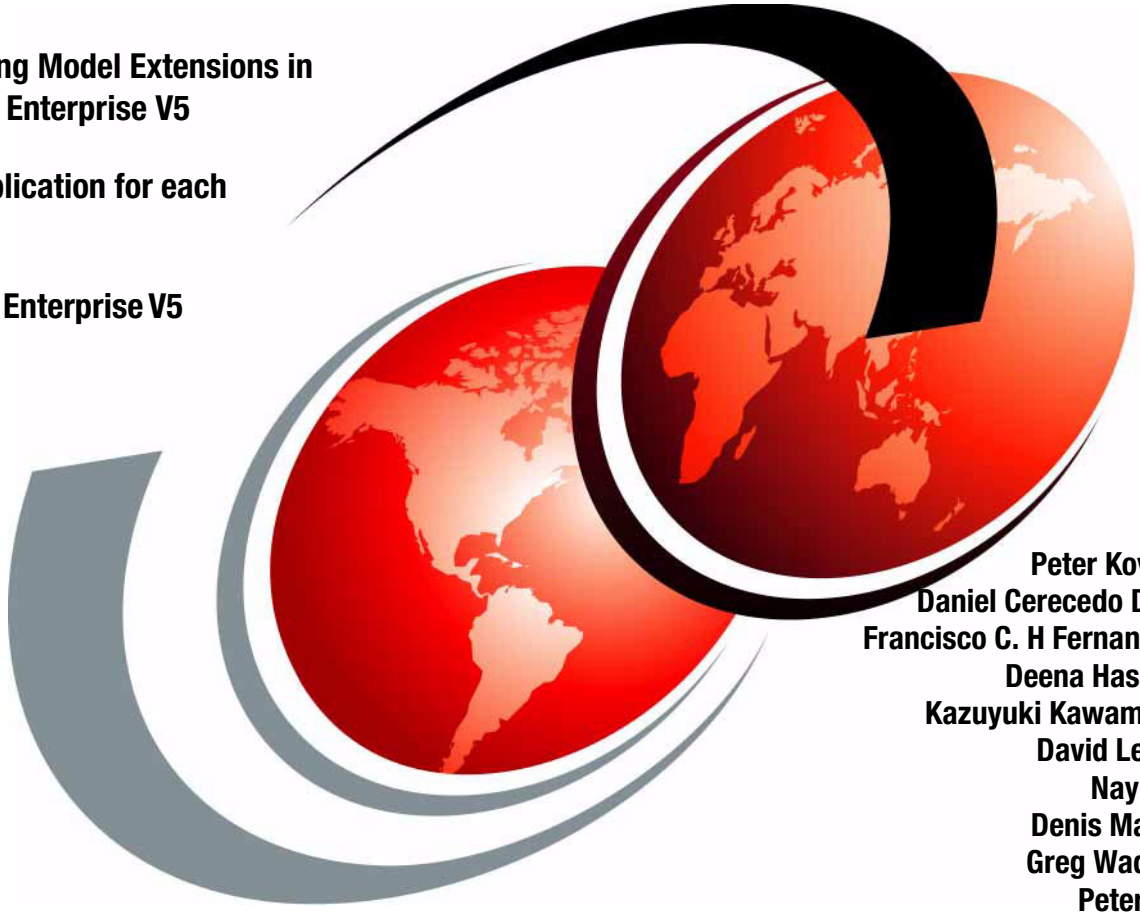
WebSphere® software

IBM

WebSphere Application Server Enterprise V5 and Programming Model Extensions

WebSphere Handbook Series

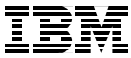
- Programming Model Extensions in WebSphere Enterprise V5
- Sample application for each PME
- WebSphere Enterprise V5 Runtime



Peter Kovari
Daniel Cerecedo Diaz
Francisco C. H Fernandes
Deena Hassan
Kazuyuki Kawamura
David Leigh
Nay Lin
Denis Masic
Greg Wadley
Peter Xu

Redbooks

ibm.com/redbooks



International Technical Support Organization

WebSphere Application Server Enterprise V5 and Programming Model Extensions WebSphere Handbook Series

August 2003

SG24-6932-00

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

First Edition (August 2003)

This edition applies to IBM WebSphere Application Server Enterprise V5.0 for use with Windows 2000, AIX 5.1; and IBM WebSphere Studio Application Developer Integration Edition V5.0 for use with Windows 2000.

Contents

Notices xiii
Trademarks xiv

Preface xv
The team that wrote this redbook xv
Become a published author xviii
Comments welcome xix

Part 1. Introduction 1

Chapter 1. Introduction 3
1.1 WebSphere Application Server Enterprise 4
1.1.1 Simplify build-to-integrate tasks 4
1.1.2 Accelerate large-scale application development 7
1.2 How this book is organized 11
1.2.1 Organization of the PME chapters 12
1.2.2 End-to-end solution implementation 13

Chapter 2. Planning 15
2.1 Planning for WebSphere Enterprise 16
2.2 Using WebSphere Enterprise 16
2.3 Service oriented architecture 16
2.3.1 Web services 18
2.3.2 J2C 19
2.3.3 Programming Model Extensions 19
2.3.4 Combining PMEs 23

Chapter 3. Sample scenario 25
3.1 Business scenario 26
3.2 Business drivers 26
3.3 Use cases 26
3.4 Business processes 29
3.4.1 Catalog update business process 29
3.4.2 Purchase Order (PO) business process 30

Part 2. Programming Model Extensions 33

Chapter 4. Process Choreographer development scenarios 35
4.1 Planning 36
4.1.1 Business processes 36

4.1.2 Why use Process Choreographer? 38
4.1.3 Comparison with WebSphere MQ workflow 39
4.1.4 Business processes for J2EE programmers 42
4.1.5 Programming model 44
4.1.6 J2EE programming model. 45
4.2 Design 47
4.2.1 Elements of a process model 48
4.2.2 Types of processes and transactions 50
4.2.3 Life cycle of a process. 56
4.2.4 Undoing service activities: compensation 56
4.2.5 Process modeling languages and standards 59
4.2.6 External programming interfaces 60
4.3 Development 64
4.3.1 Choreographer Web client 64
4.3.2 Customizing the choreographer Web client 71
4.4 Testing and debugging 148
4.4.1 Unit test environment 149
4.5 Staff support 158
4.6 Sample scenario 169

Chapter 5. Process Choreographer runtime environment 171
5.1 Process container architecture 172
5.2 Process container runtime topologies 176
5.2.1 Stand-alone application server on a single machine (SA) 178
5.2.2 Application servers network deployed with no clustering (ND-noC) 179
5.2.3 Application server clusters in Network Deployment (ND-C) 180
5.2.4 Application server cluster with WebSphere MQ Clustering (MQ-C) 183
5.2.5 Application server cluster and high-availability configuration (DB-C&MQ-C). 185
5.3 Installing the process container. 186
5.3.1 Resources required by a process container 186
5.3.2 Installing required software components. 188
5.3.3 Process container on a stand-alone topology (SA). 192
5.3.4 Installing process container in a cell topology (ND-noC) 202
5.3.5 Installing process container in topology (ND-C, ND-VS, ND-HS) . . 204
5.4 Staff plug-in provider configuration 208
5.5 Security considerations 213
5.5.1 Securing the Process Choreographer resources 213
5.5.2 Process security context 214
5.5.3 Process Choreographer J2EE security roles 215
5.6 Managing business process applications 216
5.6.1 Artifacts of a business process application 216
5.6.2 Installing business process applications 218

5.6.3 Versioning process models . . . 219
5.6.4 Starting and stopping process templates . . . 221
5.6.5 Uninstalling a business process application . . . 222
5.6.6 Editing a business process application . . . 222
5.6.7 Managing process instances . . . 223
5.7 Problem determination and troubleshooting . . . 223
5.7.1 Error messages . . . 224
5.7.2 Tracing process container . . . 224
5.7.3 Process audit trail . . . 225

Chapter 6. Extended Messaging . . . 227
6.1 Planning . . . 228
6.1.1 Java Message Service . . . 228
6.1.2 Extended Messaging . . . 229
6.1.3 Why use Extended Messaging? . . . 229
6.2 Design . . . 230
6.2.1 Messaging patterns . . . 230
6.2.2 Programming considerations . . . 233
6.3 Development . . . 234
6.3.1 Sample scenario . . . 234
6.3.2 Creating the sample . . . 235
6.4 Unit test environment . . . 249
6.4.1 Configure Extended Messaging . . . 250
6.4.2 Test the sample . . . 254
6.5 Configuration . . . 258
6.5.1 Comparison of WebSphere MQ and Embedded Messaging . . . 258
6.5.2 Configuration with JMS Embedded Messaging . . . 259
6.5.3 Configuration with WebSphere MQ as the JMS provider . . . 268
6.6 Deployment . . . 273
6.7 Transactions and workload management . . . 276
6.8 Handling late responses . . . 278
6.8.1 Late response description . . . 278
6.8.2 Configuration of late response . . . 280
6.9 Problem determination and troubleshooting . . . 284
6.10 Security considerations . . . 285

Chapter 7. Asynchronous Beans . . . 287
7.1 Planning . . . 288
7.1.1 What are Asynchronous Beans? . . . 289
7.1.2 Asynchronous Beans programming interfaces . . . 289
7.1.3 Asynchronous Beans: simple Java objects or EJBs? . . . 299
7.1.4 Asynchronous Beans: programming model . . . 300
7.1.5 When to use Asynchronous Beans . . . 302

7.2 Design 303
7.2.1 Base application overview 304
7.2.2 Asynchronous patterns 305
7.3 Development - base application 306
7.3.1 Set up the base application 306
7.3.2 Understand the base application 307
7.3.3 Configure the base application 310
7.3.4 Run the base application 311
7.4 Development: “Asynchronize” the base application 312
7.4.1 Asynchronously initialize the cache using Work 313
7.4.2 Asynchronously keep cache updated using EventListener 319
7.4.3 Asynchronously update database using AlarmListener 321
7.4.4 Set up the extended application 325
7.5 Unit test environment 325
7.6 Assembly 328
7.7 Configure 329
7.8 Deployment 331
7.9 QoS (Quality of Service) considerations 333
7.9.1 Multiple WorkManagers 334
7.9.2 Dynamically tuning WorkManagers at runtime 334
7.10 Security considerations 335
7.11 An additional sample 335
7.11.1 Implementation details 337
7.11.2 Configuration and requirements 341

Chapter 8. Application Profiling and Access Intent 343
8.1 Overview 344
8.2 Planning 349
8.3 Performance report 354
8.4 Assembly 362
8.4.1 Creating an Access Intent Policy 363
8.4.2 Creating an Access Intent 366
8.4.3 Creating an Application Profile 371
8.4.4 Creating a Dynamic Query Access Intent 378
8.4.5 Application Profiling API 379
8.5 Problem determination and troubleshooting 381

Chapter 9. Transactional Services 385
9.1 Transactions overview 386
9.2 Last Participant Support 389
9.2.1 Configuration 390
9.2.2 Troubleshooting 392
9.3 ActivitySession 396

9.3.1 Extended Local Transaction . . . 397
9.3.2 Extended EJB life cycle . . . 406
9.3.3 Usage scenarios . . . 410
9.4 Runtime . . . 410
9.4.1 Enable the ActivitySession service . . . 411
9.4.2 Troubleshooting . . . 411
9.5 JTA extensions . . . 412
9.6 Samples . . . 413

Chapter 10. Business Rule Beans . . . 419
10.1 Planning . . . 420
10.2 Design . . . 423
10.2.1 Business Rule Beans framework . . . 423
10.2.2 Architecture . . . 424
10.3 Development . . . 426
10.3.1 Development environment setup . . . 426
10.3.2 Creating the rule implementor . . . 435
10.3.3 Creating and configuring the rule . . . 436
10.3.4 Creating the rule client . . . 440
10.3.5 Integrating the sample application with the business rule . . . 441
10.4 Unit test environment . . . 444
10.4.1 Rule unit testing . . . 444
10.4.2 Testing the sample application with the rule beans enabled . . . 446
10.5 Assembly . . . 447
10.6 Deployment . . . 448
10.6.1 Running the Rule Management Application . . . 449
10.7 Problem determination and troubleshooting . . . 450
10.8 Performance considerations . . . 451
10.8.1 Client-side caching . . . 451
10.8.2 Rule firing location . . . 453
10.8.3 Creating database indexes . . . 454
10.9 Security considerations . . . 455

Chapter 11. Dynamic Query . . . 457
11.1 Planning . . . 458
11.1.1 Dynamic Query . . . 458
11.2 Design . . . 461
11.3 Development . . . 463
11.3.1 Dynamic Query Bean . . . 464
11.3.2 Development environment setup . . . 467
11.3.3 Development of Dynamic Query sample . . . 468
11.3.4 Integration of Dynamic Query with sample application . . . 475
11.4 Unit test environment . . . 476

11.4.1 Configure application server 476
11.4.2 Running the sample application 477
11.5 Assembly 478
11.5.1 Projects export 479
11.5.2 Configuring EJB Access Intent for Dynamic Query 480
11.5.3 Incorporating bpeWebclient.war 481
11.6 Configuration 482
11.6.1 Installing query.ear 482
11.6.2 Application class loader policy configuration 483
11.7 Deployment 483
11.8 Performance considerations 484
11.8.1 Transactions and Dynamic Query 484
11.9 Security considerations 486

Chapter 12. Startup Bean 489
12.1 Introduction 490
12.1.1 Why use Startup Beans? 490
12.2 Design 490
12.3 Development 491
12.4 Unit test environment 493
12.5 Assembly 493
12.6 Development 496
12.6.1 Sample scenario 496
12.7 Configuration 502
12.8 Deployment 502
12.9 Runtime environment 502
12.9.1 Priorities when using multiple Startup Beans 504
12.9.2 Scalability 506
12.10 Problem determination and troubleshooting 506

Chapter 13. Scheduler service 509
13.1 Introduction 510
13.2 Design 510
13.3 Development 511
13.3.1 Scheduler API 511
13.3.2 Steps for using the Scheduler service 511
13.4 Unit test environment 516
13.5 Assembly 516
13.6 Building and tools 517
13.7 Sample scenario 517
13.8 Configuration 527
13.9 Deployment 531
13.10 Scheduler service runtime 531

