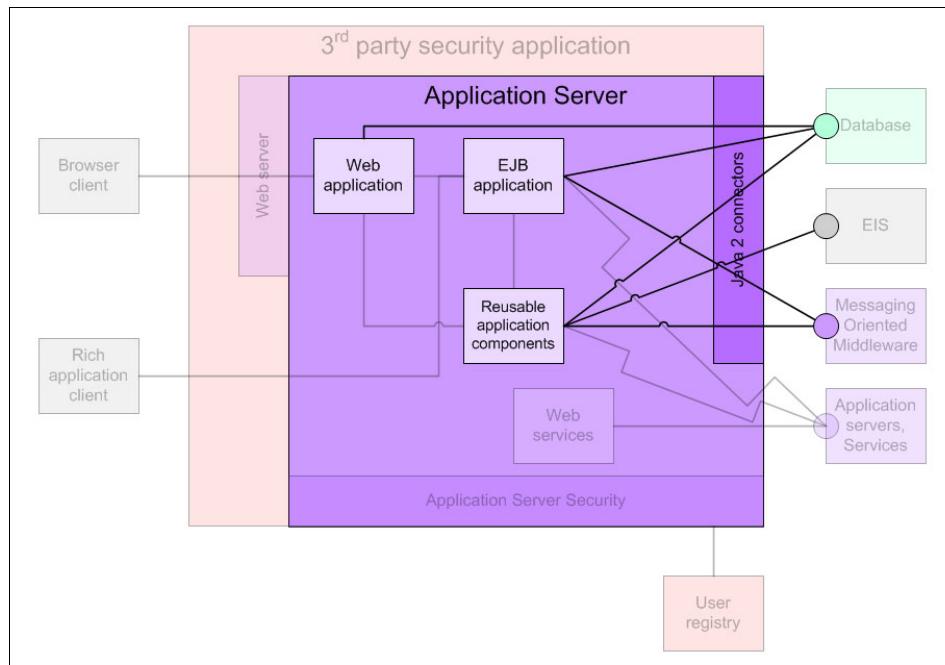
difference. Rather than selecting the local Service Integration Bus name for the Bus name, select the name of the foreign bus used to connect to WebSphere MQ. Once the page refreshes select the Foreign destination from the Queue name field. All other steps are the same.

13.4 Sample application

Refer to “Configuring Embedded Messaging” on page 393 for more information about configuring messaging for applications.

You can find the details of a messaging sample application in “Sample application for messaging” on page 402.

Java 2 Connector security



14.1 What is the J2EE Connector Architecture?

The J2EE Connector Architecture specifies a standard architecture that allows J2EE applications to connect to EIS systems. The architecture defines a set of contracts that EIS vendors and application server vendors code to. These contracts specify standard APIs to which the resource adapters must adhere and services that the system must provide to support the resource adapter. These system level services include transaction management, connection management and security management, among others. EIS vendors coding their resource adapters to the connector API take advantage of the system level services on the J2EE server on which they are running. Applications are then free to take advantage of the resource adapter or connector.

Once coded to the API, the resource adapter is plugged into an application server that supports the J2EE Connector Architecture. WebSphere Application Server V6 supports the J2EE Connector Architecture version 1.5 specification. After the resource adapter is installed, applications running on the application server can access the EIS system without having to handle the system level services; these are called outbound calls. Applications utilize the Common Client Interface (CCI) for EIS access. This interface provides a common API through which applications can access EISs.

Starting with the version 1.5 specification, the EIS application can make calls into the application server to access application components and perform tasks. Calls initiated by EIS to WebSphere Application Server are called inbound calls.

The specification itself is beyond the scope of this book, but the architecture defines a standard set of APIs and services for scalable, transaction-oriented, secure connections to backend EIS such as ERP systems, databases, transaction processing and messaging systems. Figure 14-1 on page 337 shows a high-level representation of the J2EE Architecture.

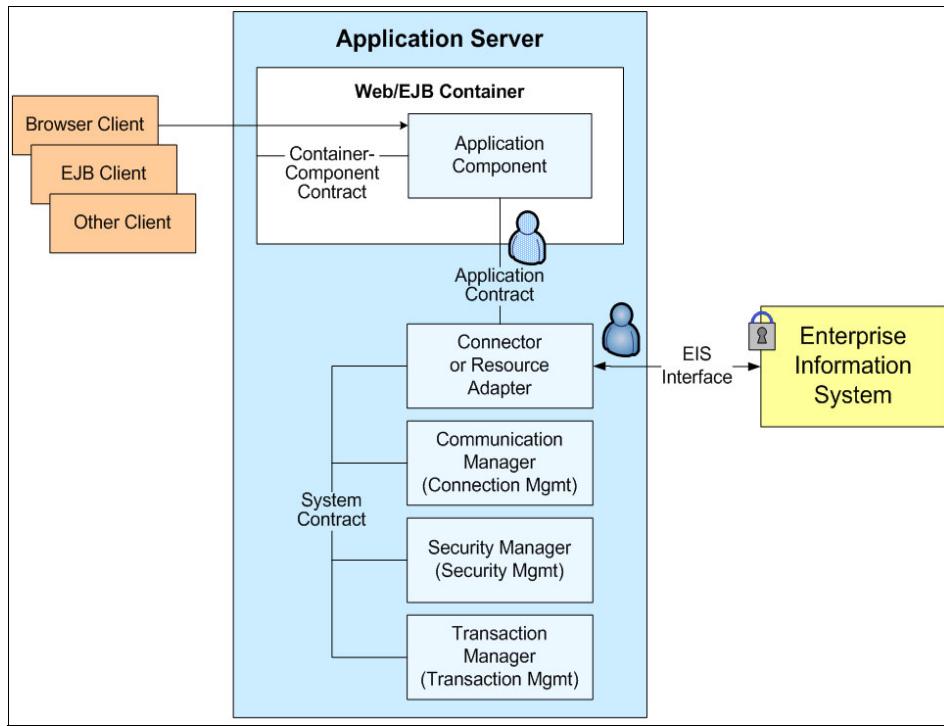


Figure 14-1 J2EE Connector Architecture

The full J2EE Connector Architecture can be downloaded from Sun Microsystems at:

<http://java.sun.com/j2ee/connector/>

14.2 Securing the J2EE Connector

By their nature, EIS resources generally require a high level of security to protect the information they contain from unauthorized access. In general, this means that in order to connect to an EIS system, the user must supply a proper set of credentials, usually a user ID and password, before gaining access to the EIS backend. When accessing an EIS backend, the request for access flows from the application to the resource adapter and then to the EIS. In a secure environment, when the request is made to the adapter, the proper credentials are sent to the adapter when a connection is requested. The adapter then uses the credentials to connect to the EIS system and perform the requested action(s).

J2EE Connectors follow the J2EE security model. Access to EIS resources takes place under the security context of a resource principal. Where the security

context comes from depends on the security model in use at the time of access to the resource adapter. The two security models are component-managed authentication (also called application-managed authentication) and container-managed authentication. Each will be discussed below.

Note: You can also find the component-managed authentication referred to as Per Connection Factory. Mostly you will see this naming in the Rational Application Developer.

Component-managed authentication

In the case of component-managed authentication, the application component accessing the resource or adapter is responsible for programmatically supplying the credentials, or WebSphere can supply a default component-managed authentication alias, if available. After obtaining the connection factory for the resource from JNDI, the application component creates a connection to the resource using the create method on the connection factory supplying the credentials. If no credentials are supplied when creating a connection and a component-managed authentication alias has been specified on the J2C connection factory, the credentials from the authentication alias will be used. Assuming the credentials are valid, future requests using the same connection will use the same credentials.

Note: Component-managed authentication is specified by setting the res-auth entry in the Deployment Descriptor for the Resource Reference to Application. The steps to define a resource reference, using a JDBC data source as an example, can be found in 15.3.1, “Defining the Resource Reference” on page 355.

Creating a sample EIS resource adapter is beyond the scope of this book. Sample code for looking up a resource adapter connection factory and connecting to the resource can be found in Example 14-1 on page 339. The code assumes that a Resource Reference has been defined and named EISResourceName and maps to a J2EE Resource Adapter connection factory.

The basic steps are as follows:

1. Get initial JNDI context.
2. Look up the connection factory for the resource adapter.
3. Create a ConnectionSpec object holding credentials.
4. Obtain the Connection Object from the Connection Factory by supplying the ConnectionSpec object.

Once a connection is obtained using the credentials specified in the ConnectionSpec object, all future interactions, through interaction objects, will carry the user credentials and the EIS will fulfill the request or deny it based on the Authorization properties in EIS.

Example 14-1 Get Resource Connection

```
try
{
    Context ic = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory)
ic.lookup("java:comp/env/EISResourceName");
    try {
        //Use the following if res-auth=Application
        //This is for Component Managed Authentication with
        //no JAAS Authentication Alias set on the Connection Factory
        WSConnectionSpecImpl conSpec = new WSConnectionSpecImpl();
        conSpec.setUserName("username"); // replace the username with the value
        conSpec.setPassword("password"); // replace the password with the value
        Connection con = cf.getConnection(conSpec);
        //Use the following if res-auth=Container
        //This is for Container Managed Authentication
        //Connection con = cf.getConnection();
    } catch (ResourceException re) {
        System.out.println(re.toString());
    }
}
catch(NamingException ne) {
    System.out.println(ne.toString());
}
```

Container-managed authentication

Container-managed authentication removes the requirement that the component programmatically supply the credentials for accessing the EIS. Instead of calling the getConnection() method with a ConnectionSpec object, getConnection() is called with no arguments. See Example 14-1 for sample code. The authentication credentials used for connecting to the EIS are then supplied by the Web container, the application container or the EJB container, depending on where the resource is accessed from. WebSphere Application Server V6 supports the JAAS specification, so the credentials for accessing the EIS can be mapped from any of the configured JAAS Authentication login modules, including any custom JAAS Authentication login module.

When defining the Resource Reference in the deployment descriptor, either Web application deployment descriptor or EJB deployment descriptor, once Authentication is set to Container and the WebSphere Bindings JNDI Name has

been entered, three options become available for the JAAS Login Configuration. The three options are explained below.

Container-managed authentication (deprecated)

This option uses the container-managed authentication settings that are defined for the resource's connection factory. The credentials can come from a JAAS Authentication Alias when using the DefaultPrincipalMapping Mapping-configuration alias setting, or be mapped from another JAAS Authentication login module. Any application that can get the resource's connection factory from JNDI will be able to access the EIS. This creates a security exposure where unauthorized applications can gain access to the EIS.

Note: Selecting this option and specifying DefaultPrincipalMapping and selecting a JAAS Authentication Alias when defining the resource's Connection Factory provides the same functionality as WebSphere Application Server V5.

This is no longer the recommended method. The Use Default Method option is recommended and discussed next.

Use Default method

The Use Default Method setting behaves very similarly to container-managed authentication using the DefaultPrincipalMapping option. A JAAS authentication alias is linked to the Connection Factory and all container-managed authentication requests using the resource reference use the credentials from the alias when connecting to the EIS. The difference is that the linking from the JAAS authentication alias to Connection Factory is done at the resource reference level within the application. This alleviates a security exposure by limiting the scope of the credentials to the application defining the resource reference. All other applications would need to supply their own credentials when accessing the Connection Factory directly from JNDI. This is the recommended method for mapping JAAS authentication aliases to Connection Factories.

Use Custom Login Configuration

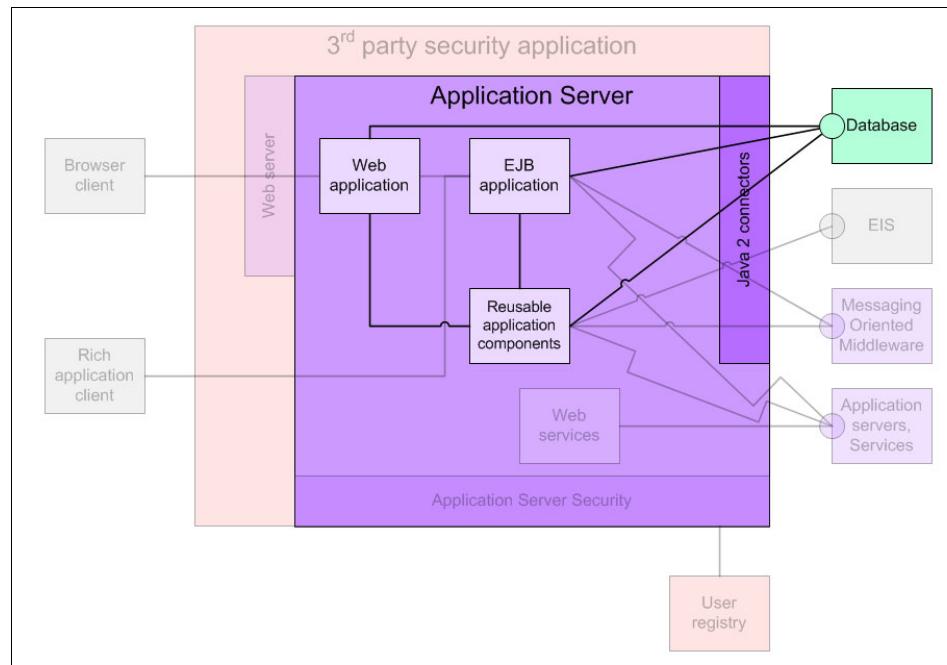
This option allows for the use of any defined JAAS authentication module to be used. Enter the name of the JAAS authentication modules as it is defined in **Security → Global Security → JAAS Configuration → Application Logins** and specify any parameters needed. When a connection to the resource is required, the specified module will be used to obtain the credentials that will be passed to the connector.

14.3 Additional information

Additional information about the J2EE Connector Architecture Specification can be found at:

<http://java.sun.com/j2ee/connector>

Securing the database connection



15.1 Database security

Database security is typically broken down into two areas: first, securing the connection between the client and server, and secondly, access control.

Securing the connection generally is concerned with using encryption between the database client and server so that others on the network cannot sniff data as it flows across the network. The encryption can be as simple as using SSL between client and server, or some other proprietary encryption scheme.

15.1.1 Securing the connection

The IBM DB2 Universal Database allows for the specification of one or more authentication types. These authentication types specify how and where the authentication of the user will be verified and what type of encryption is required. The authentication types allowed between DB2 UDB V8.1 clients and servers are shown in Table 15-1, along with what encryption is performed. See the IBM DB2 Information Center article titled “Authentication methods” for further information about the authentication types and how to set them.

Table 15-1 DB2 V8.1 authentication types

DB2 version	Authentication Type	Where Specified	Encryption
8.1	Server	Client or Server	None. User ID, password, and data sent unencrypted.
8.1	Client	Client or Server	None. Only server trusts client ID so password is not sent to server.
8.1	Server_Encrypt	Client or Server	Password encrypted when sent to server only.
8.2	Data_Encrypt	Client or Server	Same as Server_Encrypt and the following encrypted also: SQL Statement SQL program variable data Output data from server processing SQL Answer data set from a query LOB data streaming SQLDA descriptors
8.2	Data_Encrypt_CMP	Server	Same as Data_Encrypt except for clients that don't support it, they will fall back to Server_Encrypt.
8.2	Kerberos	Client or Server	

DB2 version	Authentication Type	Where Specified	Encryption
8.2	Kerberos_Encrypt	Server	

Note: When explicitly cataloging a database with a specific authentication type, it is important to make sure that the server is set to handle that encryption type. If, for example, the client catalogs the database using Kerberos and the server is set to Server_Encrypt, an SQL error will result and connection will not complete.

IBM currently ships two JDBC data source provider implementations with DB2 Universal Database V8.1. These are the *DB2 Legacy CLI-based Type 2 Provider* and the new *DB2 Universal JDBC Provider*. The Universal provider is both a JDBC type 2 and JDBC type 4 driver, while the Legacy provider is a type 2 driver only. Encryption of the JDBC connection to the DB2 server depends on which type of JDBC driver is used. Figure 15-1 on page 346 contains a diagram of the type 2 and type 4 driver communication paths. The security implications of each type of driver will be discussed below.

In order to use the Legacy provider, the following tasks must be accomplished on each WebSphere Application Server:

1. Install the DB2 client.
2. Catalog the remote node(s) containing database that are needed.
3. Catalog the database from the remote nodes, specifying the correct Authentication type.
4. Set up the DB2 environment for the user running WebSphere.
5. Set the *DB2 JDBC Driver Path* WebSphere variable to point to the sqllib/java directory for the DB2 client instance.

To use the Universal driver, the following tasks need to be completed:

1. Install/Copy the db2jcc JAR files to the WebSphere Application Server.
2. Set the *DB2 Universal JDBC Driver Path* WebSphere variable to point to the directory containing the jar files.

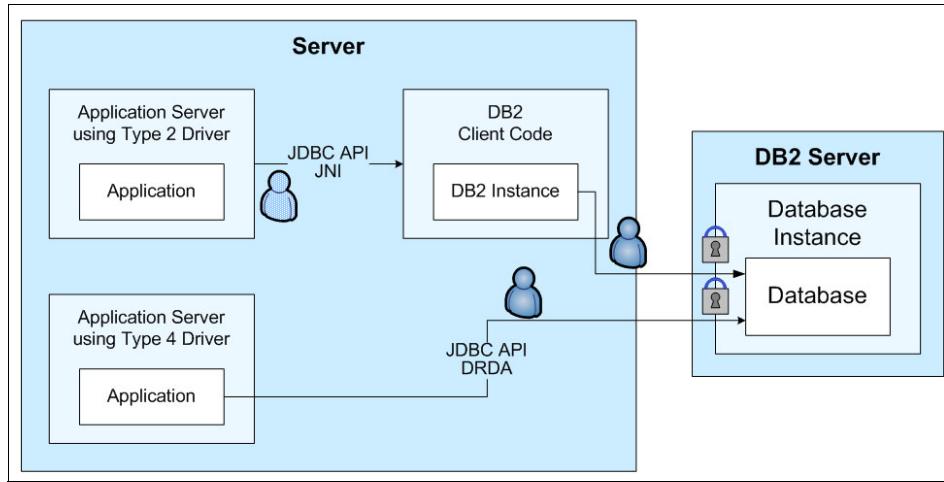


Figure 15-1 IBM DB2 Universal Database JDBC drivers

JDBC type 2 driver

The type 2 driver uses the locally installed native DB2 client libraries. By making JNI calls through the local DB2 instance, as shown in Figure 15-1, the Java application can access databases that have been cataloged on the system. Security over the JNI calls is not required. Security between the DB2 client instance and the DB2 server is determined by the security settings specified during cataloging of the database at the client and the DB2 server's authentication setting. The following example shows how to catalog the sample database using the `db2 catalog` command and setting the Authentication mechanism to SERVER_ENCRYPT. SERVER_ENCRYPT specifies that authentication of the user credentials is done at the server and only the password is encrypted between client and server. All requests and data flow unencrypted, in the DB2 binary format, between client and server.

Example 15-1 Sample DB2 catalog command

```
db2 catalog database sample at node db2node_name AUTHENTICATION SERVER_ENCRYPT
```

JDBC type 4 driver

The JDBC type 4 driver is a pure Java, client-side implementation used to connect to an IBM DB2 Universal Database server. The type 4 driver uses the DRDA, *Distributed Relational Database Architecture*, protocol to connect directly to the database server. When using the type 4 driver, the driver gets a list of the authentication types the server will accept and will use one of those if possible. If the client and server have no allowed authentication types in common, an exception will be thrown and connection will fail. The authentication

type chosen determines the level of security between the JDBC client and the DB2 server. Allowing the client and server to negotiate the authentication type is the recommended way of handling authentication and encryption. It is also possible to specify programmatically which authentication type the type 4 driver can use, by setting the *securityMechanism* custom property on the data source's connection pool.

Note: When using the type 4 driver, a local install of the DB2 client code is not required. Only the db2jcc JAR files will need to be copied to the WebSphere Application Server and the WebSphere environment variable UNIVERSAL_JDBC_DRIVER_PATH pointed to the appropriate directory. The JAR files can be found on the DB2 server in the <DB2_root>/java directory.

15.1.2 Securing access to database data

The second important area of database security, access control, is generally broken down into two topics, authentication and authorization. Authentication, validating a user is who they say they are, and authorization, validating access to particular data within the database, are processes handled by the database server. When a user supplies his credentials, usually a user ID and password, to the database server during a connect request, the database engine validates those credentials. Once the credentials are validated, the database engine then checks to see if the user is authorized to perform the action requested, such as connecting to the database or interacting with data in the database.

The authentication credentials that are used by WebSphere Application Server to access a database can either be programmatically supplied or provided by the various application server containers. When the database is accessed from a resource reference with component/application managed authentication defined, the credentials are either supplied programmatically during the connect request or from a component-managed authentication alias defined on the JDBC connection factory. For container-managed authentication access, the container, Web, EJB or client, supplies the credentials to the connection factory. Where those credentials come from is defined by a JAAS login module. For further information regarding these two authentication types and JAAS, refer to Chapter 4, "JAAS for authentication and authorization in WebSphere" on page 49.

Note: WebSphere Application Server V6 can provide data sources for use in J2EE 1.2 and J2EE 1.3 applications. Programmatically, applications using the JDBC API will behave the same. The underlying architecture of the data sources are very different. J2EE 1.3 data sources use the J2EE 1.3 connector architecture as discussed in Chapter 14, “Java 2 Connector security” on page 335 and can be accessed using the JDBC api or the WebSphere Java 2 Connector API.

Each database engine has a method for granting, revoking and storing authentication and authorization information. For example, IBM DB2 Universal Database uses the **grant** and **revoke** commands to specify authorization information. If using a database engine other than DB2, refer to the documentation specific to your installation. Some general guidelines to consider when implementing database security are given below.

- ▶ Require users to supply credentials to access the database. This generally requires revocation of “public” or anonymous access.
- ▶ Do not share user IDs and passwords among users.
- ▶ Specify authorization to tables, stored procedures, functions, etc. using groups rather than user IDs. While not a true security requirement, this makes administration a lot easier.
- ▶ Grant only the minimum rights required for a user/group to accomplish the tasks assigned to them.

15.2 Defining a JDBC data source

In order for an application to access a database, a JDBC connection needs to be established. WebSphere Application Server V6 provides the facilities to define JDBC data sources, connection factories, and register them in the JNDI namespace. The JDBC connection can then be obtained by retrieving the connection factory from JNDI and calling one of the connect methods. These connection factories are defined at the Cell, Node or Server scope. An application can obtain a data source defined in the same scope as the application or higher.

WebSphere Application Server V6 and IBM Rational Application Developer now support the definition of JDBC data sources in the EAR deployment descriptor. These data factories are defined in the Application scope.

Note: It is also possible to programmatically load the JDBC driver and connect to the database, by passing the application server and JNDI. However, this is not recommended due to the additional services (transaction services and connection pooling) that are provided by the application server.

This section will define the basic steps needed to define a JDBC data source via the Administration Console and via the deployment descriptor.

15.2.1 Defining a JDBC data source using the Administrative Console

Define a JDBC resource using the Administrative Console by doing the following:

1. Make sure that WebSphere Application Server or RAD's *WebSphere Application Server V6 Test Environment* (WTE) is started.
2. Open the Administrator Console and log in.

Note: The URL for the Administrator Console is
`http://servername:9060/admin` where *servername* is the name of the server running WebSphere Application Server V6.

If you are targeting the RAD WTE, you can right-click the server in the Servers view and select **Run Administration Console**.

3. Click + next to Resources to expand the list.
4. Click **JDBC Providers**. You will see a window similar to Figure 15-2 on page 350.

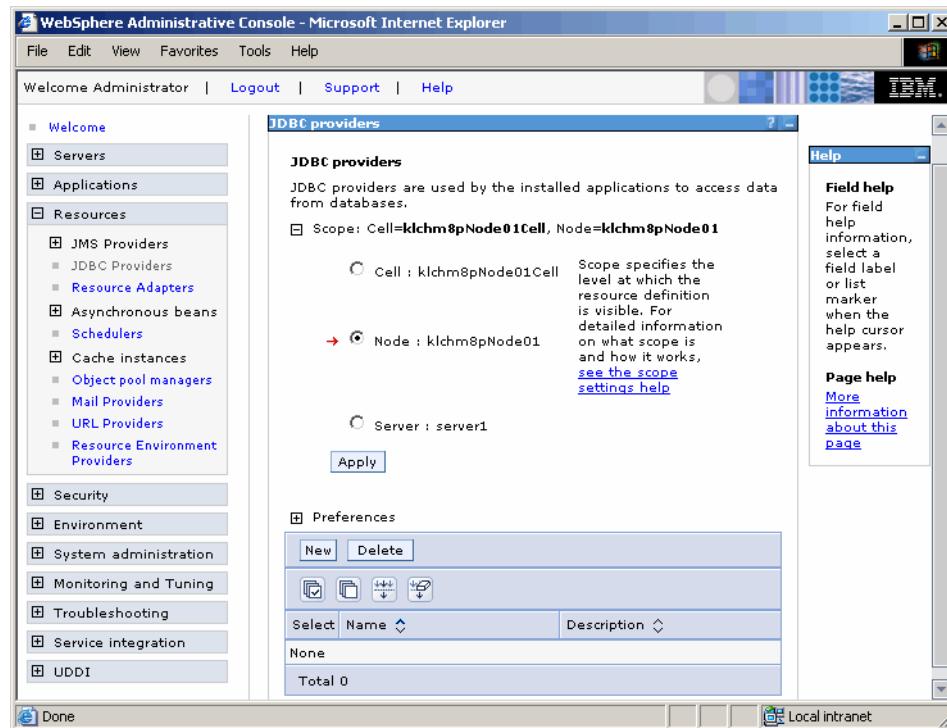


Figure 15-2 JDBC Provider List

5. Specify the Scope in which you want to create the JDBC Resource.

Note: In a single server environment, Node is generally the correct scope. In a Network Deployment environment, Cell is generally the correct scope.

6. Scroll down and click an already defined JDBC Provider that matches the type of Database server and XA type required for the data source, if one exists. The JDBC Provider details will be displayed as shown in Figure 15-3 on page 351.

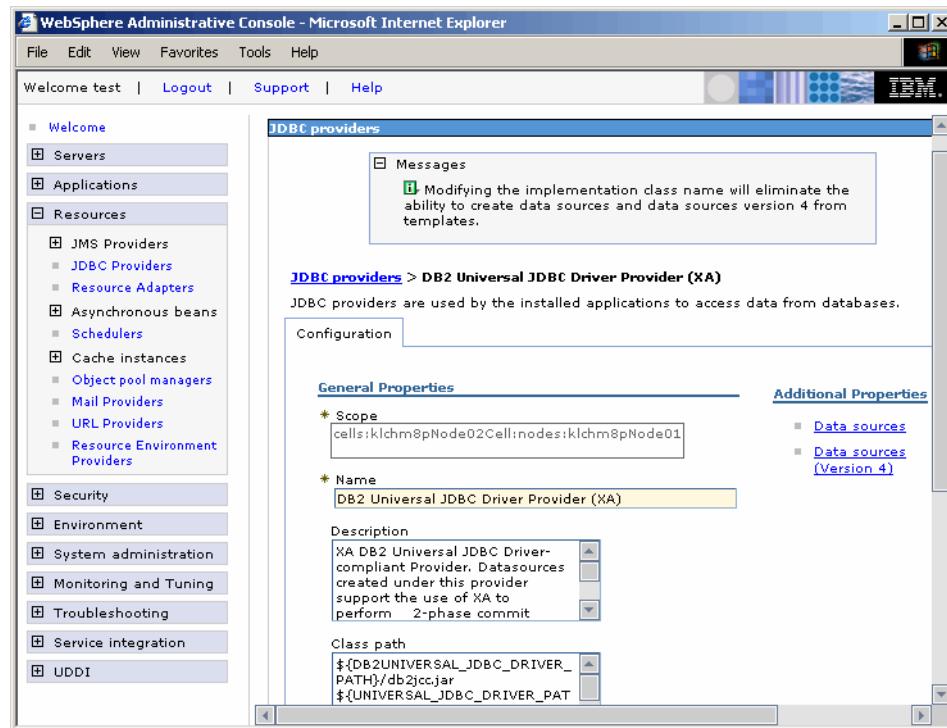


Figure 15-3 Data source provider details

7. If an appropriate JDBC Provider is not already defined, do the following to create a new JDBC Provider:
 - a. Click **New**.
 - b. Select the database type. For the sample application, select **DB2**.
 - c. Select the provider type. For DB2 running on a distributed platform, Unix, Linux or Windows, select either **DB2 Universal JDBC Driver** or **DB2 Legacy CLI-based Type 2 JDBC Driver**.
 - Note:** The DB2 Universal JDBC Driver is the new JDBC driver that ships with DB2 since V8. The driver is both a type 2 and type 4 JDBC driver. The implementation used is controlled by the Driver type setting on the data source.
 - d. Select the required Implementation Type. If your application requires a two- phase commit then select the **XA data source**, otherwise choose **Connection pool data source**.

Note: If a two-phase commit is not required by the application, do not use the XA data source, because it can cause additional locks to be placed on data in the database; instead, use **Connection Pool DataSource**.

- e. Click **Next** to display the new JDBC provider page.
- f. In the Name field, enter a name for the provider. For example, enter IBM DB2 Universal XA Provider.
- g. Click **OK** to create the provider and return to the JDBC providers page.
- h. From the list of providers, click the newly created provider.
8. On the right side of the page, click **Data sources** to work with J2EE 1.3 data sources, or click **Data sources (Version 4)** to work with J2EE 1.2 data sources. Clicking **Data sources** will cause the Data sources page to be displayed.
9. Click **New** to create a new data source. The new data source page will be displayed.
10. For the new data source, the appropriate values will need to be filled in. In general the following fields should be populated:
 - Name
 - JNDI Name
 - Description
 - Database name
 - Component-managed authentication alias (if needed)
 - Driver type (If using DB2 Universal JDBC Driver)
 - Servername (Should not fill this in when using DB2 type 2 drivers)
 - Port number
11. Click **OK** to create the data source and return to the Data sources page.
12. Save the configuration for WebSphere.

15.2.2 Defining a JDBC data source in EAR

Define a JDBC resource in the EAR deployment descriptor by doing the following:

1. Open the Deployment Descriptor for the EAR in Rational Application Developer.
2. Click the **Deployment** tab. The Deployment tab will look similar to that shown in Figure 15-4 on page 353.

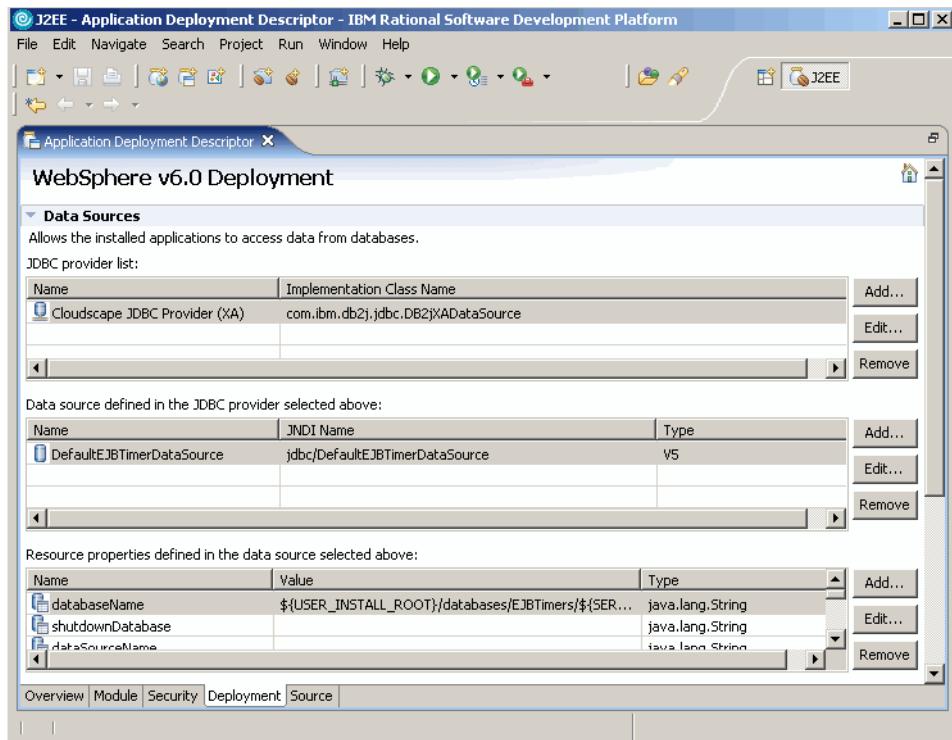


Figure 15-4 EAR Deployment Descriptort

3. Click **Add** next to the JDBC provider list to add a new JDBC provider.
4. A new window is displayed, select the database type and JDBC provider type. For example, select **IBM DB2** as the database type and **DB2 Universal JDBC Provider (XA)** for the JDBC provider type.
5. Click **Next**.
6. Enter name for the new JDBC provider. For example, enter **IBM DB2 Universal XA Provider**.
7. Click **Next** to define more properties for the JDBC provider if available, otherwise click **Finish** to return to the Deployment Descriptor Deployment tab.
8. Select the JDBC provider from the provider list under which you want to define the data source.
9. Click **Add** next to the Data source list to add a new data source.
10. A new window displays; select the type of JDBC provider, for example, **DB2 Universal JDBC Provider (XA)**.

11. Select the data source type to create. For example, select **Version 5.0 data source** for J2EE 1.3 applications.
12. Click **Next**. The Modify Data Source window will be displayed as shown in Figure 15-5.

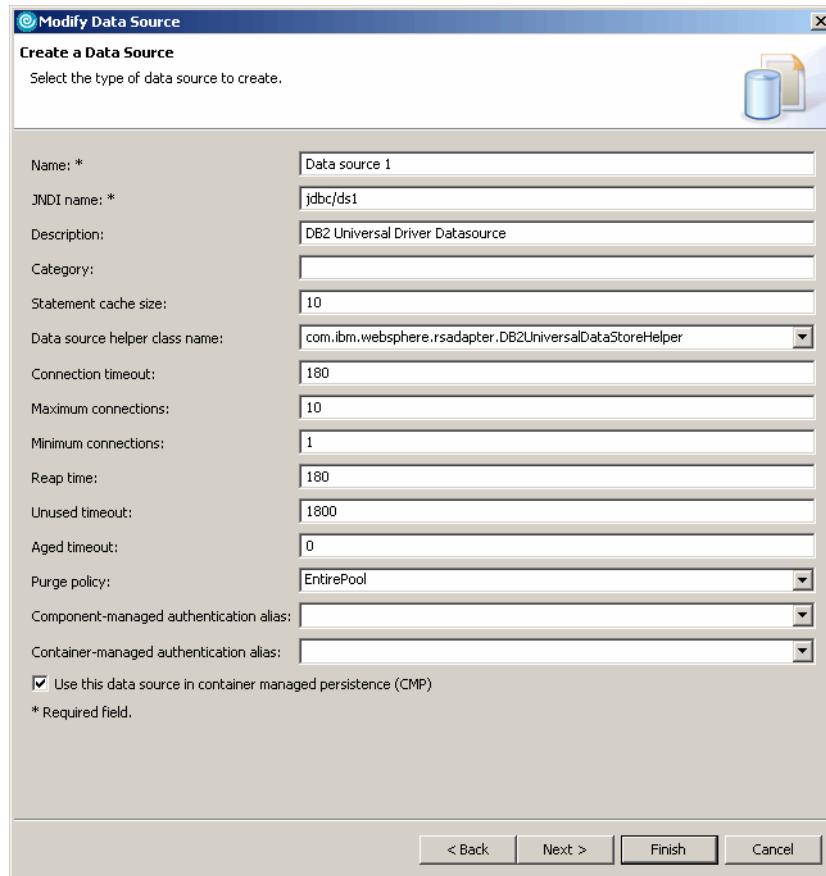


Figure 15-5 Modify Data Source window

13. Fill in the appropriate values for the data source. In general, the following fields should be populated:
 - Name
 - JNDI Name
 - Description
 - Component-managed authentication alias if one has been defined
 - Container-managed authentication alias if one has been defined
14. Click **Next**. The Create Resource Properties window will be displayed.

15. From the list of Resource Properties, select the property to change and enter the value in the value field.

For the DB2 Universal JDBC Provider, the following values should be modified:

- databaseName
- driverType
- serverName (if using type 4 driver)
- portNumber

16. Click **Finish** to return to the Deployment Descriptor Deployment tab.

17. Save the changes and close the file.

15.3 JDBC connection to DB2 from WebSphere Web applications

Connecting to a database from Java is done through a Java Data Source object. The Data Source has two connection methods; one requires a user ID and password as arguments and the other has no arguments. Proper security requires a user ID and password, or Kerberos token if supported when connecting to the database. In J2EE, authenticating to the database within a Web application can be handled by the servlet, JSP, bean, etc., or by delegating it to the Web container. When the authentication is delegated to the Web container, it is called container-managed authentication, otherwise it is component-managed authentication. Which authentication style is chosen determines which connect method on the Data Source to use and the settings in Deployment Descriptor for the Resource Reference. These differences will be noted in this section.

Note: In the following sections, we discuss how to specify information for the WebSphere Bindings of a Resource Reference for connecting to a data store in the Web application Deployment Descriptor. These settings are used either when testing via the integrated WebSphere Test Environment or when using Rational Application Developer to deploy directory to a WebSphere Application Server. Once deployed, or during deployment via the Administrative Console, the WebSphere administrator can override the suggested settings from the Deployment Descriptor.

15.3.1 Defining the Resource Reference

Prior to WebSphere V6, the JAAS authentication alias for Component and Container managed authentication on a JDBC connection was specified when

the Java Data Source was defined. With WebSphere V6, specifying the JAAS authentication alias for Container-managed authentication with the Data Source is deprecated and the selection of the authentication alias is specified in the Deployment Descriptor for the Web application using a Resource Reference. Even if not using Container-managed authentication, the use of Resource References is a best practice when developing J2EE applications. This section shows how to define the Resource Reference.

1. Open the Web Deployment Descriptor (web.xml) for the Web application.
2. Click the **References** tab.
3. Click **Add**.
4. From the Add Reference window, select **Resource Reference** and click **Next**.
5. From the Add Resource Reference window, fill in the appropriate values. For the sample application, use the values from Table 15-2 as shown in Figure 15-6 on page 357.

Table 15-2 Resource Reference Values

Label	Value	Description
Name	jdbc/sample	Name of object in JNDI
Type	javax.sql.DataSource	Object type
Authentication	Application	Determines who supplies the authentication credentials
Sharing Scope	Shareable	
Description	Sample database Resource Reference	Description of the resource reference

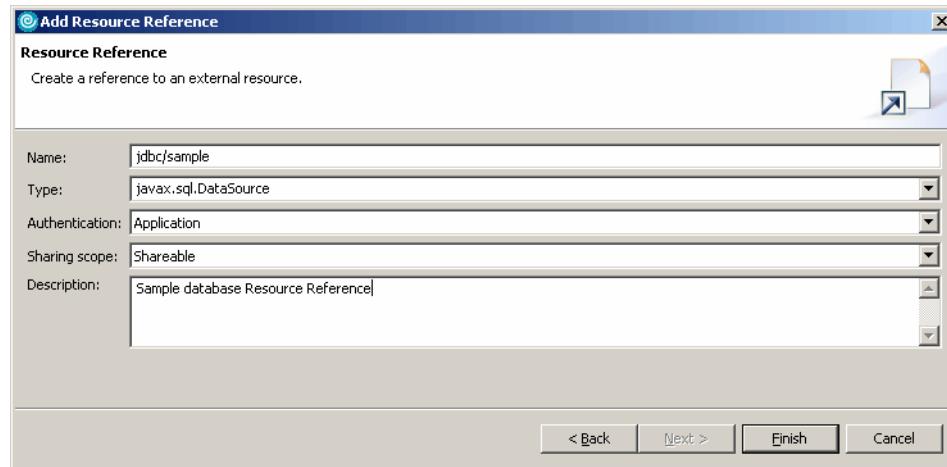


Figure 15-6 Add Resource Reference

6. Click **Finish** to save the Resource Reference to the Web Deployment Descriptor.
7. On the Web Deployment Descriptor References tab, select the Resource Reference just defined.
8. In order to test the application on a WebSphere Test Environment or to deploy the application directly from Rational Application Developer to a WebSphere server, the WebSphere Bindings JNDI Name field must be filled in with the name of the JDBC Data Source object that has been properly defined for accessing the database. Enter the name of the JDBC Data Source object defined in WebSphere Bindings JNDI Name field for the Resource Reference. For the test application, we named the JDBC datasource the same as the Resource Reference, jdbcsample.

Note: In order to test the application on a WebSphere Test Environment or to deploy the application directly from RAD to WebSphere Application Server, the WebSphere Bindings information must be specified. The values specified must be the names of the previously defined data source object and JAAS authentication alias if required for the JDBC Resource Reference.

During deployment of the application using the Administration console, these values can be overridden.

9. If Authentication is set to Container for the Resource Reference, a JAAS Login alias must be specified as discussed below. For the sample application, Authentication was set to Application.
 - a. Select **Use Default Method** from the WebSphere Bindings section.
 - b. Enter the name of a defined JAAS Authentication Alias for use when connecting to the database. For the sample application, use the JAAS Authentication Alias created in “Defining the J2C Authentication Alias” on page 414.

Note: For Resource References using Container-based authentication, the only change needed to the sample code in 15.3.2, “Obtaining the JDBC Connection” on page 358 is documented in Example 15-4 on page 360.

Note: If Authentication is set to Container for the Resource Reference, there are three options for JAAS Login Configuration that become active in the WebSphere Bindings section. The options are Container-managed Authentication (Deprecated), Use Default Method, and Use Custom Login Configuration.

Container-managed Authentication (Deprecated) is used when the JAAS Authentication Alias is specified on the Data Source in WebSphere Application Server. This functions the same as in WebSphere Application Server V5, but is deprecated in WebSphere Application Server V6 in favor of specifying the linking between the JAAS authentication alias and the Data Source via the Resource Reference by selecting **Default Method** and specifying the Authentication Alias name.

For further explanation and configuration information, see the WebSphere Application Server V6 Information Center article titled “J2EE Connector Security”.

10. Save and close the Web Deployment Descriptor.

15.3.2 Obtaining the JDBC Connection

In order to access data within a database, a connection to the database must be established. A JDBC Connection is a Java implementation of the connection to the database. JDBC Connection objects are obtained from a JDBC Data Source. WebSphere managed Data Sources are stored in the JNDI Namespace and can be obtained by using the JNDI APIs.

The following steps show how to locate and open a connection to a database from a Data Source stored in JNDI.

1. Obtain the JNDI Context

- Get the InitialContext
- Get the specific context

Example 15-2 Getting the JNDI Context

```
InitialContext ic = null;
try {
    //Get the jndi initial context
    ic = new InitialContext();
} catch (NamingException e) {
    //Can't open InitialContext so exit
    e.printStackTrace();
    return;
}
Context myEnv = null;
try {
    //Get the context that is specific for this WebApp
    //This holds items such as resource references, ejb references, etc
    //from the web.xml
    myEnv = (Context) ic.lookup("java:comp/env");
} catch (NamingException e1) {
    //Can't find our specific Context to exit
    e1.printStackTrace();
    return;
}
```

2. Look up the Resource Reference.

Example 15-3 Looking up the Resource Reference

```
try {
    //Get the resource reference for the sample datasource
    //that is defined in the web.xml with the following attributes:
    // Name: jdbc/sample Type: javax.sql.DataSource
    // Authentication: Application
    // WebSphere Bindings JNDI Name: jdbc/sample
    sample_Datasource = (DataSource) myEnv.lookup("jdbc/sample");
} catch (NamingException e2) {
    //Can't find the resource reference so exit
    e2.printStackTrace();
    return;
}
```

Tip: To avoid looking up the JNDI context and datasource objects each time they are needed, this is usually done in the init method for the servlet and the data source object stored as a private field for the servlet.