13.11 Problem determination and troubleshooting 532
13.12 Performance monitoring 534
13.13 Security considerations 536

Chapter 14. Object pools 537
14.1 Planning 538
14.2 Design 539
14.3 Development 539
14.3.1 Object Pools API 540
14.3.2 Steps for using object pools 542
14.4 Unit test environment 543
14.5 Assembly 543
14.6 Sample application 544
14.7 Configuration 552
14.8 Runtime environment 554
14.9 Problem determination and troubleshooting 555
14.10 Performance monitoring 555

Chapter 15. Shared Work Area service 559
15.1 Planning 560
15.2 Design 561
15.3 Development 562
15.3.1 Work Area API 563
15.3.2 Steps for using the Shared Work Area service 564
15.4 Unit test environment 568
15.5 Sample application 568
15.6 Configuration 580
15.6.1 Shared Work Area service configuration 581
15.6.2 Shared work area client properties 581
15.7 Problem determination and troubleshooting 582

Chapter 16. Internationalization (i18n) service 583
16.1 Planning 584
16.1.1 The traditional solutions and the limitations 585
16.1.2 The Internationalization service solution 585
16.2 Design 586
16.2.1 Internationalization context 586
16.2.2 Internationalization type 588
16.3 Development 591
16.3.1 The internationalization context API 591
16.3.2 Using the Internationalization service 594
16.3.3 Enhanced Internationalization Service 596
16.4 Unit test environment 598
16.5 Assembly 599

16.5.1 Specify the internationalization type 599
16.5.2 Specify the container internationalization attribute 601
16.6 Sample scenario for the EJB client 605
16.6.1 Description 605
16.6.2 Prerequisites 606
16.6.3 Develop 606
16.7 Sample scenario for the Web client 618
16.7.1 Implementation details 620
16.7.2 Configuration and requirements 621
16.8 Configuration 622
16.9 Deployment 623
16.10 Problem determination and troubleshooting 623
16.11 Install Enhanced Internationalization Service Technology Preview . . 624

Chapter 17. WebSphere Enterprise runtime 627
17.1 Introduction 628
17.2 Architecture 628
17.2.1 WebSphere Application Server V5 base 629
17.2.2 WebSphere Application Server Enterprise 634
17.3 Administration 643
17.4 Workload management 646
17.4.1 Scalability and high availability basics 647
17.5 Where to find more information 649

Part 3. Appendixes 651

Appendix A. Installation and configuration 653
Planning for installation 654
Installations 654
Install Enterprise and base at the same time 655
Install Enterprise to the existing base 658
Install Enterprise to the existing Network Deployment 661
Troubleshooting the installation 663
Configuration 664

Appendix B. Sample scenario 665
Sample application 666
User registry 666
Database 666
Development environment 667
Importing the sample application 667
Importing the extended sample application 669
Configuring the test environment 670
Additional configurations for the extended sample application 672

Running J2EE clients in WebSphere Studio 674
Runtime environment 675
Configuring the runtime environment 675
Deploying the base sample application 677
Uninstall the base sample 679
Deploying the extended sample application 679
Deploy the Universal Test Client (optional) 681

Appendix C. Additional material 683
Locating the Web material 683
Using the Web material 683
System requirements for downloading the Web material 684
How to use the Web material 684

Abbreviations and acronyms 685

Related publications 687
IBM Redbooks 687
Other resources 687
Referenced Web sites 688
How to get IBM Redbooks 689
IBM Redbooks collections 689

Index 691

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:

Redbooks (logo)™ 
@server™
e-business on demand™
ibm.com®
iSeries™
xSeries®
zSeries®
AIX®
Cloudscape™
CrossWorlds®
CICS®
Domino™
DB2 Universal Database™
DB2®
Holosofx®
Informix®
IBM®
IMS™
Lotus®
MQSeries®
Redbooks™
TCS®
TME®
WebSphere®

The following terms are trademarks of other companies:

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

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

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

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

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

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

Preface

This IBM® Redbook provides system administrators, developers and architects with the knowledge needed to implement WebSphere® Application Server V5.0 Enterprise runtime environment, to design, develop, assemble and deploy enterprise applications, and to perform ongoing management of the WebSphere environment.

Part 1, “Introduction” explains how the book is organized to cover the WebSphere Enterprise product. It helps you to understand the product line for planning. This part gives a broad description of the sample scenario used for the book on the business requirements level.

Part 2, “Programming Model Extensions” is the major part of the book. It covers all the Programming Model Extensions for WebSphere Application Server V5.0 Enterprise. Each extension is discussed in its own chapter starting with planning and design, through development and deployment to the configuration and administration. The book follows the J2EE roles and actions to be taken in an end-to-end solution design.

The Appendixes give detailed steps for installing and configuring WebSphere Application Server V5.0 Enterprise.There are also step-by-step instructions for configuring and deploying the sample application that is shipped with the book.

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 team who wrote the book (left to right), top line: Denis Masic, Nay Lin, Daniel Cerecedo Diaz, Greg Wadley, Francisco C Fernandes, Kazuyuki Kawamura, bottom line: Deena Hassan, David Leigh, Peter Kovari, Peter Xu

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.

Daniel Cerecedo Diaz is an IT Architect working in IBM Global Services in Madrid. He has four years of experience in J2EE development and Systems Integration. He holds a Master's degree in Computer Science and is a Sun Certified Programmer for the Java™ 2 Platform. His areas of expertise include J2EE architecture, object-oriented technologies and application design.

Francisco C. H Fernandes is a Senior IT Specialist in IBM Brasil. He has 28 years of experience in the IT industry. For the last 10 years he has worked with object-oriented application development in OO languages such as C++, SmallTalk, and Java. Currently he is working in the Technical Pre-sales Software Group with WebSphere product.

Deena Hassan is a Software Engineer at the Cairo Technology Development Center (CTDC), IBM Egypt. She is a member of the WBI Modeler development team. She has experience in the design and development of enterprise applications. She holds a Bachelor's degree in Computer Science from the American University in Cairo, and is about to receive her Master's degree in the field of Artificial Neural Networks. Her areas of expertise include object-oriented design, Enterprise Java programming, e-commerce solutions, pervasive

computing, and WebSphere products. She also co-authored *The XML Files: Development of XML/XSL Applications using WebSphere Studio Version 5*, SG24-6586.

Kazuyuki Kawamura is a Software Engineer for Hitachi Software in Tokyo, Japan. He has four years of experience in server-side Java and J2EE technologies. His areas of expertise include Java, Web services and WebSphere. Currently, he is working at the IBM WebSphere Enterprise Bringup Lab in Rochester as a trainee. He holds a master's degree in Administration Engineering from Keio University, Japan.

David Leigh is an Advisory Software Engineer in IBM Software Group's WebSphere Platform System House organization, located in Research Triangle Park, North Carolina. His areas of expertise include the Process Choreographer, application and server security, high availability, monitoring, IBM AIX®, and Linux.

Nay Lin is an Advisory Software Engineer on the IBM WebSphere Enablement Team in the United States. He has three years of hands-on expertise in the WebSphere software platform, having engaged in many pre-sales consulting engagements, proofs of concept, and resolution of critical situations on WebSphere Application Server on z/OS and distributed systems, WebSphere Enterprise Programming Model Extensions, WebSphere application development with WebSphere Studio Application Developer Integration Edition. Before joining IBM in 2000, he had eight years of experience as a software engineer involved in object-oriented analysis and design, embedded and real-time software development, and management of software processes.

Denis Masic is an Advisory IT Specialist working in IBM Global Services Slovenija as a member of the EMEA WebSphere Back Office team. Before joining IBM two years ago, he worked as a C software developer. He is an IBM certified system expert for WebSphere. He holds a BsC in Computer Science from the University of Ljubljana. His computer career started 16 years ago. His areas of expertise are computer reliability and optimization methods.

Greg Wadley is a Sr. Certified IT Specialist in Kennedale, TX. He provides technical sales support for WebSphere Application Server, WebSphere Portal Server, and WebSphere Studio Application Developer. Greg has worked with IBM since 1986 in a variety of roles, including development, presales technical support, and architecture.

Peter Xu is a Consulting I/T Specialist with IBM Software Services for the WebSphere group, helping customers deploy IBM products into their organizations. He provides consulting services, education, and mentoring on J2EE technologies, and specifically WebSphere and WebSphere Studio products to Fortune 500 clients. Peter is a certified WebSphere Enterprise

Developer and System Expert. He holds a Master's degree in Computer Science from the State University of New York.

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

Cecilia Bardy
Gail Christensen
Linda Robinson
Jeanne Tucker
Margaret Ticknor
International Technical Support Organization, Raleigh Center

William Alward
Logan Colby
Ryan Cox
Eric Erpenbach
Eric Herness
Chris D Johnson
Richard Johnson
Martin Keen
Alex Koutsoumbos
Yiu Cho Lau
Billy Newport
Steve Parsons
Ruth Schilling
Lin Sharpe
Jeff Stratford
Phil Wakelin
Sherri Wayne
Gunnar Wilmsmann

Special thanks to the WebSphere Enterprise Bringup Lab in Rochester for the invaluable help during the project.

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HZ8 Building 662
  P.O. Box 12195
  Research Triangle Park, NC 27709-2195

Part 1

Introduction

1

Introduction

The first chapter of this book consists of two parts.

The first section is a high-level summary of the WebSphere Enterprise V5 product line, the WebSphere Application Server V5, and WebSphere Studio Application Developer Integration Edition V5 integration, including the end-to-end solution design, development, and runtime.

The second section tells how the book is organized, in order to give a better understanding of what is discussed in subsequent chapters.

1.1 WebSphere Application Server Enterprise

Together, WebSphere Application Server Enterprise, Version 5 and WebSphere Studio Application Developer Integration Edition for Linux and Windows®, Version 5 deliver a next-generation application server and development environment designed to deliver on demand e-business applications by offering the following:

- **Simplify build-to-integrate tasks**: Help customers to reduce IT complexity, reuse existing resources, and automate business processes through a powerful but simplified build-to-integrate framework.
- **Accelerate large-scale application development**; Leverage the latest innovations that build on today's Java™ 2 Platform, Enterprise Edition (J2EE) standards to deploy a high performance e-business infrastructure designed to cut costs, build customer loyalties, promote business agility, and gain a competitive advantage.
- **Enable real-time application flexibility**: Take advantage of dynamic application support that allows customers to build applications that can adapt on demand to the ever-changing world of e-business.

1.1.1 Simplify build-to-integrate tasks

Companies today face a growing problem as they begin to explore new e-business initiatives. The past 40 years of IT evolution have left them with an enterprise-computing infrastructure that is heterogeneous, widely distributed, and increasingly complex.

Business logic and application data are scattered throughout the organization across multiple software assets. Much of the business logic resides in databases, packaged applications (such as Enterprise Resource Planning (ERP) systems), or in back-end systems (such as IBM CICS®) offering varying levels of transaction support. Other business logic can be found in existing Java and J2EE applications and Web services. Furthermore, companies face constant pressure to create new applications in order to cut costs, build customer loyalties, and gain a competitive advantage.

Instead of reinventing the wheel with every new application they build, companies need a way to reuse their existing software assets and to leverage the power of Web services in the development of new J2EE-based applications.

WebSphere Application Server Enterprise and WebSphere Studio Application Developer Integration Edition help companies to reduce IT complexity, reuse existing resources, and automate business processes through a powerful but simplified build-to-integrate framework.

- The **service oriented architecture** reduces the complexity of large-scale application development and promotes reuse by offering a standard way of representing and interacting with virtually all software assets.
- **Integrated workflow** increases development productivity and promotes reuse by enabling developers to visually choreograph interactions between software assets.
- **Advanced transactional connectivity** capabilities help developers avoid custom coding by providing support for the many challenges related to integrating existing software assets with a J2EE environment.

Service oriented architecture (SOA)

Building new applications that integrate business logic and application data within the organization and with suppliers, partners, and customers is critical to the success of today's organizations. However, this integration remains complex, expensive, and risky.

A service oriented architecture leverages open standards to represent virtually all software assets as services, including legacy applications, packaged applications, J2EE components, or Web services. This approach provides developers with a standard way of representing and interacting with software assets without having to spend time working with unique interfaces and low-level APIs. Furthermore, individual software assets become building blocks that can be reused in developing other applications.

Using this new service-oriented approach to integration, WebSphere Application Server Enterprise and WebSphere Studio Application Developer Integration Edition help reduce the complexity, cost, and risk of integration by providing a single, simple architectural framework based on Web services in which to build, deploy, and manage application functionality.

Integrated J2EE-based workflow

Once a developer has created services out of an organization's software assets, the next logical step is to use those assets as part of a business process. Integrated J2EE workflow capabilities offer developers intuitive, flow-based development tools to take existing software assets and quickly define how those assets are used within a J2EE-based application.

For example, the visual workflow tools can be used to combine inventory information from a packaged ERP application and J2EE components from a previously built customer-facing application with new business logic to create a new Web-based order entry application. The reach of the application can then be extended by exposing it as a Web service for use by business partners or to allow manual intervention for exception handling. The result is faster development of

new applications, improved consistency, and lower costs through the reuse of existing IT investments.

- **Visual process editor (choreographer)**: Provides intuitive drag-and-drop tools to easily compose and choreograph application interactions and dynamic workflows among J2EE components, Web services, existing applications, and human activities. Developers can quickly and easily build, debug, and deploy complex applications using powerful workflow tools and advanced messaging capabilities to streamline and automate business processes. New services can be added, or existing services modified without affecting the other components in the business process.
- **Human interaction**: Offers support for including activities that require a person to perform a task as a step in an automated business process. Specialized staff support allows the dynamic assignment of responsibilities based on existing organizational definitions. Worklists can be created to let the designated recipient know that their action is required.
- **Event triggering**: Offers support for including asynchronous events such as Web services or human interactions to be included as part of business processes. Events can be used to trigger the start of a business process, or a business process can be configured to stop and wait for an external event to occur before resuming the process.
- **Compensation pairs**: Provides transaction “rollback” support for long-running, loosely coupled business processes that cannot be undone automatically by the application server. For example, the compensating transaction for an order that has already started manufacturing might be to put the complete item into inventory (rather than disassembling the item). Compensation pairs allow you to define, for each step in your business process, the associated “undo” service.
- **Flexible workflow design**: Provides developers with the ability to design workflows using a top-down, bottom-up, or meet-in-the-middle approach. Using top-down, developers can create skeleton processes that choreograph the sequence of events in a workflow without worrying about the underlying implementation. Building from the bottom-up, developers first create the individual components and then use them as building blocks to define a workflow. Meet-in-the-middle offers the flexibility of using both approaches at the same time.