3. Get a connection from the Data Source.

Example 15-4 Getting a JDBC connection

```
try {  
    //If Authentication set to Application for the Resource Reference, and  
    //we are not relying on the Component Managed authentication alias  
    //specified on the data source, use the following:  
    con = sample_Datasource.getConnection(userID, password);  
    //If Authentication is set to Container for the Resource Reference, or  
    //Authentication is set to Application and we are relying on the  
    //specification of the Component Managed authentication alias on the  
    //Data source, use the following:  
    //con = sample_Datasource.getConnection();  
} catch (SQLException e) {  
    request.setAttribute("ErrorMessage", "Error connecting to database.");  
    e.printStackTrace();  
    return;  
}
```

Note: To avoid resource starvation and connection pool object reclamation by the application server, remember to close the connection as soon as possible using the close() method on the connection.

15.4 EJB persistence in WebSphere using DB2

How do you secure the database connections used for the EJB container?

The connection between an EJB Entity Bean and its back-end data source depends on the type of persistence used. Container-Managed Persistence or Bean Managed Persistence can be used. As with Web applications, the basic method of securing the Java component to data source connection is via the required use of a user ID and password when connecting to the data store. The method used to specify and secure the connection between the Entity bean and the data store for each type of persistence is very different and will be discussed in the following sections.

Note: In the following sections, we discuss how to specify information for the WebSphere Bindings of an EJB bean or Resource Reference for connecting to a data store in the EJB Deployment descriptor. These settings are used either when testing via the integrated WebSphere Test Environment or when using Rational Application Developer to deploy directory to a WebSphere Application Server. Once deployed, or during deployment via the administrative console, the WebSphere Application Server Administrator can override the suggested settings from the EJB Deployment Descriptor.

15.4.1 Bean Managed Persistence Entity Beans

Bean Managed Persistence (BMP) Entity beans are responsible for connecting to the data store and retrieving/updating its properties. The process of creating a secure connection to the BMP's data source is very similar to the steps required to create a secure connection to a data store from a servlet, JSP, etc. in a Web application. The BMP bean looks up the JDBC object in JNDI then connects to the data store and performs the required operation. While it is possible to directly look up the JDBC data source in JNDI, it is better to define a Resource Reference for the bean to use when looking up the data source. Due to the similarities with the steps found in 15.3, “JDBC connection to DB2 from WebSphere Web applications” on page 355, we will only discuss the differences in this section.

- ▶ Defining the Resource Reference

In 15.3.1, “Defining the Resource Reference” on page 355, we detailed the steps to create a Resource Reference for the Web application component to connect to the data source. Creation of the Resource Reference for a BMP Entity is identical except for the following:

- In step 1 on page 356, instead of opening the Web Application Deployment Descriptor, open the EJB Deployment Descriptor for the EJB project.
- In step 3 on page 356 you must first select the BMP Entity from the list before clicking **Add**.

Note: Unlike Resource References defined in a Web Application Deployment Descriptor (web.xml), where a Resource Reference once defined is available for all components of a Web application, Resource References for an EJB are only available for that EJB and not other EJBs in the same project.

► Obtaining the JDBC Connection

In 15.3.2, “Obtaining the JDBC Connection” on page 358, we gave sample code for looking up the Resource Reference and getting a connection to the data source. The sample code found in Example 15-2 on page 359 and Example 15-3 on page 359 can be placed in the ejbCreate() method of the BMP bean. The sample code found in Example 15-4 on page 360 should be placed in a private method that returns a connection object so that connections can be obtained as necessary through the Bean’s life cycle. As when connecting to any resource, remember to disconnect from the resource as soon as the resource is no longer needed.

15.4.2 Container Managed Persistence Entity beans

The other type of Entity bean that we are concerned about is the Container Managed Persistence (CMP) Entity bean. For CMP beans, the EJB Container handles the connection between the bean and the data store and automatically updates the data store when changes are made to the bean’s properties. In this case, the connection between the bean and the data source is specified in one of two ways. The JNDI Connection Factory can be specified at the CMP bean level, or a default JNDI Connection Factory can be defined for all CMP beans in the EJB jar that do not specify their own Connection Factory. Regardless of which method is used the information required is the same.

Following are the steps to define the connection factory.

1. Open the EJB Deployment Descriptor.
2. If defining the “default” connection factory, do the following:
 - a. Click the **Overview** tab and scroll down to *JNDI - CMP Connection Factory Binding*.
 - b. In the JNDI Name field, enter the JNDI name of a previously defined data source to be used when deploying this application directly to WebSphere Application Server or during testing using WebSphere Test Environment. For the sample application, enter jdbc/sample.
 - c. Select the appropriate Container Authorization type. For the sample application, we used the following:

Table 15-3 Container-managed security configuration

Attribute	Value
Container authorization type	Container
JAAS Login Configuration	Use Default Method

Attribute	Value
Authentication Alias	Alias defined in , “Defining the J2C Authentication Alias” on page 414

- d. If the authorization type is set to Container, enter the appropriate JAAS authentication information.

Note: If Authentication is set to Container, there are three options for JAAS Login Configuration that become active. The options are Container Managed Authentication (Deprecated), Use Default Method, and Use Custom Login Configuration.

Container Managed Authentication (Deprecated) is used when the JAAS Authentication Alias is specified on the Data Source in WebSphere Application Server. This functions the same as in WebSphere Application Server V5 but is deprecated in WebSphere Application Server V6 in favor of specifying the linking between the JAAS Authentication alias and the Data Source via the Deployment Descriptor using the “Default Method.”

For further explanation and configuration information, see the WebSphere Application Server v6 Information Center article titled “J2EE Connector Security.”

- e. Save and close the EJB Deployment Descriptor.
3. If defining a connection factory at the CMP bean level, do the following:
- a. Click the **Bean** tab.
 - b. Select the CMP bean from the list.
 - c. Scroll to the WebSphere Bindings section.
 - d. In the JNDI Name field, enter the JNDI name of a previously defined data source to be used when deploying this application directly to WebSphere Application Server or during testing using WebSphere Test Environment. For the sample application enter jdbc/sample.
 - e. Select the appropriate Container Authorization type. For the sample application, we used the following:

Table 15-4 Container-managed security configuration

Attribute	Value
Container authorization type	Container
JAAS Login Configuration	Use Default Method

Attribute	Value
Authentication Alias	Alias defined in , “Defining the J2C Authentication Alias” on page 414

- f. If the authorization type is set to Container enter the appropriate JAAS authentication information. Refer to step d on page 363 for a note regarding Container authorization and JAAS authentication information.
- g. Save and close the EJB Deployment Descriptor.

15.5 Sample application

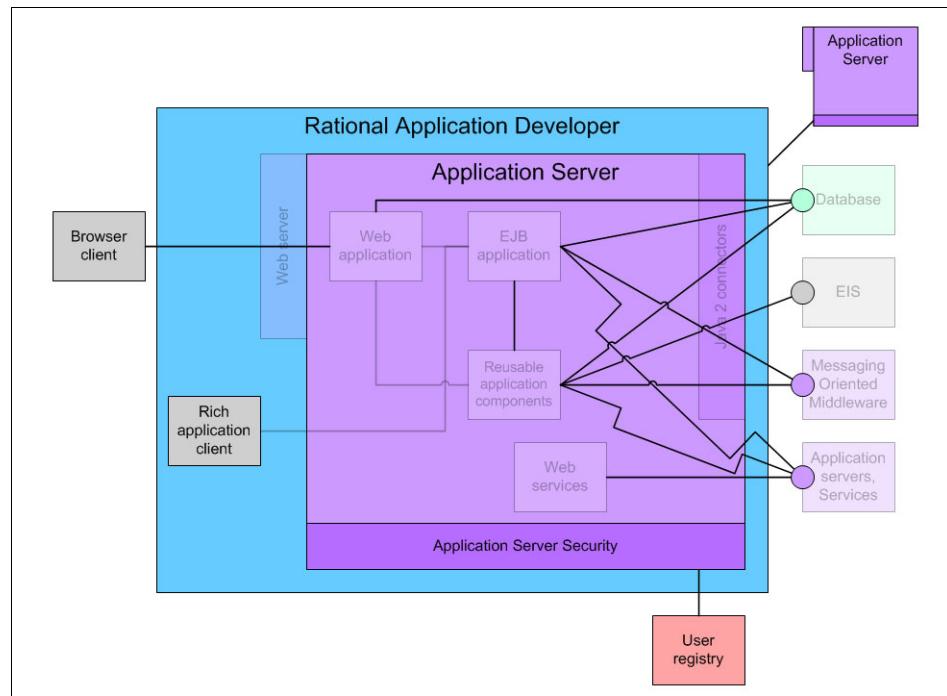
For the details of the sample application refer to “Sample application for database connections” on page 411.



Part 3

Development environment

Development environment security



16.1 Rational Application Developer

Rational Application Developer (formally known as WebSphere Studio Application Developer) is a comprehensive Integrated Development Environment based on Eclipse open source platform to quickly design, develop, analyze, test, profile and deploy Web, Web Services, Java, J2EE and portal applications. Rational Application Developer will be closely integrated with Rational products to leverage the powerful Rational Software technologies to improve the software development life cycle.

The new WebSphere Rapid Deploy feature makes Rational Application Developer easier to system, unit test and deploy applications to WebSphere Application Server V6 and also provides continued and full support to WebSphere Application Server V5.0 and V5.1.

There are a very few changes between WebSphere Studio Application Developer and Rational Application Developer from a security point of view. This section will discuss the development tool from a security point of view. The application development security issues and WebSphere Application Server security configuration issues are already discussed in various sections of this book.

16.1.1 Securing the workspace

The primary issue from the development tool point of view is to secure the workspace where the actual application code resides. Many organizations will use a code repository tool like CVS or Rational Clear Case. Rational Application Developer will utilize the security mechanism provided by the repository actual tool. For setting up security for the tool used with the repository, refer to the appropriate tool documentation.

Regarding the security of the actual workspace, where the copy of the code resides, Rational Application Developer relies more on operating system security. To gain access to the Rational Application Developer, one should be able to log in to the machine on which Rational Application Developer has been installed. There is no separate user ID or password needed to access the Rational Application Developer application. Though there is no direct authentication mechanism for Rational Application Developer access, there are a couple of indirect ways to secure from unauthorized users performing unnecessary or unwanted changes to the artifacts.

Operating system access control

Rational Application Developer keeps all the artifacts under the specified workspace directory. One way of securing the workspace is to provide access to

only the developer on the machine so that other users cannot access the directory.

Windows hosted development environment

Secure the workspace directory:

1. Open the Windows Explorer, select the workspace directory you want to secure and open up the properties.
2. Select the **Security** tab and provide proper access to the right users and groups.

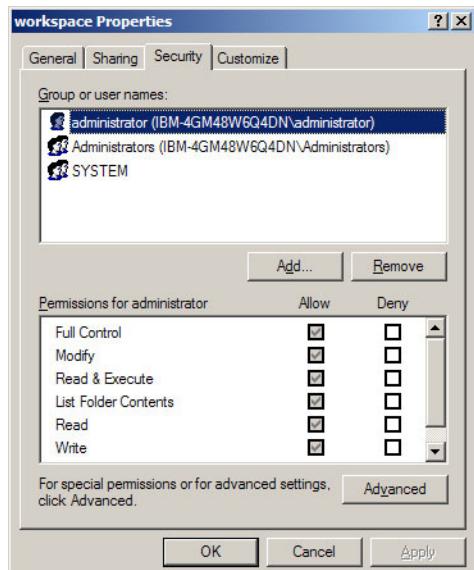


Figure 16-1 Modifying workspace directory access under Windows

If any user tries to open the workspace without proper access, Rational Application Developer will not be able to open the workspace since the user ID does not have read access to the workspace directory; the following error message will be displayed.

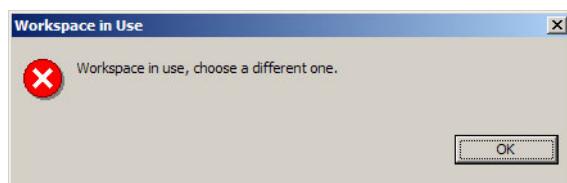


Figure 16-2 WorkSpace directory access error

Linux hosted development environment

Under a Linux operating system, the access can be changed by creating user groups, adding the user to the group and providing workspace directory read access to the group by using the following command:

```
chmod 770 <workspace_directory>
```

The above command will provide complete access to the owner and the group and will not give any access to others. Another way of providing access control is at the user level using the following command:

```
chmod 700 <workspace_directory>
```

This will provide full access to the user and others will not be able to access the workspace directory at all.

Network security

Another option is to keep the workspace on the shared network drive instead of the local drive. The access to the directory can be authenticated at mounting time, mapping time or access time and can be done by providing a user ID and password.

16.2 WebSphere Test Environment

The creation of a new server, configuration or modifications can be done under the server view.

Note: There is no Server perspective in Rational Application Developer; only the Server view is available.

The Server view displays the list of all the available servers and the configurations associated with each server. The server name and the host name identify a unique server. The server view also displays the status of all the servers. The *Status* could be one of the following:

- ▶ Starting
- ▶ Started
- ▶ Started in debug mode
- ▶ Started in profile mode
- ▶ Stopping
- ▶ Stopped

The *State* of the server defines what action needs to be taken based on the server configuration set. For example, when a resource associated to the server

has been modified (for example, a JSP file has been updated), the project needs to be re-published.

Note: If the server tools detect that a file defined to run on a particular server has changed, and the **Automatically restart servers when necessary** check box has been selected on the Server preferences page (**Window** → **Preferences** → **Server**), the server tools automatically restart that server. The Status column in the Servers view changes from Started to Stopped to Started.

Table 16-1 Possible server status

Server state	Description
Server is synchronized	Both the server configuration and the applications are in sync.
Server should be restarted	The server needs to be restarted in order for the changes to take place
Server should be restarted and republished	Either the server configuration or the applications or both have changed. The changed files need to be republished

Important: If Rational Application Developer has been installed with the IBM WebSphere Application Server V6.0 Integrated Test Environment option, a WebSphere Application Server V6.0 server will be automatically created when the workbench has been started for the first time and can be viewed under server perspective.

16.2.1 Creating a new test server

A server is a runtime environment that will be used for testing the project resource. The test environment does not have to reside on the same development machine. The WebSphere V6 server can be created in a separate machine and can be configured under Rational Application Developer as a test server. This increases performance since the test environment process and the development tool run on different machines.

To create a new WebSphere Test Server, follow these steps:

1. Select **File** → **New** → **Other** from the menu.
2. Select the **Show all wizards** check box.
3. Expand the server folder and then select **Server**. Click **Next**.

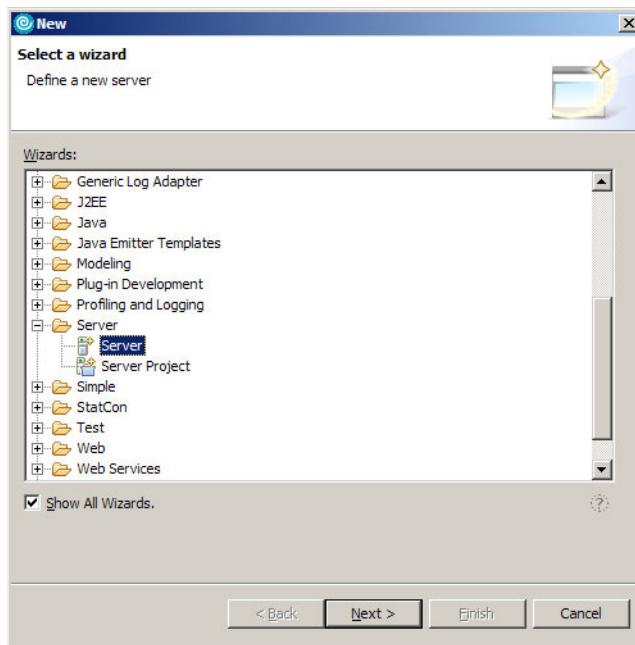


Figure 16-3 Creation of new Server

4. The server creation wizard starts. Any existing server can be created and configured as a WebSphere Test Environment.

In the Host name field, provide the fully qualified DNS name or IP address of the remote host name that the server is running on; or use localhost if you are planning to use a local application server.

In the *Select the server type* list, select the server or test environment where the resources are to be published. This could be a WebSphere V6 or v5.x server. The next configuration is valid only if WebSphere V6 is selected.

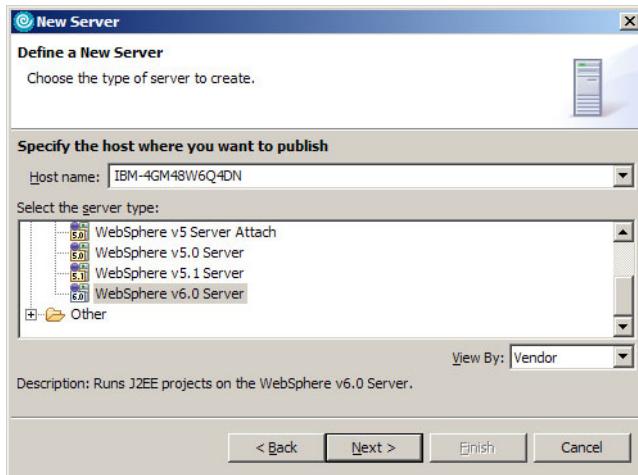


Figure 16-4 Selection of server version

5. If the host name provided has multiple WebSphere profiles running then the WebSphere profile name field will display all the available profiles and any one can be selected as a WebSphere Test Environment server profile. With the default application server, a default profile is created. You can create new profiles in your application server; for more details, refer to “Creating a new profile” on page 381.

You can use the **Detect** button to find out the type of server for the profile under the given host name. This option can be used to check whether the profile selected is Network Deployment profile or Base / Express profile, otherwise you can manually set the type of the server.

You can leave the Enable security option unchecked at this time; the option can be turned on later.

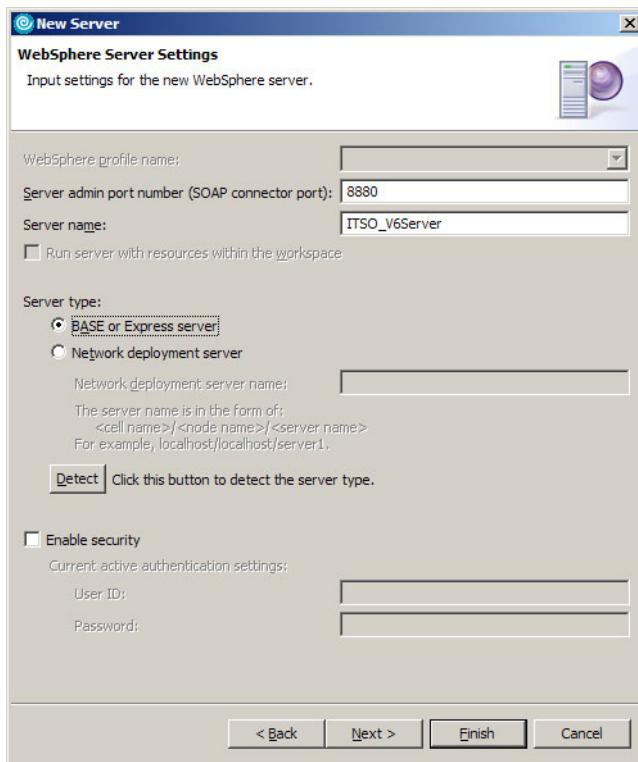


Figure 16-5 WebSphere Server Settings panel

6. Click **Finish**. The new server appears in the Servers view.

16.2.2 Enabling security for the WebSphere Test Server V6

The Enable Security option provides the security authentication.

This section applies for both local as well as remote WebSphere Application Server V6.0. Refer to Chapter 3, “Global Security” on page 31 to understand how to configure Global Security. Only authorized users can start and access the server. When Rational Application Developer tool starts the server, it uses the user ID and password provided under the Enable Security section.

This security configuration can be done either during the creation of the test server or after the test server has been created.

Enabling security when creating the new server

You can configure security for the test server at the time of creating the configuration. Specify the user name and password in the wizard under the WebSphere Server Settings panel.

Restriction: The server must be configured with Global Security before configuring a WebSphere Test Environment under Rational Application Developer.

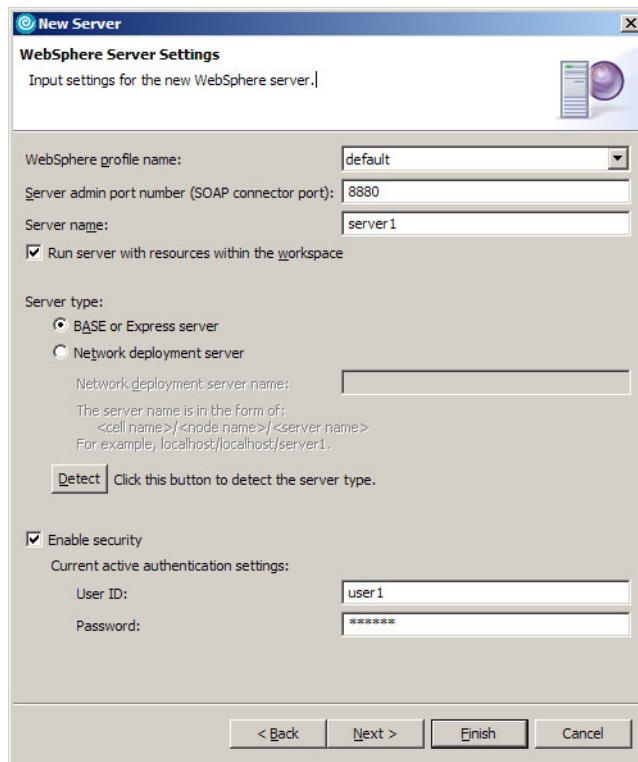


Figure 16-6 Test Server startup security setting

Configuring security for an existing test server

In the Server view, select the test server you are going to configure.