Advanced transactional connectivity

Since its inception, the J2EE platform has made huge strides in providing enterprise-level support for integration, including support for messaging, security, and database access. The Java Connector Architecture (JCA) 1.0 standard begins to offer support for integrating with packaged and legacy applications. However, due to lack of adherence to data standards and limited transactional

support, integrating with most back-end resources and legacy data is still complex, expensive, and risky.

WebSphere Application Server Enterprise and WebSphere Studio Application Developer Integration Edition offer advanced transactional capabilities to help developers avoid custom coding by providing support for the many challenges related to integrating existing software assets with a J2EE environment.

- **Dynamic application adapter support**: Offers the ability to build and deploy rich, open standards-based application adapters for popular enterprise information systems such as SAP and IBM CICS.
- **Last Participant Support**: Provides automated coordination for transactions that include two-phase commit resources and a single one-phase commit resource. This support eliminates hand coding in this scenario and allows you to include one-phase commit resources, common for many legacy and package applications, in real transactions.
- **Activity session services**: Provide the ability to extend the scope of, and group, multiple local transactions. These local transactions can then be committed based on deployment criteria or through explicit program logic. This ability reduces the complexity of dealing with commitment rules and limitations associated with one-phase commit resources.
- **CORBA C++ Software Development Kit (SDK)**: Used for integrating various C++ assets. This lets C++ clients invoke J2EE components using CORBA technology and also lets WebSphere applications incorporate C++ assets behind CORBA wrappers.

1.1.2 Accelerate large-scale application development

Companies today strive to respond with flexibility and speed to customer demands, market opportunities, and external threats. However, for most companies, the time, cost, and complexity of large-scale application development make this goal extremely difficult to achieve.

WebSphere Application Server Enterprise and WebSphere Studio Application Developer Integration Edition leverage the latest innovations that build on today's J2EE standards to help customers deploy a high performance e-business infrastructure designed to cut costs, build customer loyalties, promote business agility, and gain a competitive advantage.

- **Optimize application performance**: Use powerful Application Profiling techniques, sophisticated deployment management, and advanced support for Web services to optimize performance and minimize downtime for applications that require highly available, high volume, multi-server environments.

- **Enable “next generation” development** by leveraging the latest innovations that build on today's J2EE standards to achieve greater control over application development, execution, and performance than ever possible before.
- **Increase development productivity** by taking advantage of supported, pre-built, J2EE based solutions to many of today's biggest programming challenges.

Optimize application performance

Increasingly, organizations are using Web applications both internally and externally to incorporate customers, partners, and suppliers into their business processes. For these mission-critical processes, application performance can make the difference between competitive advantage and failure to compete. WebSphere Application Server Enterprise and WebSphere Studio Application Developer Integration Edition help you optimize performance and minimize downtime for applications that require highly available, high volume, multi-server environments through powerful Application Profiling techniques, sophisticated deployment management, and advanced support for Web services.

- **Application Profiling**: Delivers powerful new capabilities that allow you to carefully optimize the performance of applications without any impact on source code. This capability offers a mechanism for specifying the Access Intent of persistent Entity EJBs allowing them to interact with the runtime infrastructure, such as a database, differently depending on the Access Intent (for example read vs. update) of the application that calls it. The result is unprecedented control in defining strategies that dynamically control concurrency, prefetch, and read-ahead.
- **Deployment manager**: Addresses the needs of highly available, high volume, multi-server environments through enhanced workload management and dynamic caching, centralized security capabilities and performance management tools that distribute workload across multiple servers through sophisticated load balancing and clustering capabilities. The deployment manager also enables isolation of application servers to avoid single points of failure and provides first failure data capture to report and analyze problems as they occur.
- **Advanced Web services support**: Offers advanced support for Web services, including a UDDI Registry that acts as a repository that allows storage of business units that describe basic Web services, and a Web Services Gateway that enables Web services invocation by users from outside the firewall with the benefit of robust security protection. Advanced Web services support also extends the Web Services Gateway by providing a programming model that allows you to use the gateway in large-scale Web services implementations to serve as a bi-directional control point for critical tasks such as validation, logging, transformation, auditing, and metering.

Enables “next generation” development

Ironically, J2EE's main advantage, its specification, can also be its biggest disadvantage for developers building applications that require them to have more control over their applications than the J2EE specifications provides. For those developers, WebSphere Application Server Enterprise and WebSphere Studio Application Developer Integration Edition enable “next generation” development by leveraging the latest innovations that build on today's J2EE standards to provide greater control over application development, execution, and performance than ever possible before.

- **Asynchronous beans**: Offer exceptional performance enhancements for resource-intensive tasks by enabling a single request to be executed as multiple tasks or threads processed in parallel within the J2EE environment. Asynchronous scheduling facilities can also be used to process parallel processing requests in “batch mode” at a designated time.
- **Startup Beans**: Allow business logic to be automatically executed when an application starts or stops. For example, they might be used to pre-fill application specific caches, initialize application level connection pools, or perform other application-specific initialization and termination procedures.
- **Scheduler service**: Helps minimize IT costs and increase application speed and responsiveness by maximizing utilization of existing computing resources. The scheduler service provides the ability to process workloads using parallel processing, set specific transactions as high priority, and schedule less time-sensitive tasks to process during low traffic off-hours.
- **Object pools**: Increase application performance by allowing instances of objects to be reused, reducing the overhead associated with the instantiating and garbage collecting the objects. Creating an object pool allows an application to obtain an instance of a Java object and return the instance to the pool when it has finished using it.

Increase development productivity

The time required to roll out new applications is a key concern across all industries. One way to vastly improve developer productivity is to reduce the need for handcrafted solutions that can be time-consuming, costly, and difficult to maintain. WebSphere Application Server Enterprise and WebSphere Studio Application Developer Integration Edition were designed to improve developer productivity by leveraging the latest innovations that build on today's J2EE standards to provide supported, pre-built solutions to many of these biggest challenges.

- **Extended Messaging**: Allows you to quickly create applications that integrate with other systems through a messaging infrastructure. This extended messaging capability offers automated support for outbound (as well as inbound) messaging, allowing you to focus on business logic instead of

complex messaging APIs. Handcrafted Java Message Service (JMS) code is no longer required. WebSphere Application Server Enterprise includes WebSphere MQ® and WebSphere MQ Event Broker to further extend your messaging infrastructure to take advantage of both products' qualities of service and to enable seamless integration with existing MQ infrastructures.

- **Internationalization service**: Allows you to automatically recognize the calling client's time zone and location information so your application can act appropriately. This technology allows you to deliver to each user, around the world, the right date and time information, the appropriate currencies and languages, and the correct date and decimal formats.
- **Work areas**: Provides a “global variable”-like ability to efficiently share information across a distributed application. For example, you might want to add profile information as each customer enters your application. By placing this information in a work area, it will be available throughout your distributed application and eliminate the need to hand-code a solution or to read and write information to a database.
- **Cheat sheets**: Make new or complex tasks easy by providing a checklist for common development patterns. The cheat sheet invokes each step in the checklist and provides detailed online help for each step, as you need it.
- **Best-in-class IDE**: Included with WebSphere Studio Application Developer Integration Edition is a fully integrated application development environment for creating and maintaining J2EE applications and Web services. Built on Eclipse V2 innovations and written to J2EE specifications, WebSphere Application Developer Integration Edition helps optimize and simplify J2EE application development with best practices, visual tools, templates, code generation, and the most comprehensive development environment in its class.

Enable real-time application flexibility

Maintaining competitive advantage in today's changing business environment requires companies to respond quickly to customer demands, market opportunities, and external threats. Very often this means making frequent updates to e-business applications to reflect changes in market conditions or to provide access to strategic information. Unfortunately, these updates usually take a great deal of time: time to bring down the application, time to make programming changes, time to test the new application, and time to redeploy.

To enable real-time application flexibility, WebSphere Application Server Enterprise and WebSphere Studio Application Developer Integration Edition offer dynamic application support to enable you to build applications that can easily adapt to the ever-changing world of e-business on demand™.

- **Business rule beans**: Offer a powerful real-time framework for defining, executing, and managing business rules that encapsulate business policies that vary based on changes in the business environment. For example, a simple business rule might be, “If a customer's shopping cart is greater than $X, then offer a Y% discount.” Once the business rule is defined, a developer or a business analyst can update the business rule at runtime using a straightforward user interface without the need to bringing the application or server down.
- **Dynamic Query service**: Delivers unprecedented application flexibility by allowing you to dynamically build and submit queries that select, sort, join, and perform calculations on application data at runtime. Dynamic Query service provides the ability to pass in and process Enterprise JavaBeans query language (EJB QL) queries at runtime eliminating the need, as with today's EJB 2.0 standards, to hard-code required queries into the Deployment Descriptors during development.

1.2 How this book is organized

This book covers the WebSphere Application Server Enterprise V5 runtime environment and Programming Model Extensions (PME). The book discusses the runtime environment specific to WebSphere Enterprise. The base application server is discussed in other books.

This book builds upon the following published Redbooks on WebSphere Application Server runtime:

- *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195. This book has all the information related to system management and configuration for the base application server.
- *WebSphere V5.0 Applications: Ensuring High Performance and Scalability*, SG24-6198. This book covers the performance and availability topics for the base application server using the WebSphere Application Server V5 Network Deployment (ND) package.
- *IBM WebSphere V5.0 Security WebSphere Handbook Series*, SG24-6573. This book discusses the security-related topics for the base application server.

A major part of the Enterprise application server are the Programming Model Extensions (PME). These following steps in working with these extensions are covered in detail in this book:

- Planning
- Design

- Development, unit test
- Assembly
- Configuration
- Deployment
- Troubleshooting

The primary development environment for WebSphere Application Server Enterprise V5 is the WebSphere Studio Application Developer Integration Edition V5. WebSphere Studio IE provides the integrated development environment and a test environment for the Enterprise application server. WebSphere Studio IE also provides tool support for some of the PMEs. Although there is no tool support for every one of the PMEs in WebSphere Studio IE, the WebSphere V4 Enterprise test environment is available, which makes the development for all the PMEs available.

Since this book is focusing on the runtime and the Programming Model Extensions for WebSphere Enterprise, there is another redbook that covers the development environment: WebSphere Studio IE, see the details below.

This book builds upon the following published redbooks on application development for WebSphere Application Server:

- *Exploring WebSphere Studio Application Developer Integration Edition 5.0*, SG24-6200. The primary book for development in the WebSphere Studio IE environment.
- *WebSphere Version 5 Web Services Handbook*, SG24-6891. Web Services is a major part of WebSphere Application Server Enterprise, this books provides an end-to-end coverage of Web Services.
- *WebSphere Application Server Enterprise Edition 4.0: A Programmer's Guide*, SG24-6504. The previous version of WebSphere Application Server Enterprise is documented in this book. Some of the PMEs have not changed much since the last version. Therefore this book can still be a good source of information.

1.2.1 Organization of the PME chapters

Each of the Programming Model Extensions (PMEs) is documented in a separate chapter. The chapters are organized in two major parts:

- Design and development
- Runtime and maintenance

The first part covers the following topics:

1. In the planning section you will find the answer to the question: What is this PME good for? It helps to decide how to use a certain PME in a solution, and how to find the solution using the PME for a certain business need.
2. The design section shows how the PME fits into the design. It also discusses the design considerations related to the PME.
3. The development section discusses the development process using the PME for the sample application. The development environment used for this part is WebSphere Studio IE.

   The unit test and the configuration of the unit test environment is an integral part of the development section.
4. The assembly part shows how to use the Application Assembly Tool to make minor changes to the application regarding the PMEs.
5. The configuration section walks through the runtime configuration steps for the PME using the sample application.
6. The deployment section uses the sample application to show the deployment procedure for the runtime.
7. The section on problem determination and troubleshooting lists the common problems that can be avoided.
8. Sections on security and transaction considerations are included to certain PMEs only where it makes sense.

1.2.2 End-to-end solution implementation

Figure 1-1 on page 14 is a representation of the WebSphere Enterprise framework, including products and implementation steps. The figure has three main object types:

- The blue rectangles represent the products.
- The yellow diamonds represent the actions you can take to implement the solution.
- The magenta ovals represent the application as it goes through the implementation.

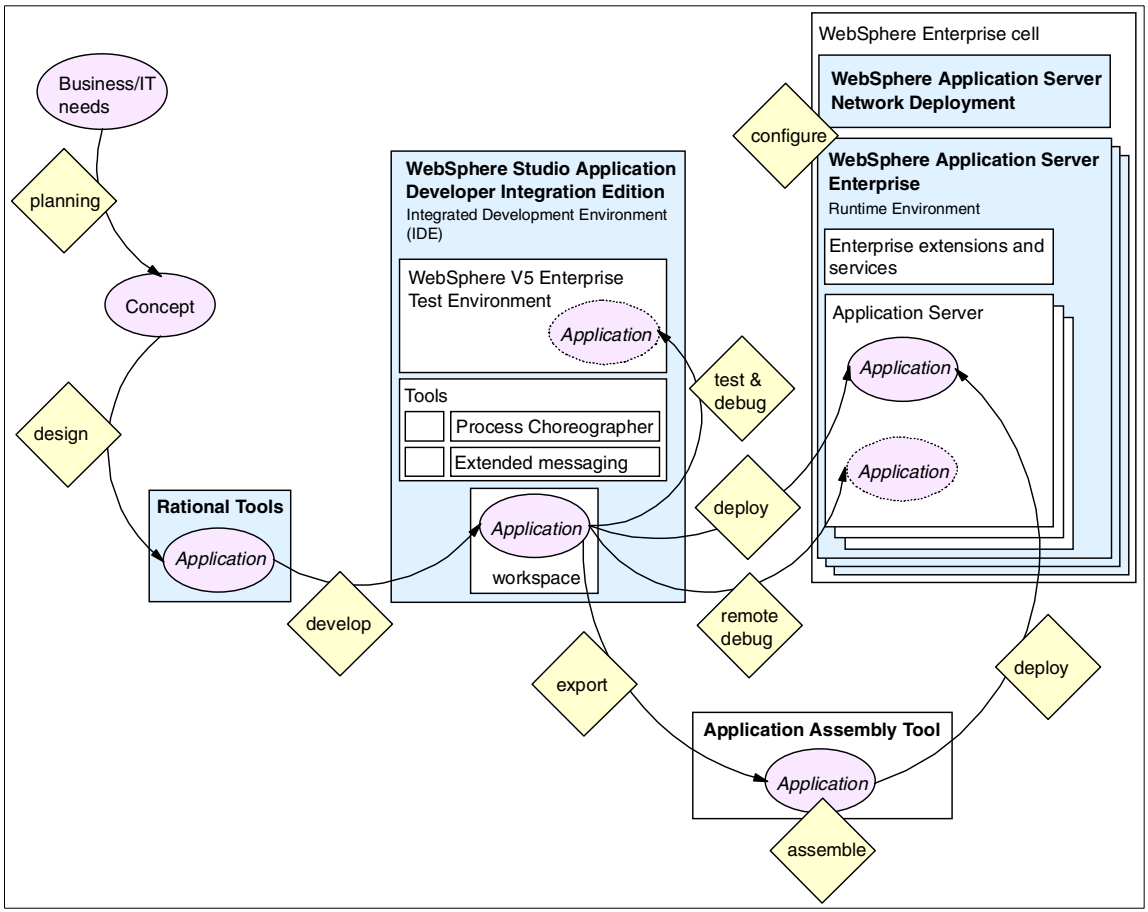


Figure 1-1 WebSphere Enterprise framework

2

Planning

This chapter provides information to help you plan your solution using WebSphere Application Server Enterprise V5.

There are mainly two approaches to planning:

- No decision was made regarding the application server and information is needed to find the right solution for the business and technical problems.
- A decision was made to use WebSphere Enterprise and information is needed to plan the details of the solution.

This chapter takes the first approach and helps to find the answer to the business and technical problems in the WebSphere Enterprise domain.

On the other hand, in the rest of the book you will find answers to the questions that will arise when following the second approach.

2.1 Planning for WebSphere Enterprise

The WebSphere Enterprise application server is based on the base WebSphere Application Server, which is J2EE 1.3 certified and implements the standard specifications. WebSphere Enterprise adds new functions and services to the Application Server. Some of the functions are simply Programming Model Extensions that help to create better solutions.

On the other hand, some extensions are major changes to the Application Server. These changes are outside of the J2EE specification, although some of the extensions already exist as proposals or are under evaluation to extend the J2EE specification.

J2EE is an open standard and as such is open to extensions. When one chooses to use the enterprise extensions for an implementation, then the Application Server does not strictly follow the standard. On the other hand, if the implementation must adhere to the strict standards, the application server has to be the base server without proprietary extensions.

2.2 Using WebSphere Enterprise

There are fundamental technologies in WebSphere Enterprise that enable the application server to be used in a demanding enterprise environment where the base application server and standard J2EE functions cannot fulfill the requirements.

2.3 Service oriented architecture

Service oriented architecture (SOA) can be broadly described as a logical collection of interacting services offering well-defined interfaces to potential consumers. A service is generally defined as a course-grained, self-described and discoverable software entity that exists as a single instance and interacts with applications and other services through a loosely coupled (often asynchronous), message-based communication model. Simply put, SOA allows software assets to be presented as services regardless of their programming language, operating system (OS) platform, or geographic or organizational location.

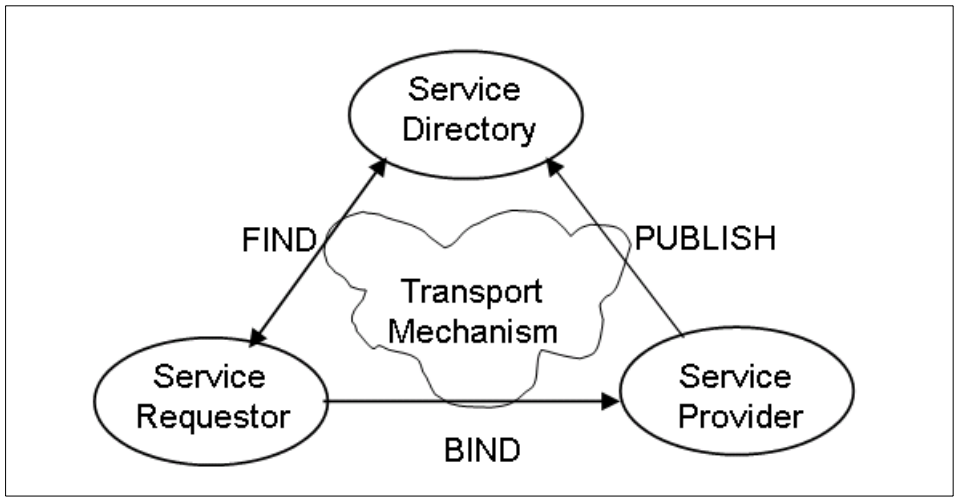


Figure 2-1 Web services architecture

Web services are a well-known example of such an architecture. Web services are self-contained, self-describing, modular business applications that can be published, located and invoked over a network, typically the Web. They are based on the industry standards to describe, to publish and discover services, and to communicate. This service oriented architecture enables applications to interoperate across organizational boundaries in a manner independent of platform and programming language. Web Services Description Language (WSDL) provides an industry-standard way of describing Web services.

A WSDL description of a Web service has:

- **Interface information**: Port type, the operations that the port type supports, and the structure (parts) of the input and output messages.
- **Binding information**: Binding interface to concrete implementations, that is, mapping of operations to methods and message parts to data types of various implementations (EJB, Java, SOAP or JMS).
- **Service information**: Map service name to the ports that implement the bindings by which the port types and operations can be reached (location of the EJB, Java class, RPC router, or JMS destinations and factories).

Note that while WSDL is well known in association with Web services implemented with SOAP (Simple Object Access Protocol) over HTTP, it can be extended to describe other types of service implementations and protocols, such as RMI/IIOP to invoke Java and EJB services, and JMS to invoke asynchronous service.

In order to provide a programming environment where we can easily incorporate services into a business process, we need a form of service integration bus or an invocation framework so that services can be invoked transparently. WebSphere Application Server provides runtime environment based on the Web Services Invocation Framework (WSIF) to invoke these services. The WSIF is a simple Java API for invoking Web services independent of transport protocols or service environments. It is an API that provides binding-independent access to any WSDL described service. It allows stubless or completely dynamic invocation of Web service, based upon examination of the metadata about the service from WSDL at runtime. It also allows updated implementations of a binding to be plugged into WSIF at runtime. It allows the calling service to choose a binding deferred until runtime. In summary, WSDL described services based on SOA together with the service integration bus. WSIF allows services to be choreographed into processes.

Note: The WSIF source code, developed by IBM, has been donated to the Apache XML project under the auspices of the Axis work. The code can be browsed at `http://cvs.apache.org/viewcvs.cgi/xylem-axis-wsif`.

Process Choreographer implements dynamic invocation frameworks for services and activities as plug-ins. Hence you can replace an IBM installed plug-ins with those from third-party providers. WebSphere Studio IE provides GUI-based wizards so that a programmer can develop business processes without dealing with WSIF directly.

A discussion of service oriented architecture is outside the scope of this book. For a detailed discussion of this topic, refer to the following IBM Redbooks:

- *WebSphere Version 5 Web Services Handbook*, SG24-6891
- *Exploring WebSphere Studio Application Developer Integration Edition 5.0*, SG24-6200

2.3.1 Web services

Web services is an implementation of the service oriented architecture (SOA) and it is the primary technology for solution implementation.

A discussion of Web services is outside the scope of this book. For a detailed discussion of this topic, refer to the following IBM Redbooks:

- *WebSphere Version 5 Web Services Handbook*, SG24-6891
- *Exploring WebSphere Studio Application Developer Integration Edition 5.0*, SG24-6200

2.3.2 J2C

Java 2 Connectors (J2C) is a fundamental technology that is responsible for connecting J2EE applications to other non-J2EE applications or resources within the enterprise.

A discussion of Java 2 Connectors is outside the scope of this book. For a detailed discussion of this topic, refer to the following IBM Redbooks:

- *Exploring WebSphere Studio Application Developer Integration Edition 5.0*, SG24-6200
- *Java Connectors for CICS: Featuring the J2EE Connector Architecture*, SG24-6401

2.3.3 Programming Model Extensions

WebSphere Enterprise brings several Programming Model Extensions (PMEs) to the application server. The extensions are delivered in different forms, including services, APIs, wizards for development, and deployment extensions.

The extensions are in three major groups:

- Business Object Model extensions operate with business objects, for example EJBs.
- Business Process Model extensions provide process, workflow functionality, and services for the application server.
- Next Generation Applications include the rest of the extensions. They can be used in enterprise applications where specific needs require these extensions.

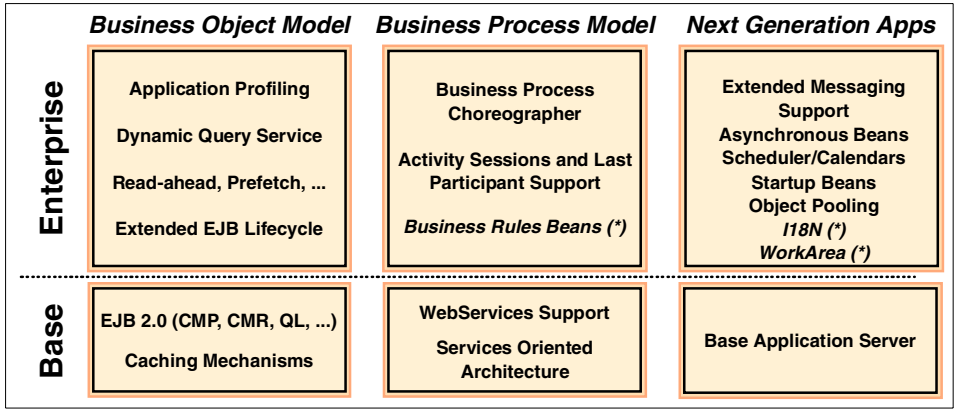


Figure 2-2 Application server functions and features in base and Enterprise

The following sections provide basic planning information for each of the Programming Model Extensions (PMEs). Each extension is discussed in detail in separate chapters in this book. The chapters each have a planning section together with the introduction where the extensions are explained.

Process Choreographer, including staff services

Process Choreographer is probably the most appealing extension in WebSphere Enterprise. It provides workflow and process functionality for the application server. It supports long-running and short-running processes, interruptible and non-interruptible processes.

The processes are developed in WebSphere Studio Application Developer Integration Edition using a visual editor to assemble the flow. The process application is then deployed in the application server's process container.

The runtime environment also provides a browser-based client application for administrative purposes. The default client can be freely customized and extended using the client API.

The Process Choreographer has an integral service called staff services, which provides advanced staff and user handling functions beyond the base user registry function.

Extended Messaging

Extended Messaging (EM) is major help in building messaging-based applications. This extension provides an API built on top of JMS, but it would not help much itself if it did not provide a development environment where building messaging applications is just as easy as building from blocks.

Developers can use the WebSphere Studio Application Developer Integration Edition Extended Messaging wizard to build messaging components for any messaging patterns. Using the wizard minimizes the application coding significantly.

For detailed information, refer to Chapter 6, "Extended Messaging" on page 227.

Asynchronous Beans

Asynchronous execution of processes in J2EE application servers was always a big need and a big problem. It was almost impossible to develop event-driven applications for application servers, not because of the technology but the lack of support in the original specification and in the programming model.

WebSphere Enterprise solves this problem by providing full support for asynchronous execution and invocation of threads and components within the

application server. The application server provides an execution and security context for the components, making them an integral part of the application.

For detailed information, refer to Chapter 7, “Asynchronous Beans” on page 287.

Application Profiling, Access Intent

Application Profiling and Access Intent provide a flexible method to fine-tune application performance for EJBs. Different EJBs and even different methods in one EJB can have their own intent to access resources. Profiling the components based on their Access Intent increases performance in the runtime.

For detailed information, refer to Chapter 8, “Application Profiling and Access Intent” on page 343.

Last Participant Support

Last Participant Support is an extension to the original J2EE transaction support for applications. It allows developers to include several two-phase commit resources and one, and only one, one-phase commit resource in one unit of work.

For detailed information, refer to Chapter 9, “Transactional Services” on page 385.

ActivitySession

ActivitySession provides further extensions to the J2EE session and transaction services. It can extend session and transaction boundaries.

For detailed information, refer to Chapter 9, “Transactional Services” on page 385.

Business Rule Beans

Business Rule Beans externalize the business rules in an application. Rules, conditions, and decisions don’t have to be hardcoded or custom tailored into applications. Business Rule Beans provide an API to look up and use the rules, and provide an Administrative Console to manage application rules.

For detailed information, refer to Chapter 10, “Business Rule Beans” on page 419.

Dynamic Query

Dynamic Query fixes a weakness in Enterprise JavaBeans by enabling the client to run custom queries on EJBs in runtime. Until now, EJB’s lookups and field mappings were implemented in development time and it required further

development to modify any of the query attributes. With Dynamic Query, the client can assemble SQL statements in runtime and execute them on any EJB. The quasars can result in not only objects and a list of objects but also aggregate functions (for example SUM, COUNT).

For detailed information, refer to Chapter 11, “Dynamic Query” on page 457.

Startup Beans

Startup Beans solve the problem of running custom code during application server startup. Start time initialization was a problem in most applications but it has now been solved with this new function.

For detailed information, refer to Chapter 12, “Startup Bean” on page 489.

Scheduler service

The problem of scheduled execution was similar to Startup Beans and asynchronous execution in the application server. The scheduler service enables the application server to execute application code as scheduled in a timely fashion.