To configure the startup services security option, expand the Security feature. Provide the user name and password for the current active authentication settings defined in the server configuration.

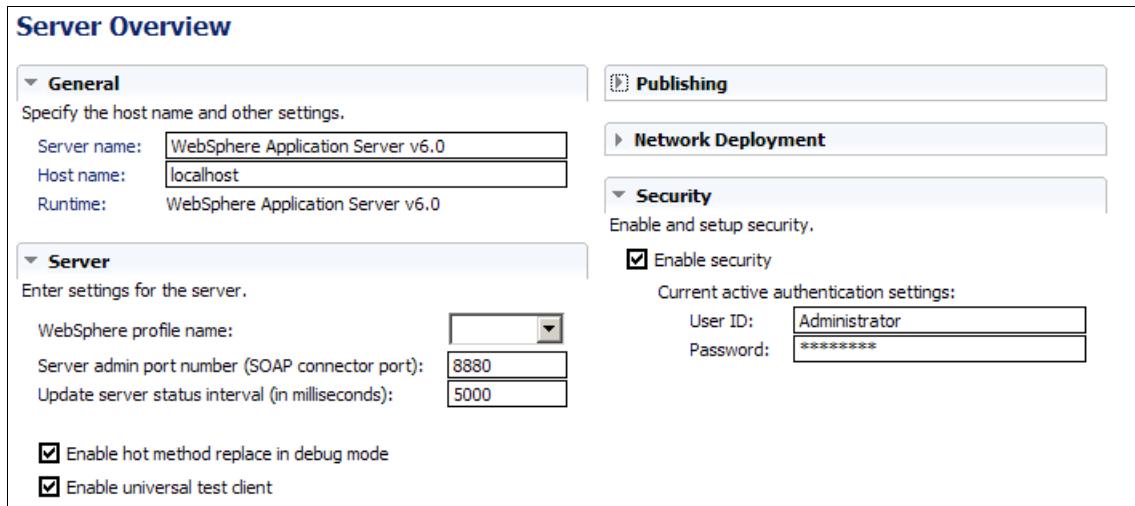


Figure 16-7 Test server startup security update

16.3 Administering and configuring the WebSphere test servers

Administering and configuring the server can be done either within or outside of the Rational Application Developer.

Inside the Rational Application Developer

On the Servers view, right-click the server and select **Run administrative console** from the context menu. This will open the browser view with the Administrative Console within the development environment.

Outside the Rational Application Developer

Open your favorite Web browser and type `http://<hostname>:9060/ibm/console` to open the Administrative Console. The port number will depend upon the profile configuration.

Note: Refer to other sections in this book to configure the server from the security point of view.

16.4 Enterprise application security

This section covers some of the security aspects of enterprise applications in the development environment. There are some deployment time tasks that can be performed in the development environment using the features of Rational Application Developer. You will find two topics discussed here:

- ▶ Configuring enterprise application security during the development phase
- ▶ JAAS authentication entries in the deployment descriptor

16.4.1 Configuring enterprise application security during the development phase

This section will discuss the mapping of application roles defined in the Web module and EJB module to actual users and groups in the user registry. To find out more about the Web resources security and creation and configuration of the Web deployment descriptor, refer to Chapter 6, “Securing a Web application” on page 65. To find out more about the EJB resources security and creation and configuration of EJB deployment descriptor, refer to Chapter 7, “Securing an EJB application” on page 119.

The user and groups mapped under a certain role may or may not match the user and group names in the user registry after deployment. Because of this reason, it is not recommended to do role mapping during development or assembly time. There are situations when role mapping before deployment time can be useful and can shorten the deployment time, for example during testing.

If you have role mapping definitions in the enterprise application descriptor and you are going to deploy the application in a different environment where the mappings are not valid, you can re-map your roles to the actual user registry during deployment.

To create a new security configuration:

1. Double-click the **Deployment Descriptor** under the enterprise application.
2. Select the **Security** tab.
3. Click the **Gather** button. This will get the security roles defined for Web and EJB modules defined under the enterprise application.

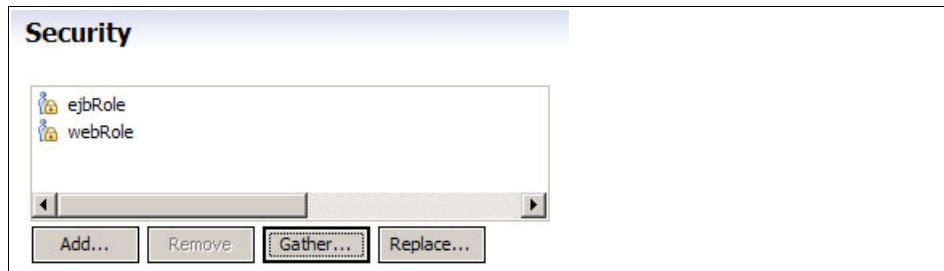


Figure 16-8 Gather the roles defined in Web and EJB deployment descriptor

You can bind the roles in the deployment descriptor to the following subjects:

- ▶ **Everyone** - Literally everyone is mapped to this role; any user can get access to the resource that is associated with the role.
- ▶ **All authenticated users** - Any user that is authenticated against the user registry can get access to the resource that is associated with the role.
- ▶ **Users/Groups** - Only the users or users from the groups mapped to the role can access the resource that is associated with the role.
 - Mapping a user to a role:
 - i. Check the **Users/Groups** box.
 - ii. Click **Add** next to the Users area.
 - iii. Provide the user name in the field in the new window that appears, then click **Finish**.
 - iv. Later you can remove or edit a mapping that is defined.
 - Mapping a group to a role:
 - i. Check the **Users/Groups** box.
 - ii. Click **Add** next to the Groups area.
 - iii. Provide the user name in the field in the new window that appears, then click **Finish**.
 - iv. Later you can remove or edit a mapping that is defined.

You can map multiple users and groups to one role.

The roles for the enterprise application are defined in application.xml.

Example 16-1 application.xml with role definitions

```
<?xml version="1.0" encoding="UTF-8"?>
<application id="Application_ID" version="1.4" ... >
...
    <security-role id="SecurityRole_1101950918401">
```

```
<role-name>ejbRole</role-name>
</security-role>
<security-role id="SecurityRole_1101950918411">
    <role-name>webRole</role-name>
</security-role>
<security-role id="SecurityRole_1101950918431">
    <role-name>applicationRole</role-name>
</security-role>
</application>
```

The security role mapping is stored in the ibm-application-bnd.xmi file.

Example 16-2 ibm-application-bnd.xmi provides the binding information

```
<?xml version="1.0" encoding="UTF-8"?>
<applicationbnd:ApplicationBinding xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns:applicationbnd="applicationbnd.xmi">
    <authorizationTable>
        <authorizations>
            <users name="ejbUser"/>
            <role href="META-INF/application.xml#SecurityRole_1101950918401"/>
            <groups name="ejbGroup"/>
        </authorizations>
        <authorizations>
            <specialSubjects xmi:type="applicationbnd:AllAuthenticatedUsers"
name="AllAuthenticatedUsers"/>
            <role href="META-INF/application.xml#SecurityRole_1101950918411"/>
        </authorizations>
        <authorizations>
            <specialSubjects xmi:type="applicationbnd:Everyone" name="Everyone"/>
            <role href="META-INF/application.xml#SecurityRole_1101950918431"/>
        </authorizations>
    </authorizationTable>
    <application href="META-INF/application.xml#Application_ID"/>
</applicationbnd:ApplicationBinding>
```

16.4.2 JAAS authentication entries in the deployment descriptor

WebSphere V6 and Rational Application Developer support extended deployment descriptors. In the extended deployment descriptor, you can configure components that you would configure during deployment time, for example:

- ▶ JDBC Provider, Data source
- ▶ Classloader mode
- ▶ Substitution variables
- ▶ Shared library

- ▶ Virtual hosts
- ▶ Authentication

Under Authentication, you can define JAAS authentication entries that you can use for the data source defined in the extended deployment descriptor as well.

To define JAAS authentication entries, follow the steps below:

1. Open the Deployment Descriptor for your enterprise application.
2. Select the **Deployment** tab.
3. Open the **Authentication** area in the view.

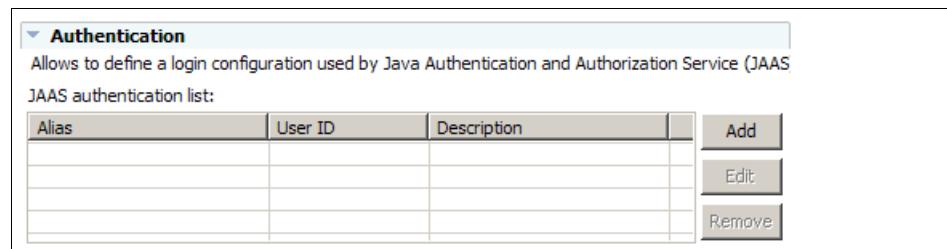


Figure 16-9 Extended deployment descriptor - Authentication

4. Click **Add** next to the Authentication area; a new window appears.

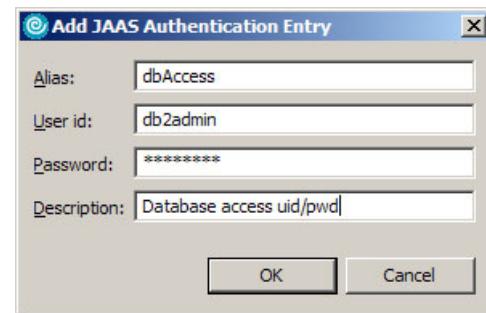


Figure 16-10 Adding a new JAAS entry

5. Fill out the form as needed, then click **OK**.

The JAAS configuration you specify here is actually stored in the security.xml file under the directory structure ibmconfig/cells/defaultCell under the META-INF directory of the enterprise application.

16.5 Creating a new profile for the WebSphere Test Server

In WebSphere Application Server V6, the application server instances are configured under different profiles. If you need to create a new application server instance, you cannot simply create a new application server, you will have to create a new profile. The Profile creation wizard can create a new profile under WebSphere Application Server.

Advantages of multiple profiles

Rational Application Developer creates only one profile, the default, in the test environment for the test server, server1. Using the Profile Creation Wizard, we can create more application server process. Creating a new server profile has the following advantages:

- ▶ Two different teams can test independently of one another using the same machine.
- ▶ Each application under single Rational Application Developer can use an independent test server.

Creating a new profile

There are different ways to create a new profile.

One option is to issue the command directly from the command line. The script is located under <RAD_home>/runtimes/base_v6/bin/Profile Creator. The name of the command is **pctWindows.exe** (Windows) or **pctLinux.bin** (UNIX).

1. Running the command will launch Profile creation wizard.



Figure 16-11 Creating new profile using profile creation wizard

2. Provide the profile name, node name and host name.

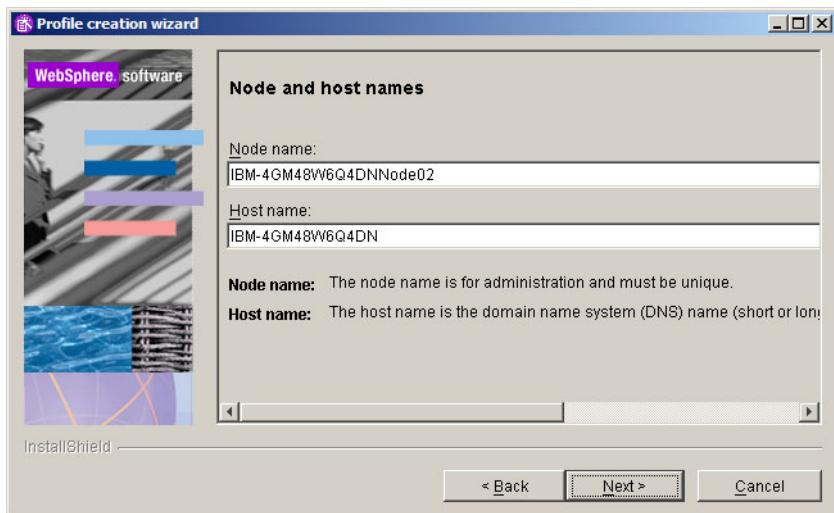


Figure 16-12 Providing node and host name

3. Each new profile is an instance of WebSphere Application Server and will share the same runtime executables and libraries. Every time a new profile is created, the port numbers need to be different than the existing servers.

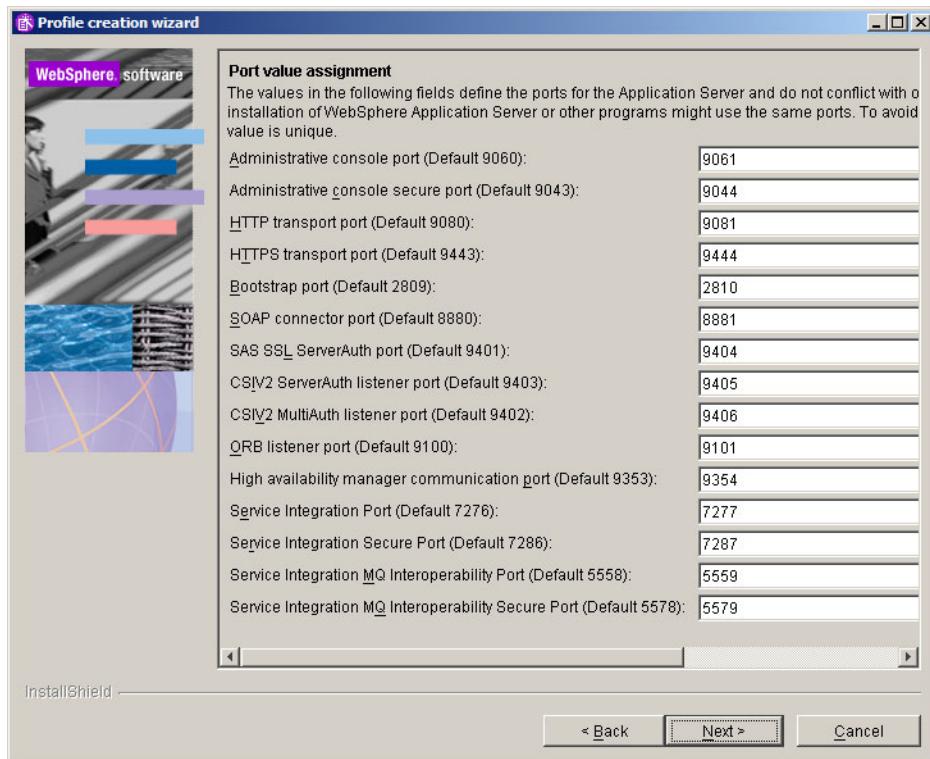


Figure 16-13 Port numbers for the new profile

4. Under the Windows environment, the profile can be registered as Windows services. If the new profile needs to be registered as services under a Windows operating system, provide the necessary information.

Important: The profile's soap port should be provided while creating WebSphere Test Environment under Rational Application Developer; see details under 16.2.1, “Creating a new test server” on page 371.

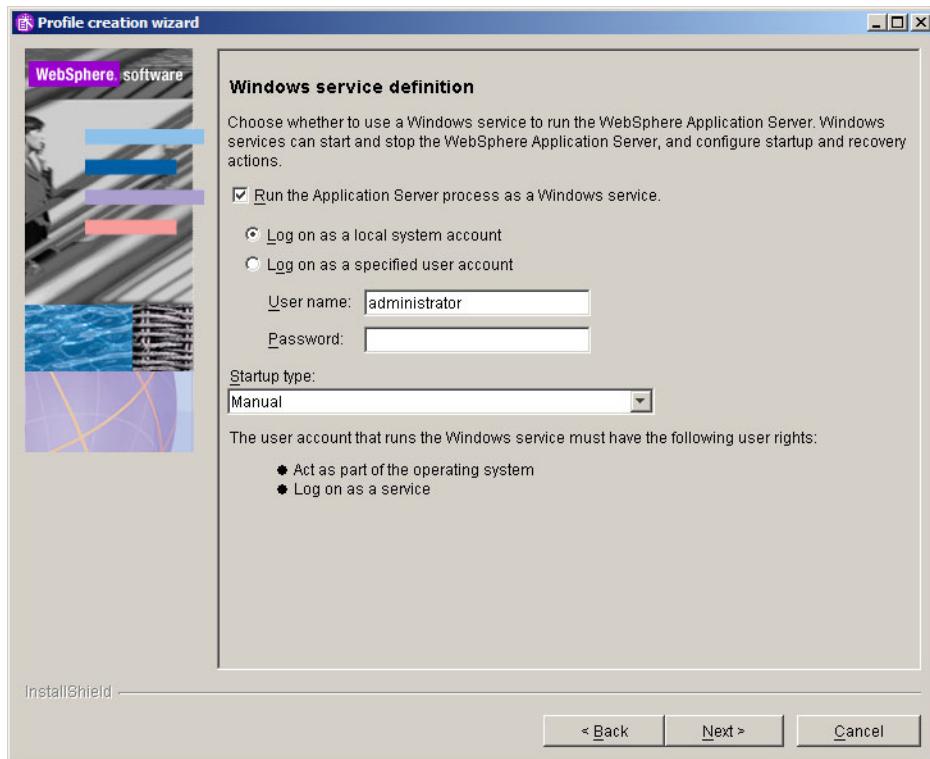


Figure 16-14 Adding profile as Windows services

Important: The specified user must have the following permissions:

- ▶ Log on as service
- ▶ Act as part of the operating system.

The specified user must be logged on as root.

5. The next screen will show the progress of the profile creation. After you are done, click **Finish** to close the wizard.

Once a new profile has been created, the server can be either started from the first step, from the command line or from Rational Application Developer after configuring a new WebSphere Test Server.



Part 4

Appendices

Page 1 of 1



A

Additional configurations

This appendix provides additional information about configurations for WebSphere 6 related to security, and additional information about the sample applications used in the individual chapters of this book.

Sample application for client security

This section provides a brief introduction of how to install the ItsohHello Web component, the ItsohHello EJB resources and the ItsohHello application clients used in this book. It has all the security features built in which were discussed in this book. No special configuration for the WebSphere Application Server is needed. You can simply use the default WebSphere installation settings where the Global Security has been enabled. Below is the high-level diagram of the ItsohHello application:

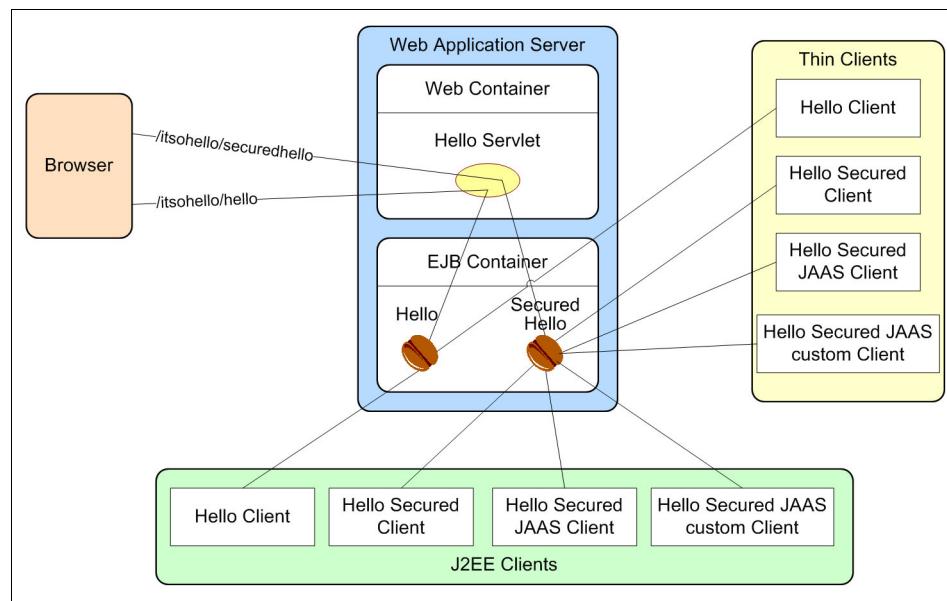


Figure A-1 ItsohHello application

Figure A-1 shows two enterprise beans: Hello and SecuredHello, as the core of the ItsohHello application, installed in a WebSphere Application Server. These resources are accessible via different remote clients: users's browser (via the HelloServlet servlet), four J2EE Java application clients and four thin Java application clients.

Installing and testing ItsohHello application

1. Create a folder AppClient on your workstation, and unzip the contents to the downloaded itsohHello.zip file into this folder:

```
AppClient/ItsohHelloEAR.ear  
AppClient/runJ2EEClient.bat
```

```
AppClient/thinClient/ItsohHelloEJB.jar
```

```
AppClient/thinClient/ItsohelloTHINCLIENT.jar  
AppClient/thinClient/runThinClient.bat  
  
AppClient/thinClient/keys/DummyClientKeyFile.jks  
AppClient/thinClient/keys/DummyClientTrustFile.jks  
AppClient/thinClient/properties/sas.client.props  
AppClient/thinClient/properties/wsjaas_client.conf
```

2. Deploy the ItsohelloEAR.ear enterprise application into a WebSphere Application Server. It will install the Itsohello servlet and the two enterprise beans. Accept all the default setting values, including the warning for was.policy.
3. Test the installed ItsohelloEAR.ear application by accessing the beans using your browser.
 - a. Test the unsecure bean using: http://<hostname>:9080/itsohello/hello. You should get a reply:

Message from bean: Hello to you UNAUTHENTICATED (roles: Anonymous)
 - b. Test the secure bean using:
http://<hostname>:9080/itsohello/securedhello. A window should appear, asking for user ID and password. Fill in *any* user ID and password combination specified in your active WebSphere user registry. This user registry is defined when you enabled the Global Security. If the user ID and password combination is correctly authenticated, you should get a reply such as:

Message from bean: [Secured] Hello to you viking (roles: BeanGuest)
4. To test the J2EE Java application clients, you need to edit the script file runJ2EEClient.bat. Change the following entries to the correct values:

```
set WAS_HOME=C:\WebSphere\AppServer  
set SERVER_HOST=mka0k1my.itso.ral.ibm.com  
set SERVER_PORT=2809
```

WAS_HOME specifies the location of your WebSphere Application Client (if it is not installed you can use the location WebSphere Application Server). SERVER_HOST and SERVER_PORT are the hostname and IIOP port number of the WebSphere Application Server where your enterprise beans (wrapped in ItsohelloEAR.ear). To run the script file, follow these steps:

- a. Open a command prompt window.
- b. Go to the folder AppClient created above.
- c. Run the modified script: runJ2EEClient.bat

The application will show four options corresponding to the four type of J2EE application clients as shown in Figure A-1 on page 388.