For detailed information, refer to Chapter 13, “Scheduler service” on page 509.

Object pools

Application performance is essential in runtime. The most effective way to improve performance is using object pools in applications. Object pools is a simple programming practice but still requires programming of libraries from the developer. WebSphere Enterprise has a built-in service that handles any Java type of object pools and provides an API for the developer to use this service in the application code. This service shortens the development time significantly and standardizes object pooling in the application.

For detailed information, refer to Chapter 14, “Object pools” on page 537.

Shared Work Area

In the process of developing software applications, the need to pass data between application components is often a fundamental requirement. Shared Work Areas provide a solution to pass and propagate contextual information between application components.

For detailed information, refer to Chapter 15, “Shared Work Area service” on page 559.

Internationalization (I18N) service

An application that can present information to users according to regional cultural conventions is said to be *internationalized*. The application can be configured to interact with users from different localities in culturally appropriate ways. The Internationalization (I18N) service gives the ability to develop internationalized applications in WebSphere.

For detailed information, refer to Chapter 16, “Internationalization (i18n) service” on page 583.

CORBA C++ SDK

The CORBA C++ SDK is also part of the WebSphere Application Server Enterprise V5 product, although this book does not discuss it. For more information about the CORBA C++ SDK, refer to the redbook written for the previous version of WebSphere Enterprise: *WebSphere Application Server Enterprise Edition 4.0: A Programmer's Guide*, SG24-6504.

2.3.4 Combining PMEs

Each of the Programming Model Extensions has its own place and use in applications. Most likely, not all the extensions are required for a specific solution. Using one or two extensions can shorten development time, and can provide additional functions or services to an application. On the other hand, combining extensions and using them together in one solution is also an option to build more complex solutions for specific needs.

Some of the Programming Model Extensions can be combined effectively into one solution. Figure 2-2 on page 19 list the extensions under three different groups. The extensions in one group can be easily combined in one particular solution, since they aim to solve the same type of problem. Extensions from different groups can be combined as well. For example, a workflow process can invoke a messaging component using Extended Messaging.

The following is only a short list of examples illustrating the numerous opportunities for combining different PMEs in one solution:

- Process Choreographer can combine most of the PMEs:
  - Extended Messaging for messaging components
  - Business Rule Beans to implement rules for the process
  - Startup Beans to start a process
  - Scheduler beans to start a process
- Extended Messaging fits nicely into a Process Choreographer business process as a component. Extended Messaging can be also fired from a Startup Bean or a scheduler bean.

- Object pool, Shared Work Area and Internationalization (I18N) services can be combined with almost any of the PMEs. They are low-level APIs that can be used in the implementation of the components.
- Asynchronous beans and Dynamic Query are also universal components and can be tied to most of the PMEs.

3

Sample scenario

This chapter gives an overview of the sample scenario used in this book to show the functions and features of WebSphere Enterprise V5.

The sample scenario is discussed on a business-case level, rather than getting into much details on the technical and implementation level. The technical details of the sample application that is based on this sample scenario can be found in Appendix B, “Sample scenario” on page 665.

3.1 Business scenario

The business scenario is based on a fictional company called ACompany. This section provides a background for the sample application and begins an exploration of the business needs.

3.2 Business drivers

ACompany has a purchasing department that handles the company purchases for office supplies and other employee needs.

The company has grown in size and the purchases have increased significantly. Handling individual purchases and running the process on paper is not sufficient anymore. The increased spending also needs to be controlled to save on expenses.

The solution is to implement a process that can replace the existing paper-based purchasing process. The application is available for every employee in the company. It is controlled by the purchasing and financial department.

The purchases are initiated by the employee, based on a catalog that is maintained and controlled within the company.

3.3 Use cases

Two use cases identified for the sample scenario will be implemented in the sample application. The two use cases are:

- Purchase order
- Catalog update

Figure 3-1 on page 27 is a UML representation of the two use cases.

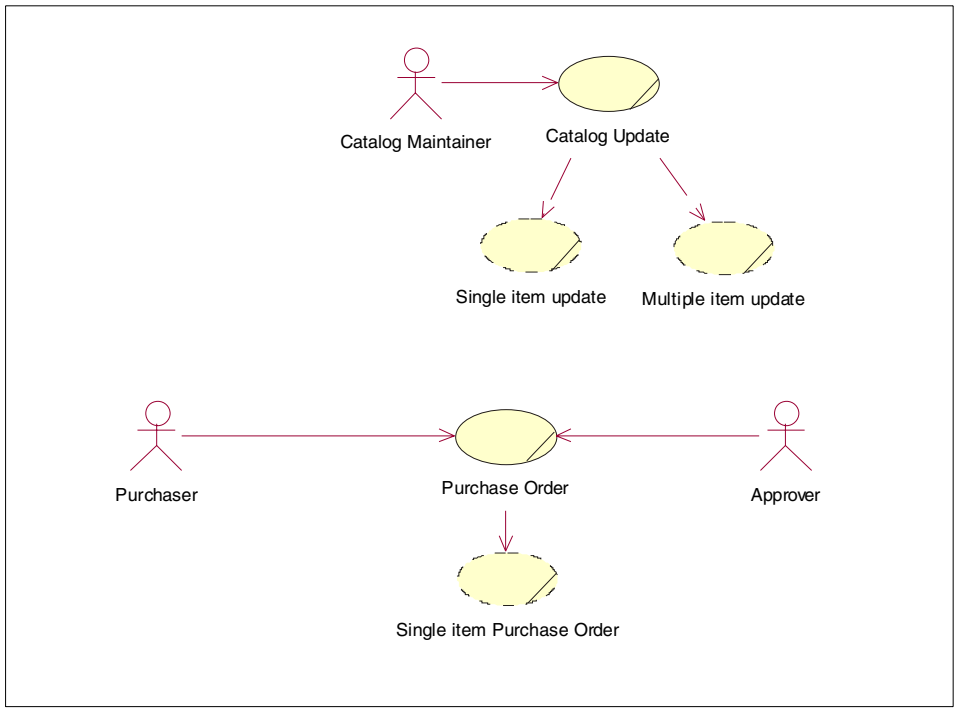


Figure 3-1 Use cases for the sample application

The following are explanations of the terms shown in Figure 3-1:

- Actors

  The list of actors involved in the business scenario, including the following:

  – Catalog maintainer

    The actor who maintains the catalog can submit an update request to the system. Requester can update one catalog item or multiple catalog items, where the item details for update are stored in an XML document.

    The actors can be business users or employees. Application server components can also invoke update asynchronously.

  – Purchaser

    The purchaser actor can submit purchase orders to the system. The orders are very simple documents that may only have one item on them at a time. Purchaser can initiate the PO from a Web client by providing the details for the order.

– Approver

Who may approve a PO is decided by the system based on the total price of the order. The approver is responsible for providing an answer to the system, whether the PO is approved or not.

Approvers may refer to helps built into the Web client when deciding whether to approve the PO, including:

- Search capabilities to track previous POs in the system. The criteria can be set dynamically for the search.
- Search capability to check multiple catalogs to find better prices for the item on the order.

► Use cases

A list of use cases for the business scenario, including the following:

– Catalog update

Catalog update is one of the main use cases. It updates only the price for catalog items. Catalog maintainers can initiate catalog updates in the system. There may be two child use cases based on the number of updated items.

- Single item update

  The single item update use case is a descendant of the catalog update use case. It updates the price for only one item in the catalog. This method requires the item details for the update.

- Multiple items update

  The multiple item update use case is a descendant of the catalog update use case. It updates the price for multiple items in the catalog. This method requires a URL for an XML document with the item details.

The major difference between the two sub-use cases is the method of invocation. The single item update is performed through a Web user interface (UI), while the multiple item update is invoked internally from the system.

– Purchase order (PO)

The PO use case is a business process, where the purchaser can submit a PO to order a certain amount of an item from the catalog. Based on the total price of the order, the PO may require approval from another business user, for example someone from the financial department.

The PO can be submitted using a Web client. The request needs an item number pointing to the item in the catalog and the quantity.

– Single item Purchase Order

  The sample application only handles orders with one single item. This descendant use case is an implementation of the parent PO business process.

3.4 Business processes

The implementation of such use cases is done using business processes. During design, the business processes are realized using process flows in UML design.

The business processes can be implemented on an application level as workflow or process flows. This is a subject for later discussion.

There are two business processes implemented in our sample application:

- Catalog update process
- Purchase Order (PO) process

The following two sections give detailed description of the two business processes on the design level.

3.4.1 Catalog update business process

The catalog update is a fairly simple process. It is responsible for updating items in the catalog that are the sources of items for the purchase orders.

Requirements:

- Catalog update can be initiated from a Web client.
- Update can be initiated in two forms:
  - Updating one item providing the update details
  - Mass update providing a document that has the details
- Process should return the number of updated items in the catalog.

Note: We will not implement other functions related to catalog maintenance, such as add item or remove item. We assume that there is a catalog already existing in the company and the necessary functions are already implemented or will be implemented.

Figure 3-2 on page 30 is the UML representation of the catalog business process.

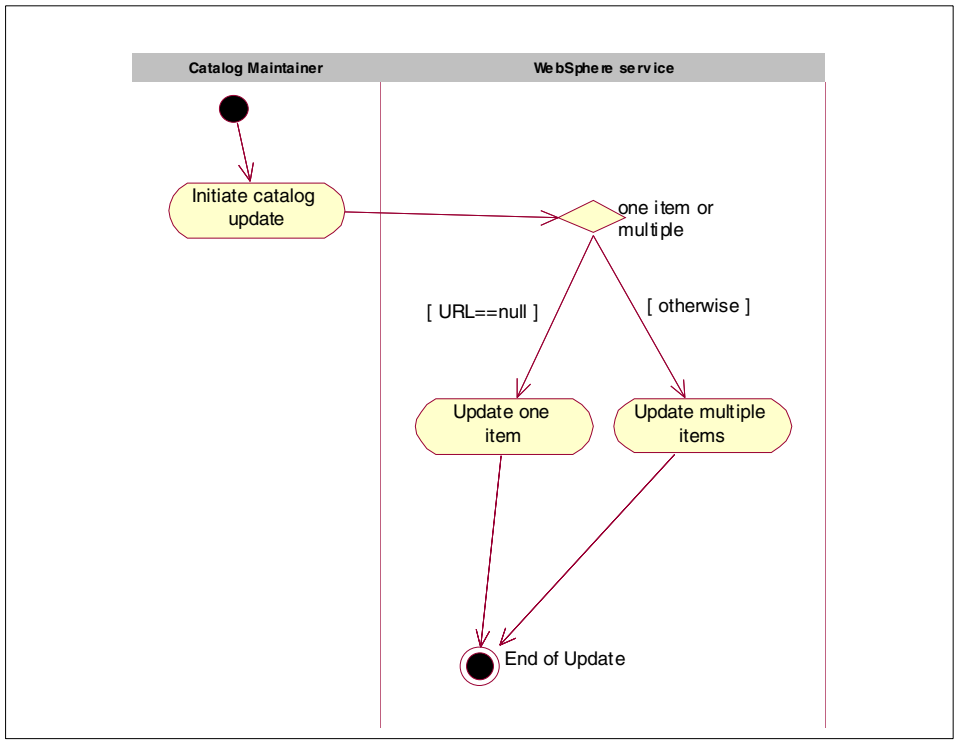


Figure 3-2 Catalog update process

3.4.2 Purchase Order (PO) business process

The purchase order (PO) process is a more complex process.

Note: From the implementation point of view, the PO process is very simple. The objective of this book and this process is to show the runtime capabilities of WebSphere Enterprise. If you want to learn more about the Process Choreographer and its capabilities, refer to the IBM Redbook *Exploring WebSphere Studio Application Developer Integration Edition 5.0*, SG24-6200, or go to Chapter 4, “Process Choreographer development scenarios” on page 35 in this book.

Requirements:

- Purchase order can be initiated from a Web client.
- Purchase order can be initiated by any employee.
- One purchase order includes one item from the catalog.

Note: This requirement would not be necessary in a real application. Again, the objective here is to show the runtime capabilities and make the sample application as simple as it can be.

In a real application, one purchase order could have multiple items and it would require more complex database structure and more application components.

- Process needs to be intelligent and capable of making a decision whether approval is required for an order. The decision is made based on the total value of the order.
- The approval is initiated from a Web browser.
- The result of the process is an order in the company's back-end application.
- At the end of the process, the process originator gets a message about the outcome of the process.

Figure 3-3 is the UML representation of the purchase order (PO) process.

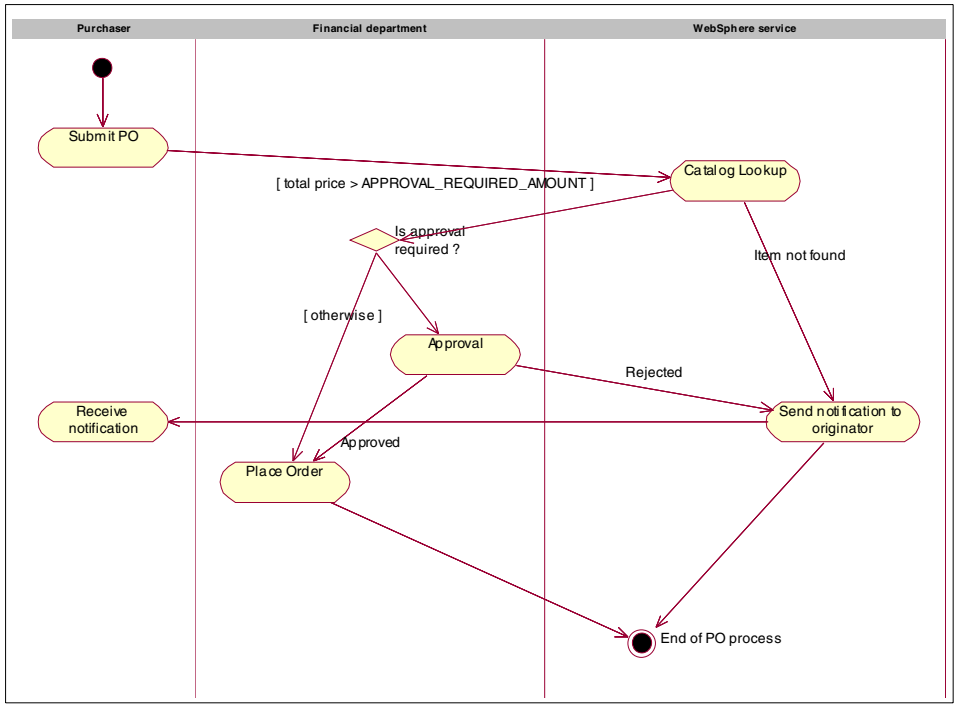


Figure 3-3 Purchase Order (PO) process

You can find more details about the implementation of the sample application in later chapters.

For more information about the development and runtime environment for the sample application, refer to Appendix B, “Sample scenario” on page 665.

Part 2

Programming Model Extensions

4

Process Choreographer development scenarios

The WebSphere Enterprise Process Choreographer is a fairly large topic to cover. Therefore the discussion of this extension (PME), unlike others in this book, is split into two consecutive chapters. One covers development, while the other describes the runtime environment.

This first chapter on Process Choreographer comprises the following topics:

- Planning and describing the Process Choreographer function
- Design
- Development
- Testing and debugging
- Using the staff support service
- Describing the deployment steps of the sample application

4.1 Planning

Today's companies are changing very quickly, so they frequently have to change, add, or rearrange their business processes, which in turn are reflected in changes in their applications and databases assets, which are usually scattered throughout the organization across multiples platforms and software assets, such as databases, packaged applications (ERP, etc.), and back-end systems such as IBM CICS, IBM IMS™, J2EE, .Net, etc. A good example of business change that affected almost all companies worldwide was the advent of the Internet as a new business channel. In responses to this new development, companies have made lots of changes, new applications, adaptations and so on in their software and hardware assets.

4.1.1 Business processes

There are many existing IBM products providing workflow technology for many different situations, scenarios, customer skills, and implementations. IBM provides this technology in many products such as WebSphere MQ Workflow for people-based workflow, WebSphere MQ integrator for message flows, Lotus® Domino™ Workflow for people-based workflow (tightly integrated with Domino), but none of them are tightly integrated with WebSphere. Business Process Choreographer brings flow support into the application server, allowing customers to easily combine workflow technology with all other services offered by J2EE, in addition to some of the new enablements of WebSphere Application Server Enterprise V5 (see Table 4-1), tightly integrated within WebSphere. The advantage of this tight implementation is that many features of WebSphere Application Server, such as centralized management, workload management, logging, debug and trace, can be used when executing a business process.

Table 4-1 WebSphere Enterprise extensions

Enterprise enablements	Description
Extended Messaging	Enhances standard J2EE Messaging by providing support for all types of messaging patterns, container support for these patterns, and code simplification. See Chapter 6, "Extended Messaging" on page 227 for more information.
Asynchronous Beans	Allows J2EE applications (EJBs) to start other threads and transfer their J2EE context to those threads. See Chapter 7, "Asynchronous Beans" on page 287 for more information.

Enterprise enablements	Description
Application Profiling	Allows you to define different Access Intents depending on the EJB client that is accessing a certain Entity EJB, allowing a lot more flexibility in the way you can tune data access. See Chapter 8, “Application Profiling and Access Intent” on page 343 for more information.
Activity Session	Offers long-running transactions semantics without requiring XA support for all the resources involved in the unit of work, but without data integrity. You can keep EJBs active through multiple transactions and have them passivated at the end of the activity session. See Chapter 9, “Transactional Services” on page 385 for more information.
Last Participant Support	Allows J2EE application to have a single-phase resource (only one per transaction) in a transaction. See Chapter 9, “Transactional Services” on page 385 for more information.
Business Rule Beans	Extends the scope of the WebSphere Application Server Enterprise to support business applications that externalize their business rules. See Chapter 10, “Business Rule Beans” on page 419 for more information.
Dynamic Query	Extends J2EE EJB QL. You can formulate queries at runtime, select multiple EJB attributes out of a CMP EJB in a single SELECT clause, support for GROUP BY. See Chapter 11, “Dynamic Query” on page 457 for more information.
Startup Beans	A special kind of EJBs that are executed automatically when an application starts up or shuts down. See Chapter 12, “Startup Bean” on page 489 for more information.
Scheduler service	Allows a J2EE application to schedule the execution of tasks in the future. See Chapter 13, “Scheduler service” on page 509 for more information.
Object pools	Enables an application to avoid creating new Java objects repeatedly. See for Chapter 14, “Object pools” on page 537 more information.

The support for Process Choreographer has been designed in such a way that compliance with the BPEL (Business Process Execution Language) industry standard can easily be met in the future. The Business Process Choreographer is implemented in pure Java, which runs on the application server. It has been

designed with a flexible and extendable architecture to allow the support for many different technologies while executing the business process.

4.1.2 Why use Process Choreographer?

Building applications using Process Choreographer architecture will reduce the time and cost of building new or changing existing applications in the following ways:

- Easily integrates services in heterogeneous environment. We can build a flow compound of activities in .Net, and back-end systems such as CICS and Java EJBs. You can reuse business logic of your back-end asset instead of rewriting it.
- Easily add new activities in an existing application process. If a bank needs to add a new level of approval in its loan process, they just have to add this new activity to their loan application.
- Easily build new application flows using existing activities. If a bank builds a new application that, for example, entails withdrawal of money from an account, they can reuse an activity that they have in the ATM or cashier application, instead of rewriting one. This is a high level of reuse, called service reuse, that will be easier to achieve than a low level, such as classes reuse.

Figure 4-1 on page 39 shows a process flow compound of five activities. As you can see, activity 1 (a Java class or an EJB) has its output connected to activity 2 (a CICS application) and activity 3 (other process flow). At the end of activity 1, both activities 2 and 3 could be executed in parallel or just one of them based on the condition specified in the connections. Process flow will wait in activity 4 for a human decision to go to activity5 (a .Net logic).

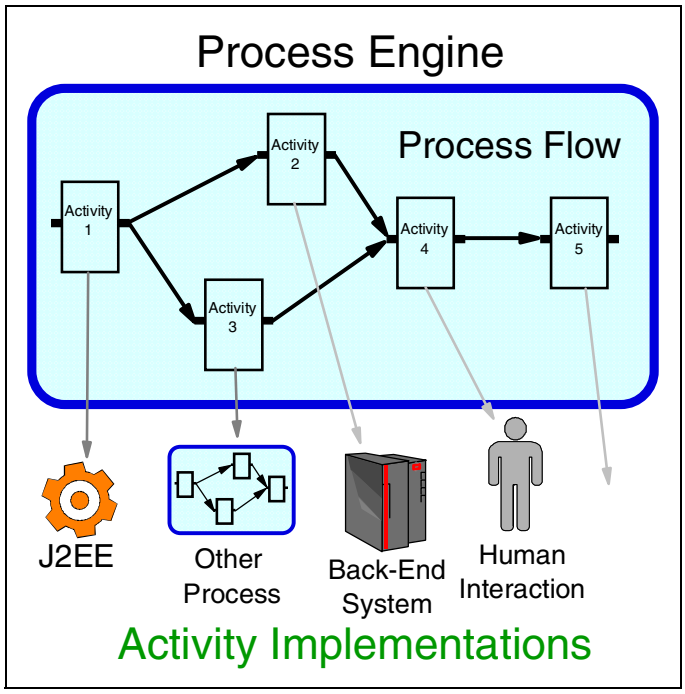


Figure 4-1 Structure of a business process-based application

In our daily activities and businesses, we repeatedly perform sequences of activities to achieve an objective. We model such well-defined and repeatable patterns of activities as processes. A business process typically involves a mixture of human activities and automated activities performed by software applications, computers, and machines. Business enterprises have been making great strides in productivity and profitability as they improve their ability to model, implement, and automate efficient business processes.

4.1.3 Comparison with WebSphere MQ workflow

WebSphere Process Choreographer provide similar features for business process support that has long been available in WebSphere MQ Workflow product. In the following, we point out the differences in programming model and supported features between the two products to assist in deciding how to employ these two products based on an enterprise's business process needs.

Table 4-2 Comparison of WebSphere MQ workflow and Process Choreographer

Feature	Supported by WebSphere MQ Workflow?	Supported by Process Choreographer?
Microflows	No	yes
Macroflows	yes	yes
Events	yes-via WMQI	yes
Timer	yes	yes
Compensation	no	yes
Audit and log	yes	yes
Plug-in of data, mapping, staff	no	yes
Synchronous process invocation via.	API call	façade EJB (API call)
Asynchronous process invocation	XML message	façade MDB(JMS,XML)
Synchronous activity invocation	PEA (API)	Web services, Java beans, EJB, J2C
Asynchronous activity invocation	UPES(XML)	JMS(XML)
Programming interfaces	C, C++, Java, ActiveX, COBOL/390, MQ/XML, JMS/XML	Java, JMS/XML, MQ/XML
Process testing/debugging	runtime test environment	within WebSphere Studio IE V5
Process monitoring	yes - via IBM Holosofx®	not yet
Process modeling and simulation	yes - via IBM Holosofx	not yet -may use UML
Open process execution language	yes - FDL	not yet - BPEL4WS
Rapid deployment	JSP generation from model	no

Feature	Supported by WebSphere MQ Workflow?	Supported by Process Choreographer?
CrossWorlds® connector	yes	no, workaround via JMS and CrossWorlds JMS/WebSphere MQ connector
JSP/Servlet Web client	yes	yes
Native windows client	yes	no
Instance monitor	yes	no
External staff repository	yes(3.4)	yes
Notifications/escalations	yes	no
Expiration	yes	yes
Substitution rules	yes	no
Suspend/resume	yes	no
Work item transfer	yes	no
Process repair	yes	no

WebSphere MQ Workflow is a proven technology providing high performance, scalability, reliability, and availability over the years. No J2EE skills are required and there is little or no requirement for new application development. It offers a tops-down approach to managed process-integration solutions including systems and people and process monitoring via IBM Holosofx. MQ Workflow is available with native implementation on the z/OS platform, exploiting z/OS features and quality of service.

Process Choreographer is fully integrated into WebSphere Application Server Enterprise and WebSphere Studio IE. It exploits WebSphere Application Server features such as clustering and load balancing for scalability, availability, and failover. WebSphere Studio IE provides a visual process editor fully integrated into the WebSphere runtime environment. It also provides a visual process debugger, in addition to a standard Java debugger. Based on full support of the J2EE programming model and open architecture, it allows J2EE programmers to create complex composite services and processes rapidly. It also provide compensation services for long-running processes.

Process Choreographer simplifies the building of new Java-based applications that must integrate with existing software assets.

It offers a bottoms-up approach to developing composite applications that are ready for integration.

You may use WebSphere MQ Workflow and WebSphere Enterprise together. J2EE, JMS, and Web services standards connect WebSphere MQ Workflow and WebSphere Enterprise. Future releases will enhance interoperability between the two products.

4.1.4 Business processes for J2EE programmers

Business process modeling and analysis at the enterprise level are traditionally done by business analysts. Process models designed and defined by business analysts drive the requirements for business functions to implement the essential steps of a business process. These requirements drive design and development of services and applications, which are typically performed by people with programming skills. We optimize business processes through feedbacks and interactions between various levels of implementation, from administrators monitoring the execution of business processes, to programmers, to business analysts.

The Process Choreographer feature of WebSphere Enterprise enhances the J2EE programming model by providing facilities to model, develop, test and execute business processes. Based on service oriented architecture, Process Choreographer allows you to present software components as services and integrate these services into business processes. Business processes themselves are exposed as services, allowing further integration. With GUI-based support and wizards in WebSphere Studio IE, you can now develop new services and processes and perform complex integration of new and existing software assets into processes with very high productivity.

Process Choreographer delivers business process modeling and programming capabilities, traditionally reserved for business analysts, to J2EE programmers in a familiar and integrated environment. With Process Choreographer, you can build J2EE components that implement business functions, then expose these business functions as services described by open standards such as WSDL (Web Services Description Language). You then choreograph these services into business processes using a visual process editor provided by WebSphere Studio IE.

These applications contain business process models called business process-based applications or simply business process applications.

The Process Choreographer runtime environment is provided by a business process engine (BPE) or process container. Business process applications offer

advantages over traditional applications, because they have a number of properties that are guaranteed by the business process engine.

These properties are:

- Concurrency.

  If a process contains parallel branches, the middleware guarantees that the branches are executed concurrently in parallel threads, possibly even on different nodes in a cluster.

- Recoverability.

  If the system crashes while executing a process-based application, the execution of the application is continued where it left off. Steps that have already been performed are not redone.

- Heterogeneous, distributed execution.

  The execution of the individual functions of a business process (its steps) can be distributed in a network, on heterogeneous operating systems and hardware platforms.

- Range of quality of service.

  The process engine supports non-interruptible processes (microflows), interruptible processes (macroflows), and combinations of the two.

In addition to delivering the benefits of business process applications above to business enterprises, Process Choreographer offers the following benefits:

- Flexible and powerful business integration capabilities.
- Ability to develop new services and processes with low cost and short time-to-market, thus allowing you to quickly respond to changes in business requirements.
- Open APIs and a service-oriented approach based on standards, which allow you to customize business process applications to meet widely varying and complex enterprise requirements.
- Efficient business-to-business interaction, because business processes call business processes of partner businesses through standard interfaces such as Web services.

The following sections present the details of how the Process Choreographer development and runtime environment works to provide these benefits.