Example: A-1 Client application's opening screen

J2EE Itsohello clients:

a. UNSECURED CLIENT.

Access the unsecured Hello bean. If you still get an authentication challenge window, just click "Cancel". Or you can also change the property "com.ibm.CORBA.loginSource" to "none" in the file "sas.client.props".

b. SECURED CLIENT.

Access the secured Hello bean. You should be authenticated, otherwise the app will throw an exception. If you don't get an authentication challenge window, you need to change the property "com.ibm.CORBA.loginSource" to "prompt" in the file "sas.client.props".

c. SECURED CLIENT with JAAS.

Access the secured Hello bean using JAAS. Authentication is done via JAAS.

d. SECURED CLIENT with JAAS using custom callback handler.

Similar like (c) with custom callback handler

Please enter your choice (a/b/c/d):

d. Select a J2EE application client you want to test. For example, for SECURED CLIENT with JAAS, press **c** and **Enter**. An authentication window will be shown asking for user ID and password. When a valid user ID and password is entered (any user ID defined in the user registry), you will see message similar like:

```
Accessing SecuredHello bean using JAAS
Message from Hello bean: [Secured] Hello to you viking (roles:
BeanGuest)
```

5. To test the thin Java application clients, you need to edit the script file runThinClient.bat. Change the following entries to the correct values:

```
set WAS_HOME=C:\WebSphere\AppServer
set SERVER_HOST=mka0k1my.itso.ral.ibm.com
set SERVER_PORT=2809
```

See also 4 on page 389, for more explanation regarding those entries. To run the script file, do as follows:

- a. Open a command prompt window.
- b. Go to the folder AppClient\thinClient created above.
- c. Run the modified script: runThinClient.bat. The rest of the procedure is the same with the procedure to test the J2EE application client.

Sample application for testing JACC

The sample application consists of one Web module, one EJB Module and utility jar. There are four roles defined in this application:

- ▶ WebRole
- ▶ First
- ▶ Second
- ▶ Third

WebRole is used to map the Web Resource and other roles are mapped to specific methods in the EJB.

Web Module

The Web Module contains three servlets. The JACCTestServlet is one invokes the EJB and displays the results. The other two servlet is used to display the deployment descriptor of Web and EJB module.

The JACCTestServlet is protected and mapped to Web Role.

EJB Module

The EJB module contains Stateless Session Bean and exposed with four methods. They are:

1. getDD() - Unprotected method to display the deployment descriptor.
2. getFirst() - Protected method and returns the strings. It is mapped to the "First" role.
3. getSecond() - Protected method and returns the strings. It is mapped to the "Second" roles.
4. getThird() - Protected method and returns the strings. It is mapped to the "Third" roles.

Deployment

In order to deploy this sample application, follow the steps below.

Note: The assumption here is that you already configured security using LDAP and Tivoli Access Manager, and enabled the external authorization using JACC and Tivoli Access Manager.

1. Open the Administrative Console.
2. Install the ITSOJACC.ear application.

3. During the installation at Step 7: Map security roles to Tivoli Access Manager users and groups, perform the mapping for your roles.

Installation verification

Once the application installed all the policy information propagated to Tivoli Access Manager Object space and can be verified through the **pdadmin** command interface

1. Open the **pdadmin** command interface; log in using your sec_master.
2. Issue the command **object list**, you will see three object trees:

```
/Management  
/WebSEAL  
/WebAppServer
```

3. The /WebAppServer is the one where all the WebSphere related information is stored. You can drill down to the lower level of the object tree.
4. Issue the command **Object list /WebAppServer/deployedResources/Roles**, you will see lot of role definition depending on your environment. The highlighted ones belong to the sample application:

```
/WebAppServer/deployedResources/Roles/administrator  
/WebAppServer/deployedResources/Roles/configurator  
/WebAppServer/deployedResources/Roles/CosNamingCreate  
/WebAppServer/deployedResources/Roles/CosNamingDelete  
/WebAppServer/deployedResources/Roles/CosNamingRead  
/WebAppServer/deployedResources/Roles/CosNamingWrite  
/WebAppServer/deployedResources/Roles/First  
/WebAppServer/deployedResources/Roles/monitor  
/WebAppServer/deployedResources/Roles/operator  
/WebAppServer/deployedResources/Roles/Second  
/WebAppServer/deployedResources/Roles/Thrid  
/WebAppServer/deployedResources/Roles/Unchecked  
/WebAppServer/deployedResources/Roles/WebRole
```

Testing

Open the Web browser, type the URL:

`http://<yourhostname>:<port>/JACC/JACCTestServlet`

You will be challenged for authentication; once you provide the right credentials, a page will be displayed.

JACCTestServlet

You were able to view this page (JACCTestServlet) since the access control policy statements specified in the deployment descriptor of this servlet allow you. Click [here](#) to view the deployment descriptor. These policy statements were translated into a set of JACC policies at deployment-time, and they were evaluated by our TAM JACC provider to make the access decision.

Access Control for EJB Methods

This servlet invokes the three methods of an Session Bean bean named JACCTestEJB. The following are the invocation results of the three EJB methods.

1. getFirst

This servlet called the "getFirst" method.

getFirst returned successfully

JACCTestServlet was able to invoke this method since the access control policy statements specified in [the deployment descriptor](#) of the EJB allow it.

2. getSecond

This servlet called the "getSecond" method.

getSecond returned successfully

JACCTestServlet was able to invoke this method since the access control policy statements specified in [the deployment descriptor](#) of the EJB allow it.

3. getThird

This servlet called the "getThird" method.

getThird returned successfully

JACCTestServlet was able to invoke this method since the access control policy statements specified in [the deployment descriptor](#) of the EJB allow it.

Visit Redbooks <http://www.redbooks.ibm.com/>

Figure A-2 Results from the test servlet

Configuring Embedded Messaging

This section will detail the steps to enable embedded messaging and define a queue bus destination. The basic process is:

1. Create the System Integration Bus.
2. Add one or more application servers or clusters to the bus.
3. Define the Queue bus destination.

Note: A default topic space is defined when the messaging engine is configured (defining a topic space will not be demonstrated here). The steps to define a topic space are basically the same as for a queue destination and the only parameters required for the destination are name and description. Once defined, properties can be modified to allow send or receive and enforce topic access checking, among others.

4. Define JMS Objects for application access.

Define the System Integration Bus

To define the System Integration Bus, do the following:

1. From the WebSphere Administrative Console, click + next to Service integration.
2. From the expanded menu, click **Buses**. The list of currently defined buses will be displayed, as shown in Figure A-3.

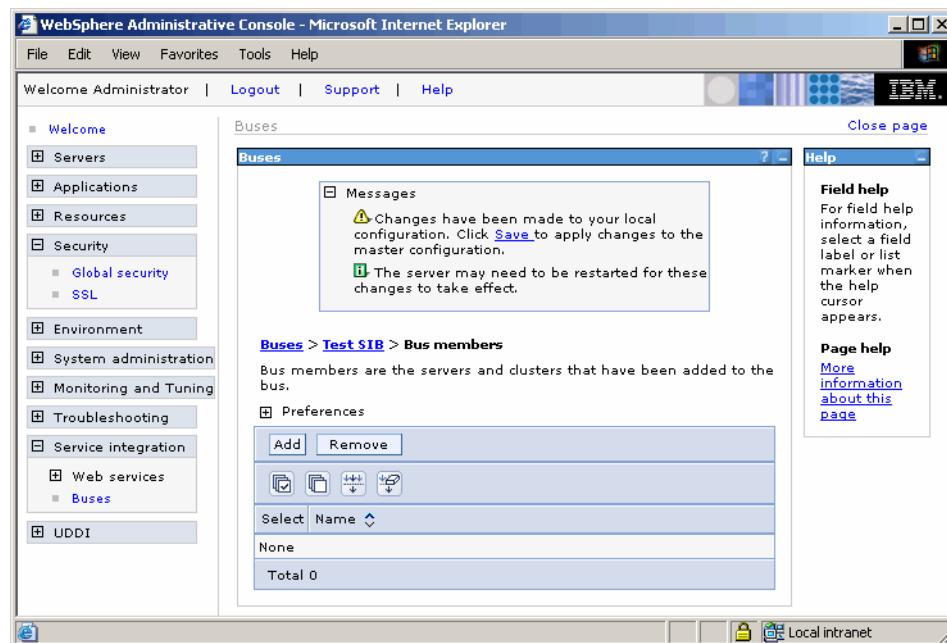


Figure A-3 Defined System Integration Buses

3. Click **Add** to create a new bus. You will see a pane as shown in Figure A-4 on page 395.

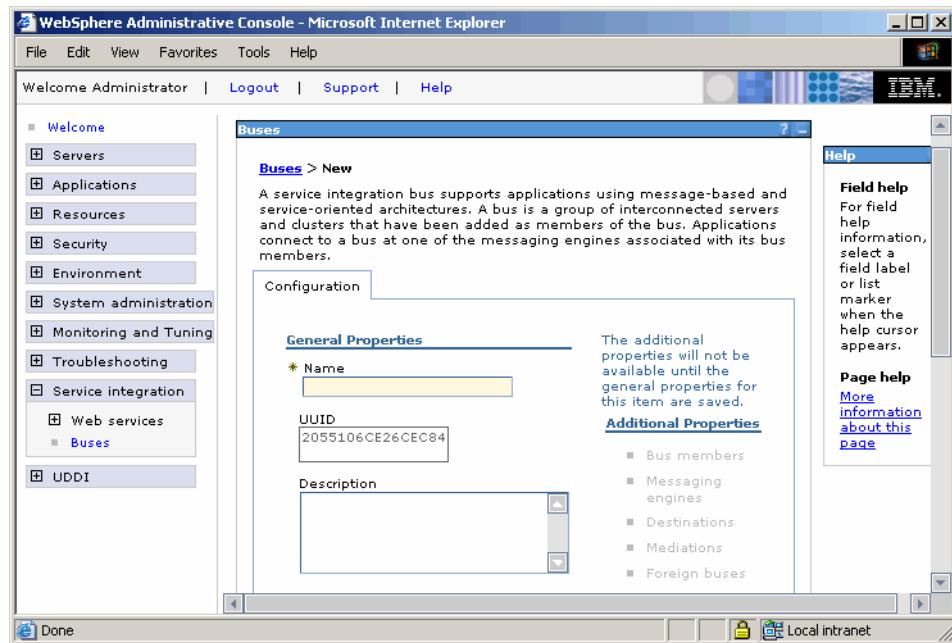


Figure A-4 New System Integration Bus

4. Enter a name for the bus.
5. Enter a description for the bus.
6. If Global Security is enabled and Security is required on the bus, scroll down and select **Secure**.

Note: Checking **Secure** enables security checking on bus resources if Global Security is enabled. If Global Security is not enabled then checking **Secure** has no effect until Global Security is enabled.

7. If SSL will be required for transport of messages between messaging servers on the bus, in the Inter-engine transport chain enter InboundSecureMessaging.
8. Set other properties for the bus as appropriate. Refer to the online documentation for additional info for these properties.
9. Click **OK** to return to the list of defined buses.
10. Save the configuration for WebSphere.

Add an application server to the bus

To add an application server to the bus, do the following:

1. From the list of defined buses, click the bus you want to add the server to.
2. From the bus properties page, click **Bus members**. The bus members page will be displayed as shown in Figure A-5.

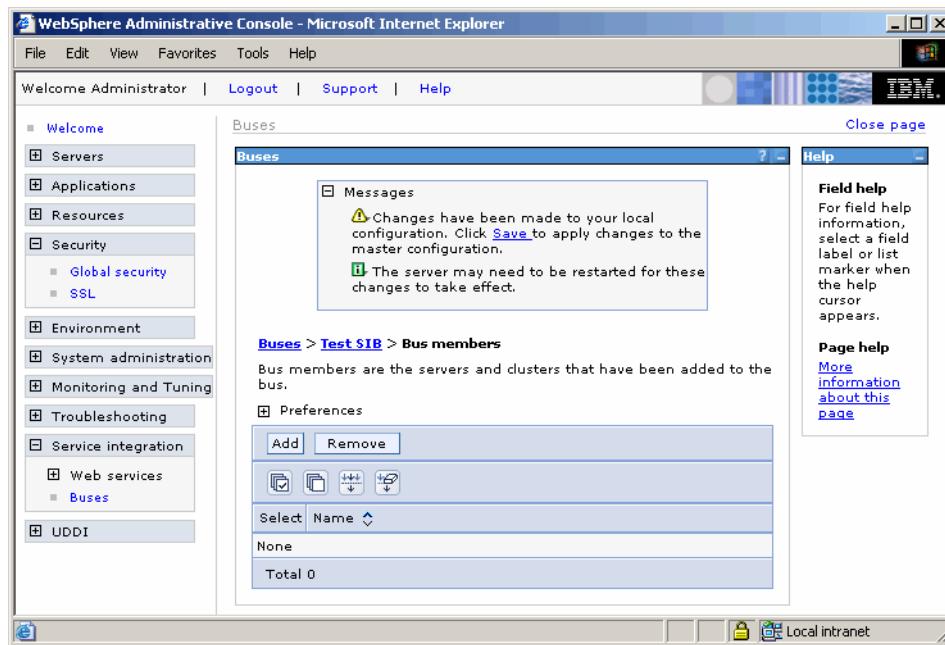


Figure A-5 Bus members list

3. Click **Add** to add an application server to the bus. The Add a new bus member page will be displayed as shown in Figure A-6 on page 397.

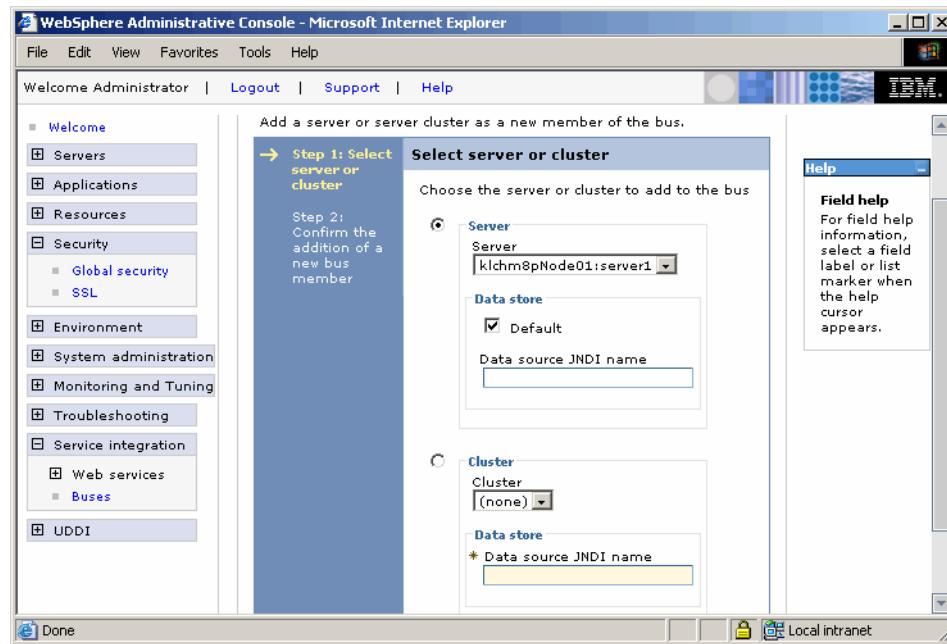


Figure A-6 Add a bus member

4. Select the cluster or server you want to add to the bus. In a single server environment, the application server will already be selected.
5. Leave the Default Data store enabled.
If the default data store is not appropriate for your environment, uncheck Default and enter the JNDI name of the data store you want to use.
6. Click **Next**.
7. Click **Finish** to confirm the changes and return to the bus members page.
8. Repeat the above steps to add additional members to the bus.
9. Save the configuration for WebSphere.

Define a queue destination on the bus

1. From the list of defined buses, select the bus you want to add the destination to.
2. From the bus properties page, click **Destinations**. The destination list page will be displayed as shown in Figure A-7 on page 398.

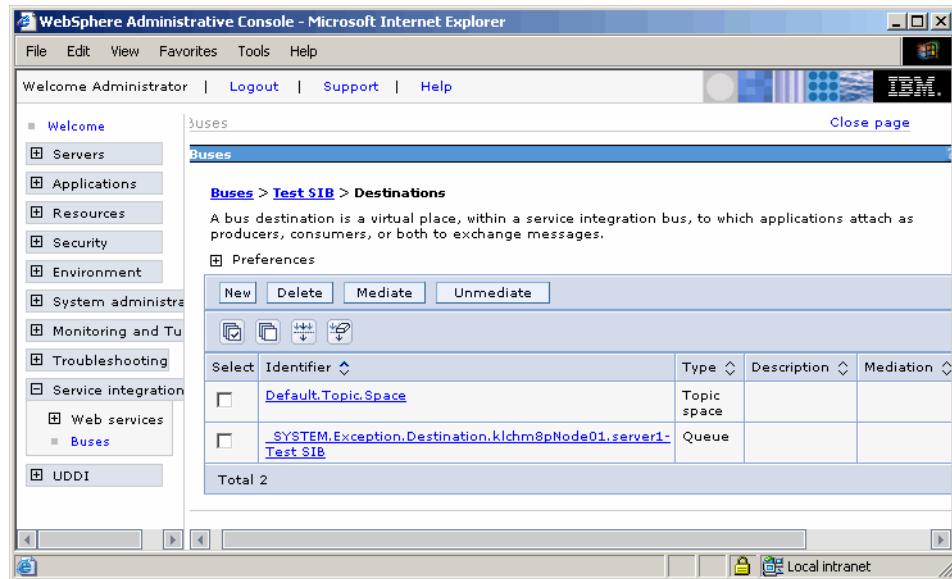


Figure A-7 Destinations List

3. Click **New** to create a new destination. The Create new destination page will display as shown in Figure A-8.

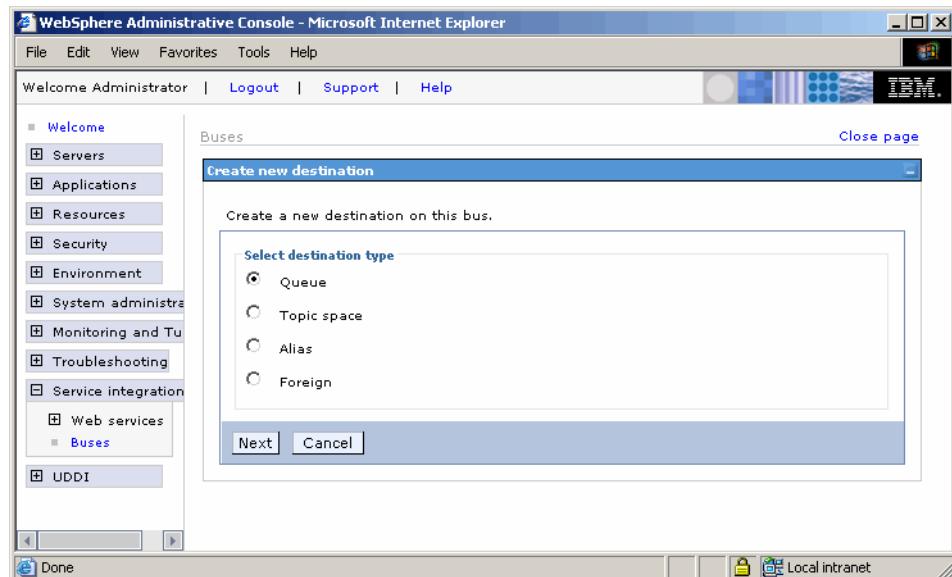


Figure A-8 New Destination Type

4. Select Queue to define a queue destination.
5. Click **Next**. The Create new queue - Set queue attributes page will be displayed as shown in Figure A-9.

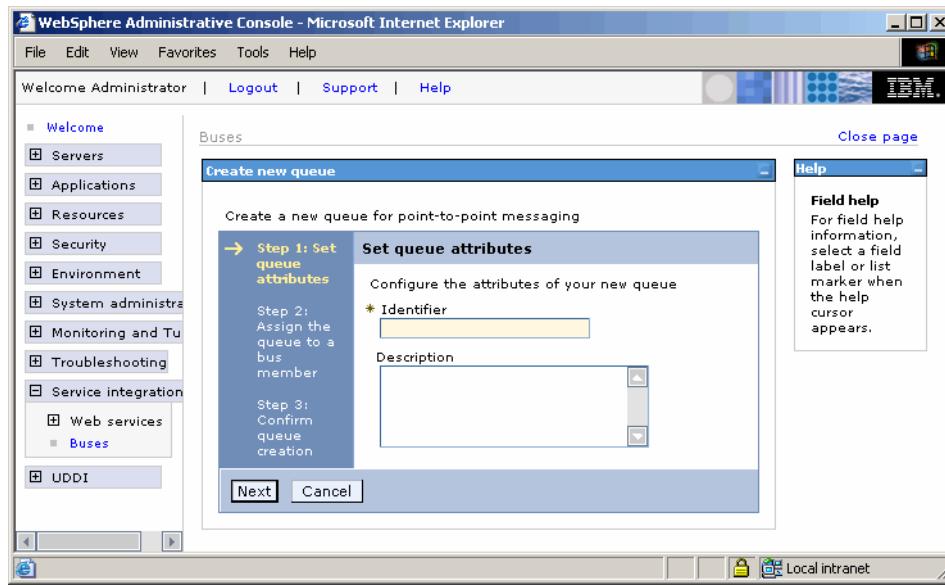


Figure A-9 Create new queue - Set queue attributes

Note: The next few windows will gather the information needed to define the destination. The basic attributes for a Queue or Topic space are an identifier, description and the bus member the destination is defined on.

6. Enter the name of the queue in the identifier field.
7. Enter a description of the queue destination in the description field.
8. Click **Next**. The Create new queue - Assign bus members page will be displayed as shown in Figure A-10 on page 400.

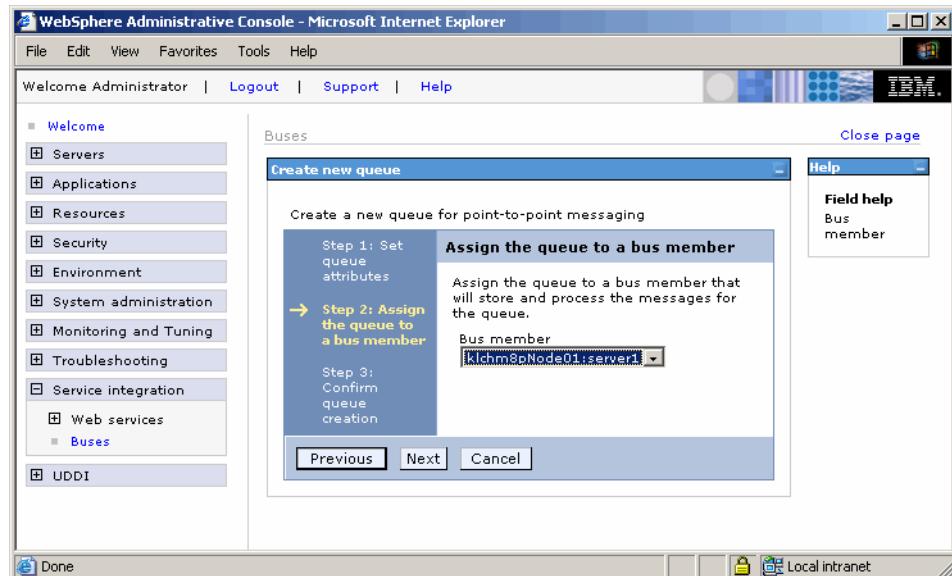


Figure A-10 Create new queue - Assign bus members

9. From the bus member list, select the bus member where the queue destination should be defined.
10. Click **Next**.
11. Review the summary of the actions to be taken and click **Finish** to return to the Destination list page.
12. Save the configuration for WebSphere.

Define the JMS Objects

JMS connection factories and target objects for destinations on the Service Integration Bus are defined using the *Default Messaging* link under JMS Providers in the Administrative Console. The following steps detail defining a JMS connection factory to connect to the local Service Integration Bus and a JMS queue object for a queue destination on the local bus.

1. Open the Administrative Console and log in.
2. Click + next to Resources.
3. Click + next to JMS Providers.
4. Click **Default Messaging** to display the Default messaging provider page.
5. Click **JMS queue connection factory** from the *Connection Factories* section to display the JMS queue connection factory list.

6. Click **New** to create a new queue connection factory.
7. Enter the name and JNDI name for the queue connection factory.
8. In the *Connection* section from the Bus name drop-down, select the local bus hosting the queue destination to connect to.
9. Specify a Target inbound messaging chain or leave blank to use the default for the messaging engine.

Note: This option is used to specify the transport level security for the connection. The two default options for the messaging chain are *InboundBasicMessaging*, which is non-SSL enabled, and *InboundSecureMessaging* which is SSL enabled.

10. Modify additional properties on this page as needed, including defining the Component-managed authentication alias if needed.
11. Click **OK** to return to the JMS queue connection factory list.
12. Click **Default messaging provider** to return to the Default messaging provider page.
13. Click **JMS queue** from the *Destinations* section to display the Destinations list.
14. Click **New** to create a new JMS queue destination.
15. Enter the name and jndi name for the queue.
16. In the *Connection* section from the Bus name drop-down, select the bus hosting the queue destination.

Note: For Queue, Topic space and alias destinations, the local Service Integration Bus where the destination is defined should be selected. For Foreign destinations, the foreign bus should be selected.

Once a bus is selected the page will refresh so that the Queue name field will contain a list of destinations on the selected bus.

17. In the *Connection* section from the Queue name drop-down, select the queue destination.
18. Click **OK** to return to the Destinations list.
19. Save the configuration for WebSphere.

Sample application for messaging

The JMSSampleApplication.ear contains a sample application to demonstrate sending and browsing messages on a Service Integration Bus destination. The sample contains two servlets, one for reading/browsing messages on the queue and the second for writing/sending messages to the queue. This chapter details the steps to configure a WebSphere Application Server's embedded messaging, and optionally WebSphere MQ objects, to demonstrate the application.

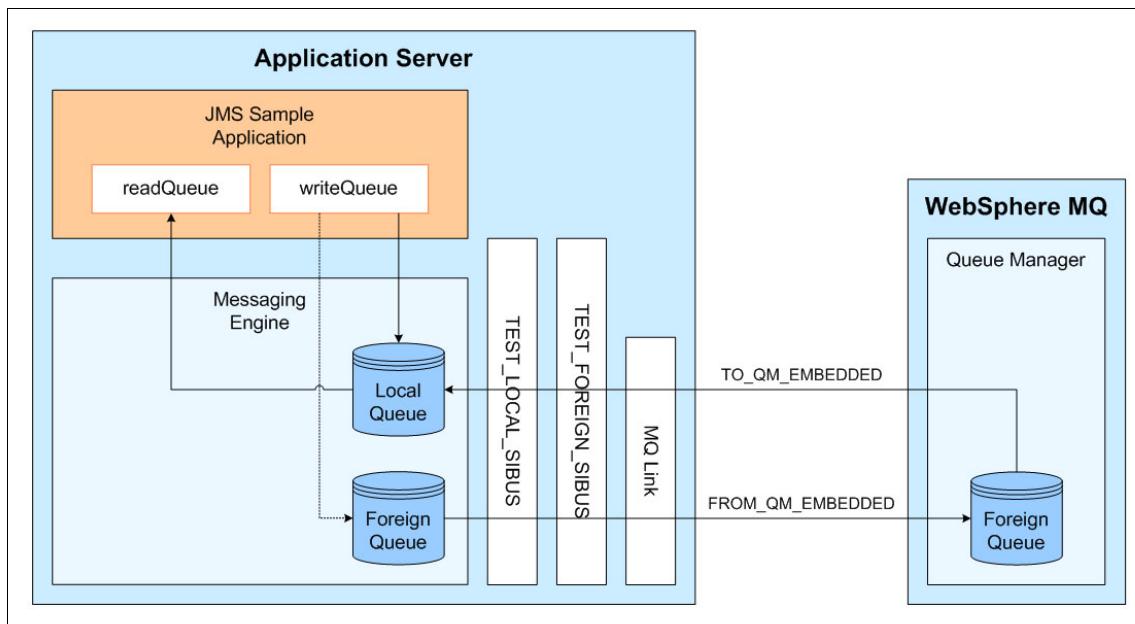


Figure A-11 JMS Sample application flow

Note: The application will be configured to read and write messages to the same queue if not testing with WebSphere MQ.

If WebSphere MQ integration is being tested then messages will be written to the *ForeignQueue* destination on the local bus. The messages are then routed automatically to the *ForeignQueue* on the WebSphere MQ queue manager. The *ForeignQueue* is defined as a remote queue to WebSphere MQ and routed back to the LocalQueue on the WebSphere Application Server messaging engine. Messages cannot be read from a foreign destination, so the application reads them from the LocalQueue on the Service Integration Bus.

Configure the application server

The following resources are needed to test the sample application. The sample application requires Global Security to be enabled. J2EE security is not required. The resources marked optional are only required if testing with WebSphere MQ.

If you are not familiar with configuring the messaging components in WebSphere, refer to “Configuring Embedded Messaging” on page 393.

Use the steps in this section to define the resources needed for the sample application.

1. Create the System Integration Bus.
2. Add the application server to Service Integration Bus.
3. Create the foreign System Integration Bus (Optional)
4. Define the MQ link (optional).
5. Create the queue destination.
6. Create the foreign destination (optional).
7. Create the JMS Queue Connection Factory.
8. Create the JMS queue objects.

Create the System Integration Bus

The steps to define a System Integration Bus are detailed in “Define the System Integration Bus” on page 394. Use those steps to define an Service Integration Bus with the following properties:

Table A-1 Local bus properties

Field	Value
Name	TEST_LOCAL_SIBUS
Secure	Checked

Add the application server to the Service Integration Bus

The steps to make an application server a member of the Service Integration Bus are detailed in “Add an application server to the bus” on page 395. Use those steps to add the application server where the sample application will be installed to the TEST_LOCAL_SIBUS.

Create the foreign System Integration Bus (Optional)

Note: A foreign Service Integration Bus is only required if the sample application will be connected to a WebSphere MQ server.

The steps to define a foreign bus are detailed in “Defining the foreign bus” on page 329. Use those steps to define a foreign bus on the TEST_LOCAL_SIBUS with the following properties, leaving all other properties at their default values.

Table A-2 Foreign bus properties

Field	Value
Name	TEST_FOREIGN_SIBUS
Routing type	Direct, WebSphere MQ link
Inbound user ID	janedoe
Outbound user ID	User name that has access to put messages to MQ. Ask MQ administrator for this.

Note: The user ID janedoe should be defined on the server running WebSphere Application Server V6 with password janedoe. This ID will also be used for the sample application in Chapter 15, “Securing the database connection” on page 343.

Define the MQ link (optional)

Note: An MQ link is only required if the sample application will be connected to a WebSphere MQ server via a foreign Service Integration Bus definition.

The steps to define an MQ Link are detailed in “Defining the MQ link” on page 330. Use those steps to define an MQ Link on the TEST_LOCAL_SIBUS to connect the application server to the WebSphere MQ queue manager using the following properties.

Table A-3 MQ link properties

Field	Value
Name	TEST_LOCAL_SIBUS To TEST_FOREIGN_SIBUS
Foreign bus name	TEST_FOREIGN_SIBUS
Queue manager name	QM_EMBEDDED
Sender MQ channel name	FROM_QM_EMBEDDED
Hostname	<i>Hostname of MQ server</i>
Port	<i>Port that MQ is listening on. Default is 1414.</i>

Field	Value
Transport chain	OutboundBasicMQLink (if not using SSL) OutboundSecureMQLink (if using SSL)
Receiver MQ channel name	TO_QM_EMBEDDED

Note: Channel names, hostname, port, and Transport chain all depend on values that you will need to work with the MQ administrator to get.

The sender channel name must be the name of a Receiver channel on the MQ server. If the channel is SSL enabled on the MQ server then Transport chain should be set to OutboundSecureMQLink, otherwise use OutboundBasicMQLink.

The receiver channel name must match the name of a Sender channel on the MQ server.

Create the queue destination

The steps to define a queue destination are detailed in “Define a queue destination on the bus” on page 397. Use the steps to define a queue destination on the TEST_LOCAL_SIBUS with the following properties:

Table A-4 Queue destination properties

Field	Value
Destination Type	Queue
Name	LocalQueue
Bus member	<i>Application server that will host the sample application. There should only be one in the list.</i>

Create the foreign destination (optional)

Note: A foreign destination is only required if the sample application will be connected to a WebSphere MQ server via a foreign Service Integration Bus definition.

The steps to define a foreign destination are detailed in “Defining a foreign destination” on page 332. Use the steps to define a foreign destination on the TEST_LOCAL_SIBUS with the following properties:

Table A-5 Foreign destination properties

Field	Value
Destination type	Foreign
Name	ForeignQueue
Bus	TEST_FOREIGN_SIBUS

Note: The name property must match a queue name on the WebSphere MQ server to which messages will be transmitted.

For the sample application a remote queue named ForeignQueue will be defined to route messages sent to MQ back to the LocalQueue on the application server. This simplifies the sample application as applications cannot read from foreign destinations.

Create the JMS queue connection factory

The steps to define a JMS queue connection factory are detailed in “Define the JMS Objects” on page 400. Use the steps to create a queue connection factory with the following properties:

Table A-6 Queue connection factory properties

Field	Value
Name	LocalSIB_QCF
JNDI name	jms/qcf_localSIB
Bus name	TEST_LOCAL_SIBUS

Create the JMS queue object for LocalQueue

The steps to define a JMS queue are detailed in “Define the JMS Objects” on page 400. Use the steps to create a queue with the following properties:

Table A-7 JMS properties for LocalQueue

Field	Value
Name	LocalQueue
JNDI name	jms/queue_LocalQueue
Bus	TEST_LOCAL_SIBUS
Queue name	LocalQueue

Create JMS queue object for ForeignQueue (optional)

If you have defined the Foreign destination for WebSphere MQ, create another JMS queue pointing to the Foreign destination. Follow the steps in “Defining the JMS objects” on page 332 and supply the following properties:

Table A-8 JMS properties for Foreign destination

Field	Value
Name	ForeignQueue
JNDI name	jms/queue_ForeignQueue
Bus	TEST_FOREIGN_SIBUS
Queue name	ForeignQueue

Configure MQ

In order for WebSphere MQ and a WebSphere Application Server messaging engine to communicate, a few MQ objects must be defined. The required objects are:

1. Transmit queue
2. Sender channel
3. Receiver channel

The sample application also requires a Remote Queue, named ForeignQueue, to be defined on the MQ server. This remote queue will route all messages placed on the queue back to the LocalQueue on the WebSphere Application Server.

After logging on to the WebSphere MQ server as a user with MQ administration privileges, do the following to define the channels and queues:

1. From the command line, run **runmqsc**.
2. From the **runmqsc** prompt, enter the following commands line by line, press enter where you see the new line symbol (↓).

```
define qlocal(QM_EMBEDDED) usage (XMITQ) ↓
define channel(FROM_QM_EMBEDDED) CHLTYPE(RCVR) ↓
define channel(TO_QM_EMBEDDED) CHLTYPE(SDR) connname('localhost(5558)')
XMITQ(QM_EMBEDDED) ↓
define qremote('ForeignQueue') RQMNAME(QM_EMBEDDED) XMITQ(QM_EMBEDDED)
RNAME('LocalQueue') Put(ENABLED) ↓
start channel(FROM_QM_EMBEDDED) ↓
start channel(TO_QM_EMBEDDED) ↓
end ↓
```

3. Log out if necessary.

Install the Application

The following steps detail the installation of the JMSSampleApplication.ear on WebSphere Application Server.

1. Open the Administrative Console and log in.
2. Click **Applications** → **Install New Application** to start the application installation process.
3. Click **Browse** and locate the JMSSampleApplication.ear file the click **Open**.
4. Click **Next**.
5. Click **Next** on the *generate default bindings* page.
6. Click **Step 3** to skip to Step 3: Map resource references to resources.
7. In the javax.jms.Queue section verify that **jms/queue_LocalQueue** is entered in both of the JNDI name fields. If not, enter it.

Note: Entering the LocalQueue JMS definition into both fields allows the application to test sending and getting messages from the local queue destination. If you wish to test against MQ, enter **jms/queue_ForeignQueue** in the **jms/queue_sender** JNDI name field.

8. In the javax.jms.QueueConnectionFactory section, verify that **jms/qcf_localsIB** is entered in the JNDI name field. If not, enter it.
9. Click **Next**.

Note: Once **Next** is clicked, an Application Resource Warning page may be displayed. If the only warnings are ADMA8019E warnings then click **Continue**. If additional warnings are displayed, they will need to be corrected.

- 10.Click **Step 6** to skip to Step 6: Summary.
- 11.Click **Finish** to install the application.
- 12.Save the configuration for WebSphere.
- 13.Click **Applications** → **Enterprise Applications** to display the installed enterprise applications.
- 14.Place a check mark next to the JMSSampleApplication entry.
- 15.Click **Start** to start the application.
- 16.Once the application starts, go to “Test the application” on page 409 to test the application.

Test the application

Follow the steps below to test the sample application.

1. Enter `http://localhost:9080/JMSSampleApplication/index.html`
If you are testing the sample application from another machine, replace the *localhost* hostname with the hostname of the application server machine.
2. Click **Click here to post a message**.
3. Enter Test Message 1 and click **Post Message**.
4. Click **Click here to browse messages on the queue**. Test Message 1 will be displayed.
5. Start wsadmin using `wsadmin -user username -password password`

Note: A valid user name and password will need to be supplied that has administrator access to WebSphere Application Server.

6. From the `wsadmin` command line, execute the following commands to list the users and groups that can access the bus.

```
$AdminTask listUsersInBusConnectorRole {-bus TEST_LOCAL_SIBUS}  
$AdminTask listGroupsInBusConnectorRole {-bus TEST_LOCAL_SIBUS}
```

7. Remove the AllAuthenticated group from the BusConnector role with the following command:

```
$AdminTask removeGroupFromBusConnectorRole {-bus TEST_LOCAL_SIBUS -group  
AllAuthenticated}
```

Note: Do not close the wsadmin console until instructed to do so since it will be used many times in the following steps.

8. Save the changes made by executing the following command:

```
$AdminConfig save
```

9. Do the following from the Administrative Console to restart the Messaging Engine for the application server:

- a. Click **Servers\Application Server**.

- b. Click the application server that is running the messaging engine to open the application server properties page.

- c. Click **Messaging engines**.

- d. Select the messaging engine by placing a check mark in the left column and click **Stop**.

- e. Once stopped, select the messaging engine again and click **Start**.

10. From the sample application, click **Click here to browse messages on the queue**. The following message should appear:

Security Exception occurred.
Access to bus denied

11. Execute the following commands to add AllAuthenticated back to bus connector role:

```
$AdminTask addGroupToBusConnectorRole {-bus TEST_LOCAL_SIBUS -group  
AllAuthenticated}  
$AdminConfig save
```

12. From the sample application, click **Click here to browse messages on the queue**. The message Test Message 1 should now be displayed.

13. From the **wsadmin** command line, execute the following command to list the groups that are in the default sender role for the bus. By default AllAuthenticated should be the only group listed.

```
$AdminTask listGroupsInDefaultRole {-bus TEST_LOCAL_SIBUS -role sender}
```

14. From the **wsadmin** command line, execute the following commands to remove the AllAuthenticated group from the default sender role.

```
$AdminTask removeGroupFromDefaultRole {-bus TEST_LOCAL_SIBUS -role sender  
-group AllAuthenticated}  
$AdminConfig save
```

15. From the sample application, click **Click here to post a message**.

16. Enter the text Test Message 2 and click **Post Message**.

"Your message was not posted to the Queue.
Security Exception occurred.
Access to queue denied"

17. From the sample application, click **Click here to browse messages on the queue**. The message Test Message 1 should now be displayed.

18. From the **wsadmin** command line, execute the following commands to remove the AllAuthenticated group from the browser default role.

```
$AdminTask removeGroupFromDefaultRole {-bus TEST_LOCAL_SIBUS -role browser  
-group AllAuthenticated}  
$AdminConfig save
```

19. From the sample application, click **Click here to browse messages on the queue**. The Security exception access to queue denied.

20. From the **wsadmin** command line, execute the following commands to add the AllAuthenticated group back to the sender default role.

```
$AdminTask addGroupToDefaultRole {-bus TEST_LOCAL_SIBUS -role sender -group  
AllAuthenticated}  
$AdminConfig save
```

21. From the sample application, click **Click here to post a message**.
22. Enter the text Test Message 3 and click **Post Message**. The message will post successfully.
23. From the sample application, click **Click here to browse messages on the queue**. The Security exception access to queue denied.
24. From the **wsadmin** command line, execute the following commands to add the AllAuthenticated group back to the browser default role.

```
$AdminTask addGroupToDefaultRole {-bus TEST_LOCAL_SIBUS -role browser  
-group AllAuthenticated}  
$AdminConfig save
```
25. From the sample application, click **Click here to browse messages on the queue**. The messages Test Message 1 and Test Message 3 will be displayed.
26. Exit the **wsadmin** command line by typing quit.

Note: Additionally, using the commands in 13.2.4, “Administering destination security” on page 318 the roles on the queue destination can be modified and tested rather than using the default roles that affect access to all bus destinations not just the LocalQueue destination.

Note: To test against WebSphere MQ, map the resource reference for jms/queue_sender to JMS Queue jms/queue_ForeignQueue and restart the application. The above steps will work for the MQ destination and bus as well.

Sample application for database connections

This section will detail the steps needed to create that DB2 sample database, configure IBM Rational Application Developer and WebSphere Application Server v6 to run the sample application. The required steps are:

1. Create the sample database.
2. Create a JAAS authentication alias for accessing the database.
3. Create a JDBC data source.

Additionally, it is assumed that a user janedoe has been created on the database server with password janedoe. This user ID will be used by the application to access the database.

Creating the sample database

For our sample application, we used the DB2 sample database from IBM DB2 Universal Database V8.1 Enterprise Edition with FP7a. For development and testing purposes, DB2 Personal or Workgroup Edition can be installed on the same machine as IBM Rational Application Developer (RAD).

The remainder of the chapter assumes that DB2 has already been installed, the database instance has been created, and if DB2 is not on the same machine as Rational Application Developer or WebSphere Application Server, communication between the machines has been properly configured.

To create the sample database do the following:

1. Log in to the database server using the instance id and password.

Note: On Microsoft Windows machines, the default instance id is db2admin. On Unix or Linux machines, the default instance id is db2inst1.