4.1.5 Programming model

The programming model offered by Process Choreographer can be described as four interacting models:

- Information model
- Organization model
- Services model
- Business process composition model

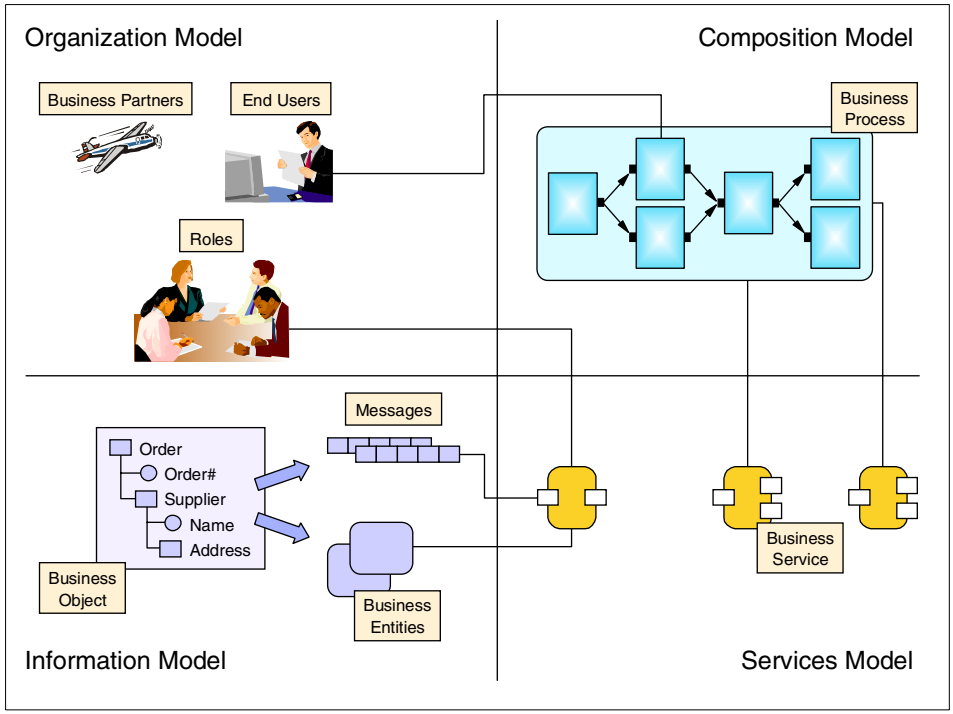


Figure 4-2 Conceptual view of business process-based programming model

The information model in business enterprises comprises business objects that model entities such as customer accounts or purchase orders. These objects may reside in Enterprise Information Systems (EIS) or database systems. In the J2EE programming model, they are modeled into message types or entities such as Enterprise Java Beans (EJB).

A services model involves programming necessary for business functions as services. These services can be implemented as business methods of a session EJB, Java bean methods, or JMS operations. Process Choreographer provide capabilities to expose these business service methods of J2EE components into operations and messages in service oriented architecture described in open standards such as WSDL.

An organization model deals with people in enterprises organized into groups such as departments. People can be organized based on lines of business, lines of products, locations, and business roles. For business applications, you can organize people based on roles in running and accessing services and processes. End users and business partners can be assigned roles for accessing business services and processes offered by an enterprise. Organization models are typically implemented through some form of user registries or directories.

The composition model or business process model involves choreographing business services into a business process. It provides standard interfaces to a business process for end users and business partners. If the process models involve human activities (staff activities), the activities are assigned to people based on organizational roles or groups in user registries.

4.1.6 J2EE programming model

The traditional J2EE programming model allows separation of back-end data and application logic by introducing the concept of application servers. A typical enterprise application consists of EJBs that handle business logic, and servlets and JSPs to support the presentation layers. Application servers provide runtime containers that provide services for transaction management, security, etc. It utilizes the organization model through a user registry to manage access to its components and services.

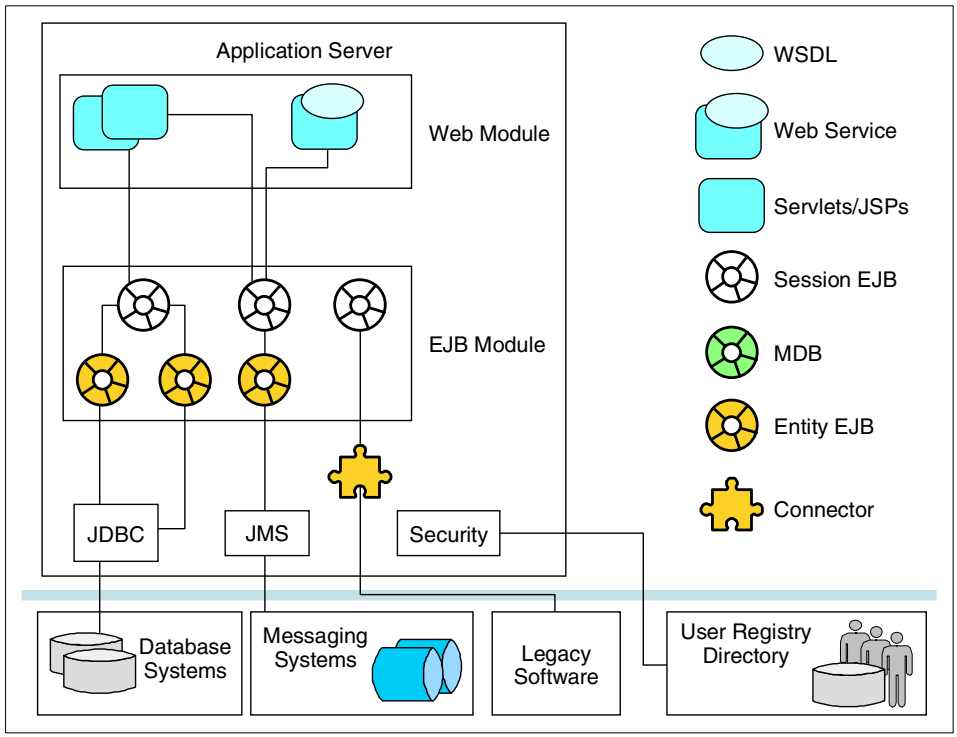


Figure 4-3 Typical components of a J2EE programming model

Process Choreographer extends the traditional J2EE programming model by adopting a service oriented architecture based on the Web services paradigm in WebSphere Application Server Enterprise to allow composition of business processes.

Figure 4-4 on page 47 illustrates the key components involved in implementing the programming model. The following sections provide details about the following components:

- Services and WSDL: service oriented architecture (SOA)
- Business process model
- Programming interfaces
- Process container architecture

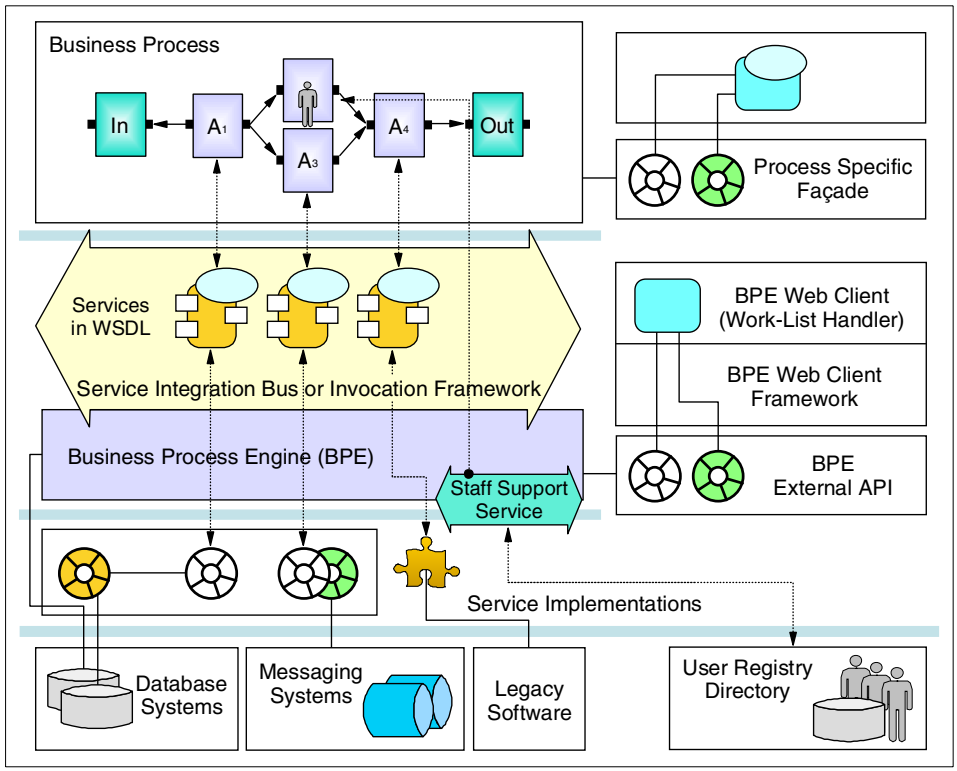


Figure 4-4 Components of Process Choreographer programming model

4.2 Design

A business process is a set of parallel or sequential activities that can be performed repeatedly to achieve business objectives. Activities can be performed by machines, computer software and hardware, and human beings.

- A process template or process model describes the structure of a business process in the real world. It defines all possible scenarios in a business process, including various activities that may take place and the rules and conditions that determine which paths within a business process should be taken.
- The process model is a template from which each process is instantiated; that is an instance of the process model is created.

Part of a process model that can be executed on a computer is called a workflow model. The term workflow is often used interchangeably with the term business

process. We will use either business process or process in our descriptions of Process Choreographer.

4.2.1 Elements of a process model

The top-level construct of the business process model is the process itself. A process is a multi-step operation. A graphical representation of a process is a directed graph. The major constructs in this process graph are activities and control connectors. The activities describe the tasks to be performed, and the control connectors describe the potential sequence in which the activities are to be carried out.

Types of activities

There are different types of activities for addressing different types of tasks. Process Choreographer supports the following types of activities:

- Service activities
- Java snippets
- Staff activities
- Receive event activities
- Process activities
- Empty activities

Service activities

Service activities are used to invoke operations or services.

There are two supported styles:

1. Synchronous service activity, which invokes an operation or a method and blocks until it returns. Examples are:
   - Invoking an EJB method
   - Invoking a Java bean method
   - Calling a J2EE connector
   - Calling a Web service

   This category includes elemental activities that just fire and forget an invocation. Examples are:
   - Sending a JMS message without waiting for a response
   - Sending e-mail
2. Asynchronous service activities, which send and commit a request. The response is received and handled in a separate transaction. An example is a

service activity with a JMS message binding that receives a response later in a separate transaction.

A process engine will wait until the response with a correlation ID is received in the reply-to queue (which is set to the process container's input queue) before proceeding to the next step.

Java snippets

Java snippets are synchronous activities that represent an inline invocation of a snippet of Java code. They are usually used for mapping of data between variables used between other types of activities. They run in a Java runtime environment of the Process Choreographer and they have access to the corresponding business process external APIs (BPE API in com.ibm.bpe).

Staff activities

Staff activities, also known as person activities, model tasks and interactions performed by people in a process. They are asynchronous in nature. A staff activity can be assigned to persons in three different roles: readers, editors, and potential owners. When a staff activity is reached in a process, a work item is created for a person or a group of persons assigned to the activity. Determining the persons who can work on the work item from a registry of persons according to the roles specified in the process model is called staff resolution. A person with a particular role in an activity can check whether a work item has been created for him or her. The person can perform tasks in his or her role by claiming the work item. When the person completes the work item, the resulting data becomes available to the workflow system and the process continues to the next activity.

Receive event activities

Receive event activities allow the process to wait for external events. These can be used for synchronization with external programs or processes. For example, a process handshakes with another program (which may be a partner process), sends off messages, continues with other activities, and then eventually waits for an external event to continue its operation. Events are sent to a process using the Process Choreographer API. A unique process ID may be used as a correlation in selecting the correct message or event for the waiting process.

Process activities

Process activities are used for nesting processes. A process can call other processes that call other processes, and so on. This construct allows you to create hierarchical processes. It also allows you to reuse process logic by factoring it out as a subprocess.

Empty activities
Empty activities do not perform operations. Instead they act as explicit synchronization points for parallel branches in a process, or to allow explicit join or branch nodes.

Variables

Activities in a process manipulate data; input data is passed to a process and output or fault data is returned by the process. Within the process, data is kept in global variables. Mapping activities, implemented by Java snippets, can be used to transform data. These mapping activities take their input from one or more variables. The result is written to another set of variables. To invoke an activity that requires an input message, variables are read. If the activity produces output or fault data, then this data is mapped back to the same or another variable.

Blocks and loops

Blocks allow grouping a set of activities in a process. Loops are a special form of block that allow the block to be traversed repeatedly until a loop exit condition is met. Loops and blocks do not have an interface. Data does not flow into or out of them, and activities within them have access to variables defined within the top-level process.

4.2.2 Types of processes and transactions

Processes can be categorized into interruptible processes and non-interruptible processes.

Process Choreographer does not implement any transaction service itself, but it participates in the standard WebSphere transactions. For non-interruptible processes (microflows), the transaction that is established by the client of the process is used for the entire process and all service invocations performed on its behalf. For interruptible processes (macroflows), there is an initial transaction established by the client of the process, plus a number of chained transactions established by JMS messages sent by the Process Choreographer engine to itself or by requests from people participating in the process. Each one of these transactions is a standard WebSphere transaction. In case of a failure, the rollback will act only in the current transaction, not in the ones that have been committed during the process. And there are some asynchronous activities that cannot be rolled back. For example if the activity was mailing a letter, the best we can do is to mail another. In this situation, Process Choreographer implements a feature called *compensation*.

Non-interruptible process (microflows)

Non-interruptible processes have the following characteristics:

- They are short-running.
- They run in one transaction **T** (see Figure 4-5 on page 52), created either by the client of the process, or by WebSphere at the boundary of the Process Choreographer engine (deploy descriptor with transaction attribute of TX_REQUIRED).
- They can contain activities with transactional and non-transactional implementations.
- They cannot have asynchronous activity implementations or activities that involve human interaction. Asynchronous request-response operations require that transaction commits for the request get delivered first, and then the response of the request is processed in a new transaction.
- They can send notifications, because a response is not expected for asynchronous notifications. Receive event activities require their own transactions for receiving an event and therefore are not supported.
- All activities in a non-interruptible process are processed in a single thread, so they all share the same thread context.
- All process states are maintained in memory during the execution.

When a transaction rollback occurs, the transaction manager rolls back all the transactional activities of a non-interruptible process. Activities with an implementation that is not transitional remain untouched.

Non-interruptible processes provide developers with the benefits of visual modeling (using WebSphere Studio IE), easy maintenance, and graphical debugging. They can be used in all cases where a developer needs to write code for the interaction of enterprise beans, Java classes, J2C connectors, and Web services.