2. If DB2 is running on Microsoft Windows, open a DB2 Command Window by clicking **Start** → **Programs** → **IBM DB2** → **Command Line Tools** → **Command Window**.
3. From the DB2 Command Window if using Windows or command prompt if using Unix or Linux, start DB2 by typing **db2start**.
4. If DB2 starts successfully you will see a message as shown in Figure A-12. If DB2 is already running you will see a message as shown in Figure A-13.

```
10-25-2004 16:02:15      0  0  SQL1063N  DB2START processing was
successful.
SQL1063N  DB2START processing was successful.
```

Figure A-12 DB2 Started Successfully

```
10-25-2004 14:04:30      0  0  SQL1026N  The database manager is already
active.
SQL1026N  The database manager is already active.
```

Figure A-13 DB2 Already Started

5. Create the sample database by typing **db2samp**.
6. In order for the EJB from the sample application to work, a primary key needs to be defined on the employee table. Do the following to define the key:

- a. Connect to the sample database by typing db2 connect to sample. You will see a message as shown in Figure A-14.

```
Database Connection Information  
Database server      = DB2/NT 8.1.0  
SQL authorization ID = DB2ADMIN  
Local database alias = SAMPLE
```

Figure A-14 Connected to sample database

- b. Type the following command to create the primary key. Replace *user_id* with the user ID of the DB2 instance owner (usually db2inst1 or db2admin).

```
db2 alter table user_id.employee add primary key (empno)
```

- c. You will see a message as shown in Figure A-15.

```
DB20000I The SQL command completed successfully.
```

Figure A-15 Primary key created

7. If DB2 is running on Windows the tables are created in the db2admin schema. In order for the sample application to work an alias must be created for each table in the db2inst1 schema. Place the create_aliases.sql file, found with the DB2 sample application, in a directory. Then run the following command:

```
db2 -tvf path_to_file\create_aliases.sql
```

Note: This command will process the contents of the create_aliases.sql file that contains the commands to create an alias for each table in the db2inst1 schema. The create_aliases.sql file can be found with the sample DB2 application.

8. You may now log out of the instance ID.
9. If the database is not on the same machine as Rational Application Developer or WebSphere Application Server, you will need to catalog the database locally on the machine(s) if using the DB2 type 2 JDBC driver. For simplicity, we will use the type 4 driver. Refer to the DB2 Information Center article titled “Cataloging the sample database” for further information.

Defining the J2C Authentication Alias

In order to use Container Managed Persistence or Container based authentication, a J2C Authentication Alias needs to be defined and associated with the JDBC Resource Reference. The next steps will describe how to define the J2C authentication alias using the administration console in either WebSphere Application Server or Rational Application Developer. The sample application contains an EJB that uses Container Managed Persistence so a J2C Authentication Alias is required for accessing the sample database.

To define the J2C Authentication Alias, do the following:

1. Make sure that WebSphere Application Server or Rational Application Developer's WebSphere Application Server V6 Test Environment is started.
2. Open the Administrator Console using a Web browser.

Note: The URL for the Administrator Console is
`http://servername:9060/admin`

Replace `servername` with the name of the server running WebSphere Application Server V6.

Note: If you are targeting the Rational Application Developer WebSphere Test Environment, you can right-click the server in the Servers view and select **Run Administration Console**.

3. Expand Security in the left navigator tree by clicking +.
4. Click **Global Security**. The Global Security settings will be displayed as shown in Figure A-16 on page 415.

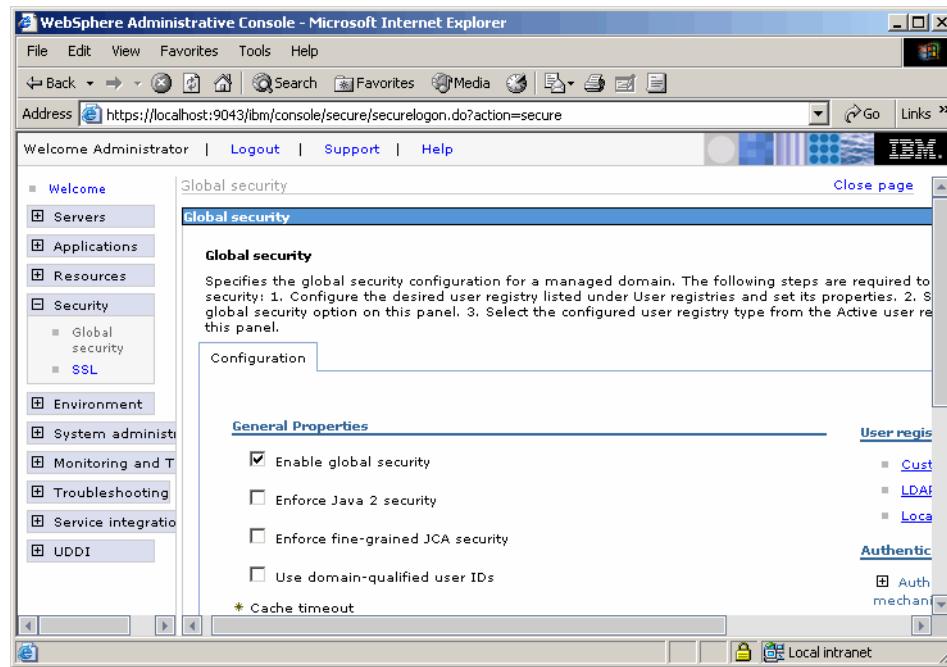


Figure A-16 Global Security

5. Under Authentication, click + next to JAAS Configuration to expand the list.
6. Click **J2C Authentication data** from the expanded list. The J2C Authentication data entries pane will be displayed.
7. Click **New** to create a new J2C Authentication Alias.
8. Fill in the values for the new J2C Authentication Alias as shown in Figure A-17 on page 416. For the sample application, use:

Table A-9 J2C Authentication alias

Field Name	Field Value
Alias	sample_db_user
User ID	janedoe
Password	janedoe
Description	User ID to connect to sample DB

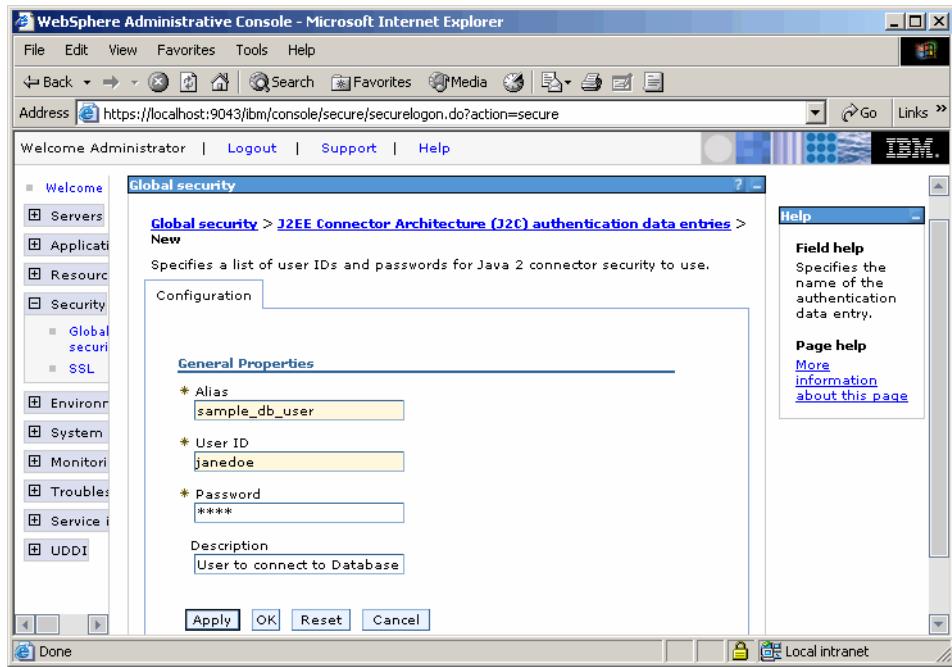


Figure A-17 New J2C Authentication Alias

9. Click **OK**. You will be returned to the J2C authentication data entries pane and will get a message stating that changes have been made and need to be saved.
10. Write down the alias name created, it may be needed in 15.3.1, “Defining the Resource Reference” on page 355.
11. Save the configuration for WebSphere.

Note: The ability to define Deployment characteristics for an Enterprise Application in the ear file is new to WebSphere Application Server/Rational Application Developer V6. JDBC data sources and JAAS authentication aliases, among other things, can be defined in the deployment descriptor for the ear for use during testing and deployment. These settings can be modified during deployment using the Administrative Console.

Defining the JDBC data source

The sample application requires a JDBC data source to connect to the sample database created in 15.2, “Defining a JDBC data source” on page 348. The

following details the steps to create the JDBC data source in both the EAR deployment descriptor and using the Administrative Console.

Creating the JDBC data source using the Administrative Console

1. Follow steps 1 through 6 from the previous section.
2. For step 7 select the following values:
 - Database type: DB2
 - Provider type: DB2 Universal JDBC Driver
 - Implementation type: XA data source
3. Click **Next**.
4. Enter Sample DB2 Universal XA Provider in the Name field.
5. Click **OK**.
6. From the provider list click **Sample DB2 Universal XA Provider**.
7. On the right side of the page click **Data sources**.
8. Follow steps 9 through 12 using the following values:
 - Name: Sample data source
 - JNDI Name: jdbc/sample
 - Description: Sample database data source
 - Database name: sample
 - Driver type: 4
 - Servername: *hostname of the server running DB2, or localhost if running on same machine.*
 - Port number: Port number that DB2 is listening on, default port is 50000.

Creating the JDBC data source in EAR

1. Follow steps 1 through 3 from the previous section.
2. Select **IBM DB2** for the database type.
3. Select **DB2 Universal JDBC Provider (XA)** for the JDBC provider type.
4. Click **Next**.
5. Enter Sample DB2 Universal XA Provider in the name field.
6. Click **Finish**.
7. Select **Sample DB2 Universal XA Provider** from the JDBC provider list.
8. Click the **Add** button next to the Data source list.
9. Select **DB2 Universal JDBC Provider (XA)** as the JDBC provider type.

10. Select **Version 5.0 data source**.

11. Click **Next**.

12. Enter the following values for the data source:

- Name: Sample data source
- JNDI Name: jdbc/sample
- Description: Sample database data source
- Container-managed authentication alias:

13. Click **Next**.

14. Select **databaseName** and enter sample in the Value field.

15. Select **driverType** and enter 4 in the Value field.

16. Select **serverName** and enter the *hostname* of the server running DB2 or localhost if running on the same machine in the Value field.

17. Select **portNumber** and enter the port that DB2 is listening on. Default is 50000.

18. Click **Finish**.

19. Save and close the file.

Note: If not already done, you will need to modify the WebSphere Variables for DB2 Universal JDBC Driver Path and DB2 JDBC Driver Path to point to the *DB2_INSTALL_DIR/java* directory.

Also, when running WebSphere Application Server on a Unix or Linux machine and using the JDBC type 2 driver, you will need to modify the environment for the user running WebSphere, usually root. The modification is needed to identify certain required properties to find the local DB2 instance that the type 2 driver will use to access local or remote databases. Add

```
. /home/instance_owner/sql1lib/db2profile
```

to the user's profile if using the ksh shell, or to .bashrc if using the bash shell. For the tcsh or csh shells, add

```
. /home/instance_owner/sql1lib/db2cshrc
```

to the user's .tcshrc file or .cshrc file.

Make sure that the user running WebSphere Application Server, usually root, is part of the DB2 instance group.

Once changes are made, you must logout and then login to pickup the changes and then start WebSphere.

Importing and starting the sample application in Rational Application Developer

In this section, we provide details of importing the sample application to the development environment.

Import the application

1. Start Rational Application Developer.
2. Click **File** → **Import**.
3. Select **EAR file**.
4. Click **Next**.
5. Click **Browse**.
6. Locate the **DB2SampleApplication.ear** file in the Open window and select it.
7. Click **Open**.
8. Click **Finish** to import the ear.

Start the application

The process detailed below assumes you have a WebSphere V6 Test Environment (WTE) already configured in Rational Application Developer. If a WebSphere Test Environment is not already configured refer to Appendix A, “Additional configurations” on page 387.

1. Click + to expand Enterprise Applications.
2. Right-click the sample application and select **Run** → **Run on Server**
3. Choose an existing WebSphere Application Server V6 (WTE).
4. Click **Finish**.
5. Wait for Server to start.
6. The sample application is now ready to be tested.

Install the sample application into WebSphere Application Server

1. Open the Administrative Console.
2. Click + next to Applications
3. Click **Install New Application**
4. Click **Browse**
5. Locate and select **DB2Sample.ear**.
6. Click **Open**.

7. Click **Next**.
8. Scroll down and click **Next**.
9. Scroll down and click **Next**.
10. Click **Next** to accept the Server assignments for the EJB and WAR.
Otherwise modify the assignments and then click **Next**.
11. For Step 3, click **Next** to accept the default backend for the EJB.
12. For Step 4, click **Next** to accept the defaults.
13. For Step 5 do the following then click **Next**:
 - a. Place a check mark in the box next to the EJB Module.
 - b. Scroll to the top of the page and select **jdbc/sample** from the Specify existing Resource JNDI name list.
 - c. Click the **Apply** button next to the Specify existing Resource JNDI name list to apply the change to the EJB Module.

Note: When done with the above steps, the EJB Module should show **jdbc/sample** in the JNDI name field and the Resource authorization should show Container for the Resource authorization.

14. For Step 6, click **Next** if the EJB should already shows **jdbc/sample** for the JNDI name. Otherwise follow steps a through c from above to change the JNDI name.
15. For Step 7, click **Next** to use the default JNDI name for the EJB reference to bean mapping.
16. For Step 8, click **Next** to accept default mapping of JNDI name to **jdbc/sample** for the Resource References.
17. Click **Step 12** to skip to step 12 of the installation process or step through the remaining steps accepting the defaults by clicking **Next** for each step.
18. Click **Finish** to install the application.
19. Save the configuration for WebSphere.
20. From the list of Enterprise Applications place a check mark in the select column for the DB2SampleApplication.
21. Click **Start** to start the application.
22. Once the application is started, showing a green arrow in the status column, the sample application is ready for testing.

Testing the sample application

1. Open the following URL if testing in Rational Application Developer WebSphere Test Environment:

<http://localhost:9080/DatabaseSampleApplication/index.html>

2. Open the following URL if testing in WebSphere Application Server:

<http://servername:port/DatabaseSampleApplication/index.html>

Note: Port is optional depending on your WebSphere Application Server setup. If IBM Http Server (IHS) has been installed along with WebSphere Application Server, configured to use the WebSphere plugin, and the plugin configuration was regenerated during the install of the sample application then :port is not needed. Otherwise, you should specify the HTTP listener port of the application server that is running the sample application, port 9080 is the default port for a single application server install.

3. If Global Security is enabled for the application server, you will be prompted for a valid user ID and password.
4. To test Container-managed authentication to the database, do the following:
 - a. Select **EMPLOYEE** from the Table Name list.
 - b. Select **Container** from the Authentication Type radio button list.
 - c. Click **Display** to show the contents of the Employee table.

Note: This test calls the QueryTable servlet with the parameters tableName=EMPLOYEE and auth=Container. The servlet uses the jdbc/sample_container_managed authentication alias to connect to the database and display the table data.

- d. Scroll down and click **Home** to return to the index page.
 - e. Repeat process for other tables in list if you like.
5. To test Application-managed authentication to the database do the following:
 - a. Select **EMPLOYEE** from the Table Name list.
 - b. Select **Application** from the Authentication Type radio button list.
 - c. Click **Display** to show the contents of the Employee table.

Note: This test calls the QueryTable servlet with the parameters tableName=EMPLOYEE and auth=Application. The servlet uses the jdbc/sample_application_managed authentication alias to connect to the database and display the table data.

- d. Scroll down and click **Home** to return to the index page.
 - e. Repeat process for other tables in list if you like.
6. To test the EJB, do the following:
- a. Select an employee number from the Employee Number list.
 - b. Click **Lookup** to display the employee info from the database using an EJB Container Managed Persistence entity bean.

Note: This test calls the Employee_Lookup servlet with the parameter empID=xxxxxx (xxxxxx = employee ID number). The servlet then obtains the employee information from the database via a EJB Entity Bean. Since the EJB bean is a Container Managed Persistence bean, the container handles all access to database including authentication.

- c. Scroll down and click **Home** to return to the index page.
- d. Repeat the process for other employees if you like.



B

Additional material

Appendix B

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/SG246316>

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, SG246316.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
sg246316.zip	Zipped Code samples for Itsohello Application clients.

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 20 MB minimum
Operating System: Windows/Linux
Processor: P4 2.x GHz or faster
Memory: 1GB or more

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.

Abbreviations and acronyms

IBM	International Business Machines	HTTP	HyperText Transfer Protocol
ITSO	International Technical Support Organization	IHS	IBM HTTP Server
ACL	Access Control List	IIOP	Internet Inter-ORB Protocol
AM	Access Manager	IOR	Interoperable Object Reference
API	Application Programmer Interface	IP	Internet Protocol
BA	Basic Authentication	IT	Information Technology
CA	Certificate Authority	ITSO	International Technical Support Organization
CLR	Certificate Revocation List	JAAS	Java Authentication and Authorization Service
CMP	Container Managed Persistence	JACC	Java Authorization Container Contract
CORBA	Common Object Request Broker Architecture	JAR	Java ARchive
CPU	Central Processing Unit	JCA	Java Cryptography Architecture
DD	Deployment Descriptor	JDBC	Java DataBase Connection
DNS	Domain Name Server	JKS	Java Key Store
DRS	Data Replication Service	JMS	Java Messaging Service
EAR	Enterprise ARchive	JMX	Java Management eXtensions
EIS	Enterprise Information System	JNDI	Java Naming and Directory Interface
EJB	Enterprise Java Bean	JNI	Java Native Interface
ERP	Enterprise Resource Planning	JRE	Java Runtime environment
FIPS	Federal Information Processing Standards	JSP	Java Server Pages
FTP	File Transfer Protocol	JSR	Java Specification Request
GIOP	General Inter-ORB Protocol	JSSE	Java Secure Socket Extension
GSO	Global Sign On	JVM	Java Virtual Machine
GUI	Graphical User Interface		
HTML	HyperText Markup Language		

KDC	(Kerberos) Key Distribution Center	SWAM	Simple WebSphere Authentication Mechanism
LDAP	Lightweight Directory Access Protocol	TAI	Trust Association Interceptor
LDIF	Lightweight Directory Interchange Format	TAM	Tivoli Access Manager
LTPA	Lightweight Third Party Authentication	URL	Uniform Resource Locator
MQ	Message Queuing	WAR	Web ARchive
ND	Network Deployment	WAS	WebSphere Application Server
NIS	Network Information Service	WTE	WebSphere Test Environment
OMG	Object Management Group	XML	eXtensible Markup Language
ORB	Object Request Broker		
OS	Operating System		
OU	Organizational Unit		
PAM	Pluggable Authentication Modules		
PKI	Public Key Infrastructure		
RAD	Rational Application Developer		
RMI	Remote Method Invocation		
RSA	Rivest, Shamir and Adleman (algorithm)		
SIB	Service Integration Bus		
SMTP	Simple Mail Transfer Protocol		
SOA	Service Oriented Architecture		
SOAP	Simple Object Access Protocol		
SPNEGO	Simple and Protected Negotiate		
SQL	Structured Query Language		
SSL	Secure Socket Layer		
SSO	Single Sign On		

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 428. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Base Administration Guide, IBM Tivoli Access Manager V5.1*, SC32-1360
- ▶ *WebSEAL Administration Guide, IBM Tivoli Access Manager V5.1*, SC32-1359
- ▶ *Enterprise Security Architecture Using IBM Tivoli Security Solutions*, SG24-6014
- ▶ *IBM Tivoli Access Manager for e-business*, REDP3677
- ▶ *WebSphere Security V5*, SG24-6573
- ▶ *Security Basics for WebSphere*, REDP-3944

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ WebSphere Application Server V6 Information Center
<http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp>
- ▶ WebSphere Application Server - prerequisites
<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>
- ▶ OMG's XMI Web site
<http://www.omg.org/XMI>
- ▶ Apache Web server documentation: htaccess
<http://apache-server.com/tutorials/ATusing-htaccess.html>
- ▶ IETF's Web site, RFC2617
<http://www.ietf.org/rfc/rfc2617.txt>

- ▶ OASIS's Web site
<http://www.oasis-open.org>
- ▶ Specification: Web Services Security (WS-Security)
<http://www-106.ibm.com/developerworks/webservices/library/ws-secure>
- ▶ WebSphere MQ Web site
<http://www.ibm.com/software/ts/mqseries/messaging>
- ▶ Sun's J2EE Web site
<http://java.sun.com/j2ee>

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

.arm file 104

A

Access check 228

Access control 120

access control 50, 347

access control directives 72

Access control list 221

access control list 219

Access decisions

Enterprise beans 277

Web Resources 278

Access Manager

Authorization Engine 221

Authorization Server 215

aznAPI 262

Client configuration 228

external JACC provider 284

J2EE Security 262

JACC 228

Lab environment 222

management objects 220

Migration 265

plug-in 215

Policy database 226

Policy Server 213

role 223

security model 218

User registry 219

Web Portal Manager 215

WebSEAL 214

WebSphere integration 216

Access Manager Client 224

Access Manager credential information 263

Access Manager Integration 261

Access Manager Java Runtime 224

Access Manager Secure Domain 213

ACE/Server 233

ACL 189, 219, 221

Act as part of the operating system 18

Active Directory 213

ActiveX application client 154

Add Security Constraints 88

Additional CORBA configuration 164

Alias 313

All authenticated 62

All authenticated users 378

AllowOverride 72

alternate name 313

AMJRTE 224

Applet application client 154

Application Client runtime 154, 164

Application clients 154

application deployment 62

application installation 62

application login module

new 57

Application Logins 57

application scenarios 4

application server instance 381

attribute layer 157

auditing 290, 297

auth_module 70

Authentication 314, 323

Basic authentication 80

Client certificate 80

Digital certificate 69

Form-based 80

authentication 290, 297

authentication and authorization APIs 214

authentication challenge 101

Authentication Configuration 161

authentication layer 157

authentication mechanism 73, 85, 171

authentication process 8

authentication strategy 57

Authentication Token 191, 200

Authentication token 199

Authorization 314, 324

Web server 72

authorization 290, 297

authorization API 218

authorization constraint 85

authorization constraints

configuration 89

Authorization Rules 222

Authorization Server 215
Authorization servers 285
authorization service 9
Authorization table support 228
Authorization Token 191, 200
Authorization token 193
AuthorizationToken 195
AuthzPropTokenFactory 200
availability 291
aznAPI 218, 262

B

Base DN 11
Basic Authentication 9, 69, 147
 logout 83
Basic authentication 78
basic authentication 300
 test 71
Bean level delegation 133
Bean Managed Persistence 361
Bind DN 11
BMP 361
buildClientRuntime 168
Bus Destination 313
bus destination roles 315
BusConnector role 314
business logic 120

C

Callback handler 50
CallbackHandler 172, 174
CCI 336
certificate
 import 104
certificate authentication with LDAP 107
certificate authentication with LocalOS 107
certificate based client authentication 103
Certificate filter 14
Certificate map mode 14
certification agency 304
Client Authentication 75
client certificate 9
Client Certificate Authentication 147
Client certificate authentication options 150
client example 154
Client ORB 158
client security 388
Client Security Enablement 160

client side certificate 105
CLIENT_CERT 108
client-certificate 73
ClientContainer 57, 172
client-side certificate 103
client-side programmatic login 174
CMP 362
CMS 67
CMS keystore 104
cn=root 11
Common Client Interface 336
Common Object Request Broker Architecture 9
Common Secure Interoperability 156, 204
Common Secure Interoperability Version 2 8
Common Security Interoperability 146
communication channel 66
communication types 66
Component Managed Authentication 338
confidentiality 290, 297, 302
Configuration
 Exact DN mapping 117
 Local OS User registry 19
 SSL 73, 75
 SSL for LDAP 14
 User registry 15
 WebSEAL form based authentication 240
configuration
 custom login module 56
Connection Factory 338
Connection Object 338
connection-based transport 157
Container Authentication 147
container authentication
 configuration 147
Container Contract 276
Container contract 273
Container Managed Authentication 339–340
Container Managed Persistence 362
Container Settings 75
Container-based authentication 358
content integrity 85
cookie 81
CORBA 9
CORBA configuration file 169
CORBA ConfigURL 160
CosNaming
 Roles 43
CosNaming with a qualified name 170
CosNaming with unqualified name 170

Creating a new profile 381
credential list attribute 172
credential token 9, 171
Credentials 188
Cross Domain Single Sign-On 235
csec_locales 18
CSlv2 8, 145, 156
CSlv2 add-on authentication protocol 162
CSlv2 inbound 148
CSlv2 inbound authentication 208
CSlv2 Inbound Transport 180
CSlv2 Security Attribute Service 157
custom attributes 189, 194, 198
Custom Authorization Token 194
Custom Authorization Token Implementation 195
Custom callback handler 51
Custom CallbackHandler 178
custom encryption 192
Custom JAAS login 50
Custom Login Configuration 340
custom Login Form 82
Custom login module 51
custom login module
 configuration 56
Custom principal 55
Custom Propagation Token
 Implementation 198
Custom Single Sign-On Token 196
custom token 192–193
Custom User Registry 8, 20
 DB2 26
 sample 23
Custom user registry
 Development 23
CustomLoginModule.java 52
CVS 368

D

Data Constraint 85
database
 authentication types 344
 Resource Reference 355
 Securing access 347
Database connection security 344
database connections
 sample application 411
Database security 344
DB2 Custom User Registry 26

DB2 Legacy CLI-based Type 2 Provider 345
DB2 libraries 26
DB2 Universal JDBC Provider 345
DB2UserRegistrySample 26
DB2UserRegistrySampleTest 27
decision-making server 215
Declarative security 122
declarative security 83
Default Authorization Token 194
Default Messaging 400
Default Method 340
Default Propagation Token 197
default search settings 13
Default Single Sign-On Token 195
default token 192
DefaultPrincipalMapping 57, 340
Delegation Policy 121
delegation policy 139
Deployment
 application 62
deployment descriptor 81
Deployment descriptor mapping 265
Deployment tools contract 271, 276
desktop Single Sign-On 233
Destination Security 318
development environment
 Linux 370
 Windows 369
Digest authentication 98
Digital certificate authentication 69
Directory directive 70, 98
Distinguished Name 109
Distributed Replication Service 201
DN 109
doAs() 173
doAs() method 58
Downstream Propagation 208
Downstream propagation 204
Downstream propagation scenario 205
DRS 201
DRS Replication Domain 201
dummy password 236
dumpNameSpace 167
Dynacache 201
dynamic attributes 189
Dynamic Module Updates 284
dynamic resources 87
Dynamic Web Projects 81

E

Eavesdropping 292
e-business infrastructure 217
e-Community Single Sign-On 235
EIS systems 336
EJB 120, 154
 Declarative security 122
 method access control 128
 Method level delegation 136
 method permissions 129
 security roles 122
EJB authenticator 9
EJB container access security 145
EJB modules 122
EJB persistence 360
EJB Policy Context Identifier 272
EJB programmatic security
 Sample code 145
EJB security methods 144
ejb-jar.xml 86, 134
embedded HTTP Server 73
Embedded Messaging
 Configuration 393
Embedded Messaging Security 313
Embedded Tivoli Access Manager 264
embedded Tivoli Access Manager
 disable 286
Embedded Tivoli Access Manager Client 224
Enable embedded Tivoli Access Manager 285
enable SSL 105
encryption 292
enhanced TAI Interface 243
enhanced TAI++ 188
Ensure all unprotected 62
Enterprise application security 377
Enterprise Java Beans 120, 154
Entity Beans 120
EPAC 263
Everyone 62, 378
Everyone role 159
Exact Distinguished Name 117
Exact DN mapping 117
Exchange certificates 74
Exclude 132
externalized security 212

F

File based registry

testing 25

File based user registry 23
FileRegistrySample 23
filter 238
Foreign 313
Foreign Bus 313
Foreign Destination 332
Foreign Service Integration Bus security 326
Foreign System Integration Bus 403
Form based
 Logout 83
Form login
 configuration 81
Form-based authentication 80, 240
form-based login 80

G

General Inter-ORB Protocol 157
GET method 87
getCallerPrincipal() 120, 144
getConnection() 339
getUniqueID() 193
GIOP 157
Global Security 80
global security 300, 374
Global Sign-On 235
group 62
Group Filter 14
Group ID Map 14
Group member ID map 14
groupDisplayName 21
groupSecurityName 21
groupUniquelD 21
GSO 218
gso 238

H

Horizontal Propagation 201
 Dynacache 201
Horizontal propagation
 JMX 202
htaccess 72
HTML pages 66, 87
HTTP 304
 BA 69
 Basic Authentication 69
 digest authentication 69
 Method security 87

HTTP cookie 81
HTTP Digest Authentication 97
HTTP methods 85
HTTP plug-in 73, 77
HTTP transport 78
http_plugin.log 78
httpd.conf 67, 98
HTTPS 67
HTTPS Information 111

I

IBM HTTP Server
 Certificate 103
 logs 106
 SSL 67
IBM Tivoli Directory Server V5.2 9
ibm_security_logout 83
ibm_ssl_module 67
ibm-ejb-jar-ext.xmi 137
identification 290, 297
Identity assertion 189
Identity propagation
 definition 189
ignore 238
IIOP 80, 156
IIOP over SSL 180, 182
IIOP over TCP/IP 181
ikeyman 15
ikeyman tool 103
import certificate 15, 104
inbound transport 147
inetOrgPerson 235
InfoCenter 6
Information Center 6
Initial Login 189
initial login 188
installation
 applications 62
Integrity 301
integrity 290, 292, 297, 301
Inter-engine transport 316
internal login 57
Internet Inter-ORB Protocol 156
Interoperability Mode 207
Interoperable Object Reference 158
introduction 4
IOR 158
iPlanet 213

isCallerInRole() 120, 144
isolate roles 86
ITSObank application 80
Itsohello application 388
iv-creds 236
iv-groups 236
iv-user 236

J

j_password 82
j_security_check 81
j_username 82
J2C 336
J2C Authentication Alias 414
J2C Authentication data 58
J2EE application client 154
J2EE Connector Architecture 336
J2EE Connector security 337
J2EE programmatic security 93
J2RE 168
JAAS 50, 93, 122, 188, 216
 Callback handler 50
 Login module 51
 Principal 55
JAAS Authentication 339
JAAS authentication alias 58
JAAS authentication entries 379
JAAS configuration 50
JAAS framework 263
JAAS login 54
JAAS login module 171, 193
 authentication alias 59
JAAS login sequence 54
JAAS programmatic login 177
JAAS Subject 171, 189
JAAS subject 9
JACC 218, 270
 Sample 287
 WebSphere 275
 WebSphere Extensions 280
JACC Access Decisions 276
JACC policy context identifiers 279
JACC policy propagation 280
JACC provider 223
JACC sample 391
Java 2 Runtime Environment 168
Java 2 security 50
Java Authentication and Authorization Service 93

Java Authentication and Authorization Services 50, 216
Java Authorization Contract for Containers 270
Java Build Path 26
Java client authentication protocol 156
Java client configuration 160
Java Database Connectivity 154
Java Management Extensions 203
Java Message Service 154
Java Native Interface 154
Java Server Pages
 JSP 84
JCA 336
JDBC 154
JDBC Connection 358
JDBC data source 348
 define 349, 416
 EAR 352
JDBC data source provider 345
JDBC type 2 driver 346
JDBC type 4 driver 346
JMS 154, 400
JMS clients
 Application clients 308
 Message-Driven Bean 308
JMS messaging services 310
JMS Objects
 define 400
JMS objects 332
JMS provider 310
JMX 203
JNI 154
JSPs 66
JSR 115 270
JSSE Repertoire 74
junction
 configuration 236

K

Key configuration 15
key database 77
key store 67
KeyFile 68
KeyStore 169

L

launchClient 164
LDAP 10, 226

Advanced configuration 13
Certificate Filter 108
Configuration 100
Configuring SSL 14
Reuse Connection 11
SSL 14
Test 13
Test SSL 17
LDAP authentication
 test 101
LDAP Configuration
 ldap.sth 100
LDAP configuration 10
 Advanced 13
LDAP keystore 15
LDAP module trace 102
LDAP server 11, 99
LDAP server certificate 15
LDAP Settings 13
LDAP User Registry 8
LDAP User registry 9
LDAP user registry
 Test 13
ldap.prop 99
Lightweight Directory Access Protocol 226
Lightweight Third Party Authentication 9, 190
LoadModule 67, 70, 100
Local OS 19
Local OS User Registry 8
Local OS User registry 17
Local OS user registry
 test 19
local replica 226
Log On As 18
Log on as a service 18
logical roles 86
Login Form 81
Login module 51
login module 171
Login process 173
login sequence 54
login-config 108
LoginContext 172
LoginModule 188
LoginModule interface 51
Logout 83, 113
Lower administration 213
LTPA 9, 81, 190, 199
LTPA cache 261

LTPA cookie 218
LTPA token 255
LTPAToken 191
LtpaToken2 195
LTPAToken2Factory 200

M

Mapping
 Administrator role to group 41
 Administrator role to user 41
 CosNaming role to user 44
marker interfaces 191
Master authorization database 219
MCA 308
Message Channel Agent 308
Message Layer authentication 147
message layer authentication options 149
Message level security 292
Message-driven Beans 120
Messaging
 Access Control 309
 Authentication 308
 Authorization 309
 Confidentiality 309
 Data integrity 309
 Non-repudiation 309
 Security Services 308
messaging
 sample application 402
Messaging Engine 312
Messaging engine inbound transports 316
method access control 128
Method level delegation 136
method permissions 129
method-level delegation policies 142
MQ client link 310
MQ Link 322, 330, 404
multi-phase negotiation 243
multiple profiles 381
mutual SSL 75, 245

N

NameServiceServerRoot 168
netstat 17
Network Identity 212
Network Information Service 18
Network security 370
new application login module 57

new test server 371
NIS 18
non-repudiation 290, 297
non-secure HTTP 77

O

OAM 324
Object Authority Manager 324
Object Management Group 156, 204
Object Request Brokers 156
OMG 156, 204
Operating System access control 368
ORB 156
ORB object 167
outbound transport 147
overhead 293

P

PAM 50
pctLinux 381
pctWindows 381
pdadmin 215
PDLoginModule 228, 263
PDPPermission 218, 263
Perform nested group search 14
Performance 293
personal certificate 103
PKCS12 103
Pluggable application client 154
Pluggable Authentication Module 50
plug-in configuration 77
plug-in file 77
plugin-cfg.xml 77
Policy Context 272
Policy Context Identifier 272
Policy Server 213
Policy server 285
policy store 215
POP 219, 221
port
 443 68
POST method 87
Principal 55
principal 96
principal-based authorization 50
PrivilegedAction 58
Programmatic J2EE security 144
Programmatic Login 171

Programmatic login
 server-side 122
programmatic login
 client-side 174
Programmatic security 92
programmatic security
 sample 95
Programming authentication 58
Programming authorization 58
Propagation Login 189–190
Propagation Token 191, 200
Propagation token 196
PropagationToken 197
protected 85
Protected object policies 221
protected object policies 219
protected object space 219
protocol 298
Provider Contract 276
Provider contract 274
Proxy LoginModule 50
public keys 73–74

Q

Queue 313
Queue destination
 define 397
Queue Manager 321

R

Rational Application Developer 26, 368
Rational Clear Case 368
Redbooks Web site 428
 Contact us xvi
Request Consumer 300
Request Generator 300
Required privileges 17
res-auth 338
resource collection 85
Resource name 88
Resource Reference 339
RMI/IOP 145
RMI/IOP Authentication Protocol 160
RMI/IOP transport channel protection 150
RMI_INBOUND 193, 205
RMI_OUTBOUND 205
Role Link 86
Role mapping 62

role name 86
RoleConfiguration 275
RoleConfiguration interface 228
RoleConfigurationFactory 275
roles 85
root authority 18
RSA Authorization API 233
RSA SecurID token authentication server 233
Run As Caller mode 134
run as server 137
Run-As delegation policy 133
Run-As mapping 62
Run-as mapping 140
Run-as Mode 133
Run-As Mode Mapping 121
run-as role mapping 133
RunAsRole 134
RunAsRole security role 134

S

sample application 5
sample configuration 5
SAS 145, 204
SAS (IBM) 157
SAS add-on authentication protocol 162
sas.client.props 160
Scenario 4
Secure Association Service 204
Secure Authentication Service (IBM) 157
secure client 166
Secure Domain 213
Secure Socket Layer 66
Secure Sockets Layer client certificate authentication 147
secure thin client 171
Securing connection 179
security 291
 enterprise 290
 global 300
 token 300
 transport channel 304
security and authorization constraint 91
Security Attribute 189
security attribute propagation 206
Security Attribute Service 145
Security Authentication 8
security aware 92
security challenge 166

Security constraints 85
security constraints
 configuration 88
Security Identity 135
security methods
 sample 93
Security role reference 86
Security role references 123
security role references 121
Security roles
 Web module 84
securityMechanism 347
security-role-ref 86, 124
self-signed certificate 73
 create 74
server creation wizard 372
Server Id 10
Server ORB 158
Server Password 11
Server perspective 370
server Status 370
service context 157–158
Service Integration Bus 310
 Connection to WebSphere MQ 329
 Integrating with MQ 322
Service Integration Bus Security 317
Servlet
 getRemoteUser() 93
 getUserPrincipal() 93
 isUserInRole() 93
Servlet Policy Context Identifier 272
Servlet Policy Enforcement 273
servlet security 93
Servlet security methods
 Sample code 94
servlet security methods 93
servlets 66
Session Beans 120
Sevice Integration Bus 312
Simple junctions 235
Simple WebSphere Authentication Mechanism 9
Single Sign-On 81
Single Sign-On junctions 238
Single Sign-On Token 191, 200
Single Sign-On token 195
SingleSignonToken 195
SMTP 304
Snoop 78
Snoop servlet 78
SOAP binding 294
SOAP Message Security 295
SOAP/HTTP transport level security 304
SPNEGO protocol 233
Spoofing 292
SSL 66–67, 78, 156
 LDAP 14
 Test 68
 Testing 78
 Web container 75
 Web server and WebSphere 73
SSL certificate 103
SSL Configuration 162, 180
SSL configuration 67
SSL entry 74
SSL handshake 113
SSL Inbound Channel 76
SSL module 67
SSL repertoire 73
SSL Settings 151
SSLEnable 68
SSO 81, 195
stateful security context 159
stateless security context 159
static content 66
static resources 87
strong private key protection 105
Subject 188
SubjectDN 109
supply 238
SvrSslCfg utility 252
SWAM 9
System capacity 293
System Integration Bus
 Define 394
System Logins 57

T

TAI 218, 236, 242
TAIResult 243
Tampering 292
TDS 9
Test
 LDAP 13
 User registry 19
test server
 new 371
Testing

Client Certificate 110
SSL 78
Testing SSL 68
Thin application client 154, 167
thin application client
 running 168
Thin Java application client 177
Timestamp 201
Tivoli Access Manager 188, 212
 for Business Integration, 306
Tivoli Directory Server 213
Tivoli Global Sign-On 218
Token Factory 199
Token framework 190
tokens 189
topic roles 315
Topic Space 313
Topic Space roles 320
trace 116
Transport 77
transport
 channel security 304
transport chain 75
Transport channel 66
transport channel 79
Transport channel encryption 146
Transport guarantee
 Confidential 91
 Integral 91
 None 91
 Web module
 Transport guarantee 91
transport layer 85
Transport level security 292, 304
Transport Security
 Confidentiality 316
Transport security 325
transport security 66
Transport security options 151
Trust Association Interceptor 218, 236, 242, 246
Trusted connection 245
trusted relationship 242
Trusted user 245
TrustStore 169

U

unauthenticated credential 159
Uncheck 132

Unchecked 129
unchecked 62, 132
uniqueIdentifier 109
uniqueUserId 21
UNIX Required privileges 18
unprotected methods 131
unsecure client 165
unsecure thin client 169
URI-based access control 216
URL bindings 88
URL definition 89
URL patterns 85
Use identity assigned to specific role 139
Use identity of caller 138
Use identity of EJB server 138
user 62
User Data Constraint 86
User Filter 14
User ID map 14
User Registries 10
user registries 8
User Registry
 Custom 8, 20
 File based 23
 LDAP 8
 Local OS 8, 17
User registry
 configuration 8
 LDAP 9
user registry 70, 215
User role 86
user-based authorization 50
User-defined objects 220
userDisplayName 21
UserRegistry interface 20
 method list 21
Users/Groups 378
userSecurityName 21

V

virtual host 68

W

WCInboundDefaultSecure 76
Web application 84
 client certificate 107
Web application module 88
Web applications 66

JDBC connection 355
Web Archive file 87
Web authenticator 9
Web browser security 67
Web clients 9
Web components 66
Web Container 73
 SSL 75
Web container 79, 84
 Authentication 80
 Web container 80
 SSL 75
 Testing SSL 78
Web module 84
 Authentication method, Authentication method 84
 Basic authentication 80
 Client certificate authentication 80
 Form based logout 83
 Form-based authentication 80
 Security roles 84
Web objects 220
Web Portal Manager 215
Web proxy authentication server 242
Web resources 85
Web security
 Options 97
Web Server
 Basic authentication 70
 LDAP authentication 100
Web server 78
 .htaccess 72
 Authentication 69
 Authorization 72
 Client certificates 105
 Configuration 100
 Configuration files 99
 ldap.sth 100
 Plug-in file 77
 SSL 73, 250
Web server certificate 250
Web Server definition 77
Web server security 67
Web service security 291
Web services
 security
 model 297
 security model 298
web.xml 86

WEB_INBOUND 193
WebSEAL 209
 Basic Authentication 238
 Basic authentication 229
 Client certificate-based authencation 231
 Form-based authentication 230
 HTTP header authentication 233
 Kerberos authentication 233
 LTPA 255
 SPNEGO Authentication 233
 TAI 242
 Token Authentication 233
WebSEAL Authentication 229
WebSEAL certificate 250
WebSEAL Integration 244
WebSEAL Junctions 234
webseald.conf 229
WebSphere
 JAAS 50
 JACC 275
 SSL 73, 75
WebSphere Application Server 4
WebSphere Information Center 6
WebSphere MQ 313
 Access Control List 324
 Configuration 407
 Direct Communication 322
 Messaging Components 321
WebSphere MQ integration 321
WebSphere MQ security 328
WebSphere MQ server 322
WebSphere profiles 373
WebSphere Test Environment 370
WebSphere Test Server
 new profile 381
 security 374
WebSphere User Registry 15
Windows required privileges 17
Workspace security 368
WS-Authentication 298
WS-Federation 298
WSGUILCallbackHandlerImpl 174
wsjaas_client.conf 172
WSLogin 57, 172
WS-Privacy 298
WS-SecureConversation 298
WS-Security 294
 authentication 300
 Roadmap 297

Web site 305
WS-Security specification 295
WSSecurityHelper 197
WSStdinCallbackHandlerImpl 175
WSSubject 50
WS-Trust 298

X

XML

 encryption
 Web site 305
 signature
 Web site 305
XML Encryption 295
XML Signature 295
xtokenauth 233



WebSphere Application Server V6 Security Handbook

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages

WebSphere Application Server V6 Security Handbook



Redbooks

J2EE application server and enterprise application security

Additional security components including Tivoli Access Manager

Sample code and applications

This IBM Redbook is part of the WebSphere V6 series; it focuses on security and security-related topics and provides technical details to design and implement secure solutions with WebSphere. This book provides IT Architects, IT Specialists, application designers, application developers, application assemblers, application deployers and consultants with information necessary to design, develop and deploy secure e-business applications using WebSphere Application Server V6. It not only discusses theory but also provides hands-on exercises and sample applications.

Part 1 discusses security for the application server and its components, including enterprise applications. You will find essential information on how to secure Web and EJB applications and how to develop a Java client using security. Part 2 introduces additional components from the enterprise environment and discusses security beyond the application server. External components include third-party security servers, messaging clients and servers, and database servers. Part 3 is a short introduction to development environment security. Here you can read about guidelines and best practices applicable to a secure development environment. Finally, Part 4 provides additional information related to chapters in the previous parts.

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