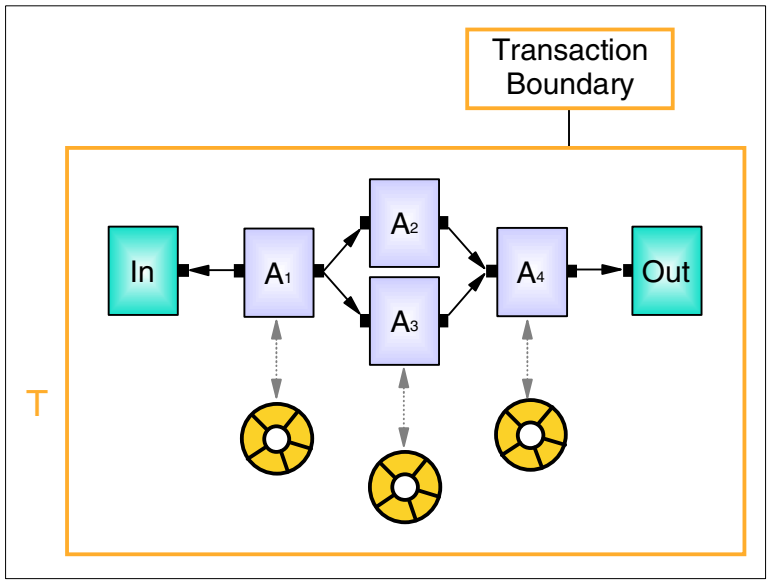


Figure 4-5 Non-interruptible microflow

Asynchronous request-response operations require that a transaction commits to send the request. Processing the result of the request is then performed in a new transaction. A response is not expected for asynchronous notifications. Therefore, sending notifications is supported in non-interruptible processes. Receive event activities require their own transaction for receiving an event and therefore are not supported within non-interruptible processes. All activities in a non-interruptible process are processed in a single thread. Therefore, all activities share the same thread context.

The Process Choreographer in WebSphere Version 5.0 supports only transactional non-interruptible processes. When a transaction rollback occurs, the transaction manager rolls back all of the transactional activities of a non-interruptible process. Activities with an implementation that is not transactional remain untouched.

Interruptible processes (macroflows)

An interruptible process can navigate to an activity which, for example, requires human interaction or a response from a remote service and wait for hours, days, or even years until the expected event occurs.

An interruptible process consists of a set of stratified transactions. This means that in a process that contains a set of activities, each navigation step is performed in its own transaction.

Interruptible processes support all kinds of activities. The actions performed in the transaction depend on the semantics of the respective activity type. To reliably hold the navigation information of a process, the database stores the persistent state of the process and the messaging system holds the persistent navigation state, that is, the information about which activities in the flow will be navigated to next. The database, the messaging system, and the transactional resources used by an activity implementation all participate in a two-phase commit protocol. Because the complete state of a running process is stored in the database, a process is not dependent on a specific application server. Interruptible processes have a much larger footprint than non-interruptible processes because they use persistent storage and transactions.

When an interruptible process runs in a cluster, each process step can be performed in parallel on any node. The load is automatically distributed over the different servers.

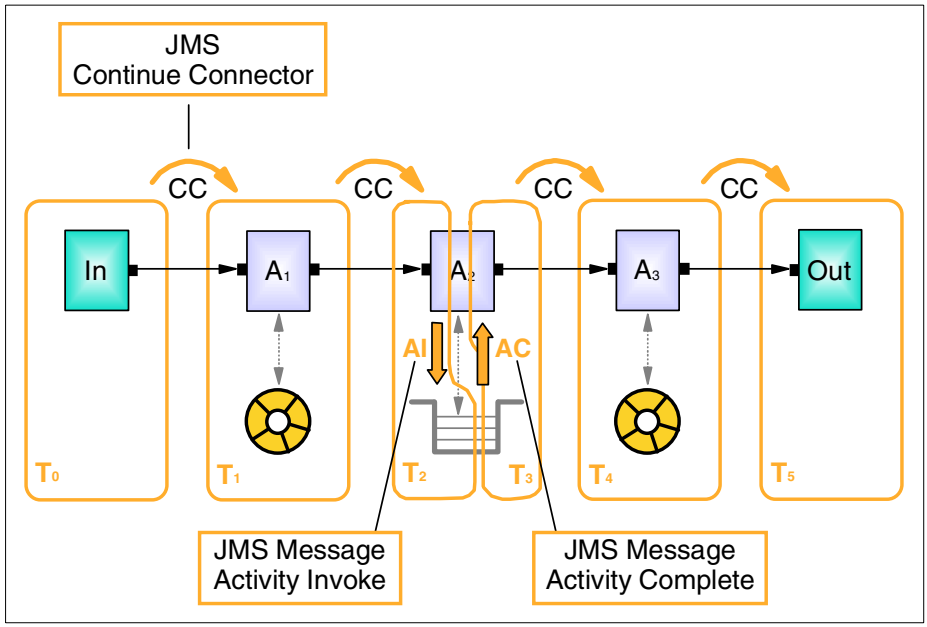


Figure 4-6 Macroflow with synchronous and asynchronous service activities

Transaction boundaries of stratified transactions depends on the type of activity invocation.

- Synchronous service invocation

  The following actions occur when an activity that performs a synchronous service invocation (for example, activity A1 in Figure 4-6) is processed:

  a. Begin the transaction (T1).

b. Get the message carrying the Boolean value of the connector from the persistent process container internal message queue.

c. If all incoming connectors have fired, check the start condition:

- If true, start the activity A1, that is, invoke the synchronous service.
- If false, skip the activity.

d. Put a message for each outgoing control connector in the persistent process-engine message queue.

e. Write the state changes to the database.

f. Commit the transaction (T1).

If an activity is non-transactional, that is, its activity implementation does not participate in the two-phase commit protocol of the transaction, compensation-based recovery can be used to undo changes performed by the activity if a failure occurs.

- Continue control connector

  In Figure 4-6 on page 53, T1 and T2 are connected by a “Continue control connector” (CC) message. In this example, when following the control connector between A1 and A2, the flow engine sends itself this message in order to be able to commit T1 and guarantee continuation of the process with T2.

- Asynchronous service invocation

  Again in Figure 4-6 on page 53, A2 is implemented by a service invoked asynchronously via an “Activity invoke” (AI) JMS message. Once this message has been put into its destination queue, the flow engine must commit T2 so that the message actually gets delivered. After the service implementation has eventually processed the invocation message, it returns with an associated reply, called “Activity complete” (AC) message in this example. The receipt of this message triggers T3, which completes A2 and continues navigation of the flow along the control connector to A3.

- Staff activity being acted upon by assigned people

  A complex interruptible process with staff activities is given in Figure 4-7 on page 55 with transaction demarcations. In this process, A1, A3, A7 and A8 are synchronous service activities invoking EJB methods. A2 and A5 are asynchronous service activities with JMS invocations. A6 is an empty activity with no operation used for synchronization. A4 is a staff activity.

  At the fork after activity A2, the conditions for both outgoing control connectors are evaluated to true; hence both branches (A2 to A4 and A3 to A5) will be traversed. The activity A4 on an upper branch is completed before A5 of the lower branch. The process will synchronize before continuing with

activity A6 to make sure conditions for both incoming control connectors (from A4 and from A5) are evaluated.

The staff activity A4 is to be implemented by a person with the role of a potential owner. When the process reaches A4, the process engine creates a work item with the activity input message and distributes it to the potential owner for the activity (this step is shown as a “work item creation” (WC) external message in the figure) before committing the transaction. Eventually, the person checks to see the assigned work item, claims the associated activity (hence becomes the owner of the activity). The works with the associated data and documents and finally completes the activity by sending an “activity complete” (AC) message through the process engine API. The receipt of this message triggers the following transaction, which continues the process.

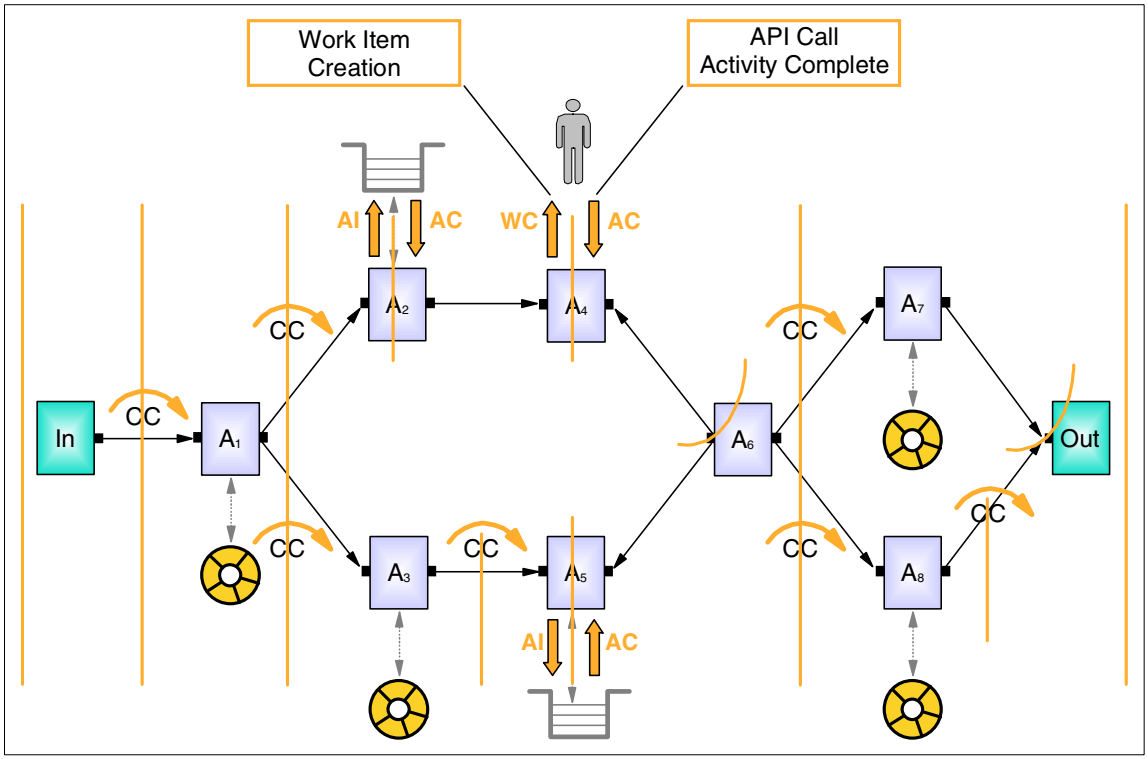


Figure 4-7 Complex interruptible process with human activities

Interruptible processes are completely forward recoverable. If the application server that is processing an interruptible process terminates unexpectedly, no information is lost. The process either continues on another node in the cluster or, in non-clustered applications, waits until the server comes up again.

Interruptible process invoking microflows

We can have a long-running process that invokes several microflows as process activities.

Microflow process with human exception handling

We can have a microflow process invoking macroflows to let people handle exceptions. microflow processes and macroflow business process models can have

4.2.3 Life cycle of a process

A process comes to life when a Process Choreographer API method that can start a process is invoked. The supported API methods are call and initiate.

When initiate is invoked, a response is not returned to the caller when the process finishes. However, by using people, an asynchronous or receive event activities interaction takes place between a process and the process starter, other people, or other services.

When a call is invoked, the process starter receives a response when the process finishes. The response mechanism can be as simple as returning a result from a synchronous non-interruptible process. Alternatively, a reply context can be provided when the process is called. This context is used by the process engine when the process ends to send back the response.

When a process is started, a process instance for an existing process model is created and started. The process engine then starts navigating the business process. That is, the process engine determines which of the activities of the business process are activated and in which order. Regular navigation of a process continues until all of its activities are in their end state. Valid end states are finished, failed, expired, or terminated.

A process ends either when one of its fault terminals is navigated or when all incoming control connectors of the process sink have been evaluated. If the process was started using a call, a response is sent back to the caller of the process.

4.2.4 Undoing service activities: compensation

Today's applications typically require transactional properties, in particular, the guarantee that a complex request is executed either in its entirety or not at all. For traditional transactions, this is described by the ACID properties: atomicity, consistency, isolation, and durability. It is achieved by a transaction manager, resource managers, and back-end systems working together according to the XA

standard, or similar, protocol. This cooperation ensures that the operations performed on behalf of a transaction are either all committed or all rolled back.

Often, however, a complex request cannot be run as an ACID transaction for a number of reasons. Back-end systems or resource managers might not be able to participate in the XA protocol. Updates to these systems are performed immediately and do not participate in the overall transaction. If the transaction fails, these updates are not undone; the processing of the complex request results in an inconsistent state.

As long as a transaction has not been committed, none of the changes done on its behalf are visible to the outside world. The isolation property guarantees that they only become visible when the transaction's final state has been reached. This works well for short-running processes that invoke synchronous operations.

However, when a process involves asynchronous steps, for example, steps implemented by a back-end system driven by JMS messages or steps involving human interaction, intermediate results of the process must be made visible. This means that the JMS message must be sent out, the information about the step the person has to work on (the work item) must be made available, and so on. Therefore, the intermediate state of the process must be committed.

There are actions in a business process that are inherently non-transactional. Sending a letter to a customer is such an operation - as soon as the letter has left the sender, there is no way to undo the operation.

Activities in an interruptible process can be transactional or non-transactional. Transactional activities run in their own transaction; however, in an interruptible process there is no transaction that surrounds all the activities in the process. Therefore, no locks are held to ensure the data integrity of the entire process. If the process fails, compensation can be used to undo the changes that were successfully done by the activities.

In all of these examples, intermediate results of the business process are made available and they cannot be undone by simply rolling back an ACID transaction with the help of the transaction manager. Instead another operation must be done that explicitly reverses the original operation.

If a back-end system has been called to update data, for example, to increment a value, the system must be called again, for example, to decrement the value.

If a letter has been sent in error to a customer, another letter must be sent to apologize for the error. These operations might need to be cascaded if other operations have already started that use the results that have been wrongly made available.

To help you to define undo operations and their automatic execution, you can specify the steps of a process as compensation pairs. This means that in addition to the standard forward operation, a backward operation is assigned to the activity, as shown in Figure 4-8.

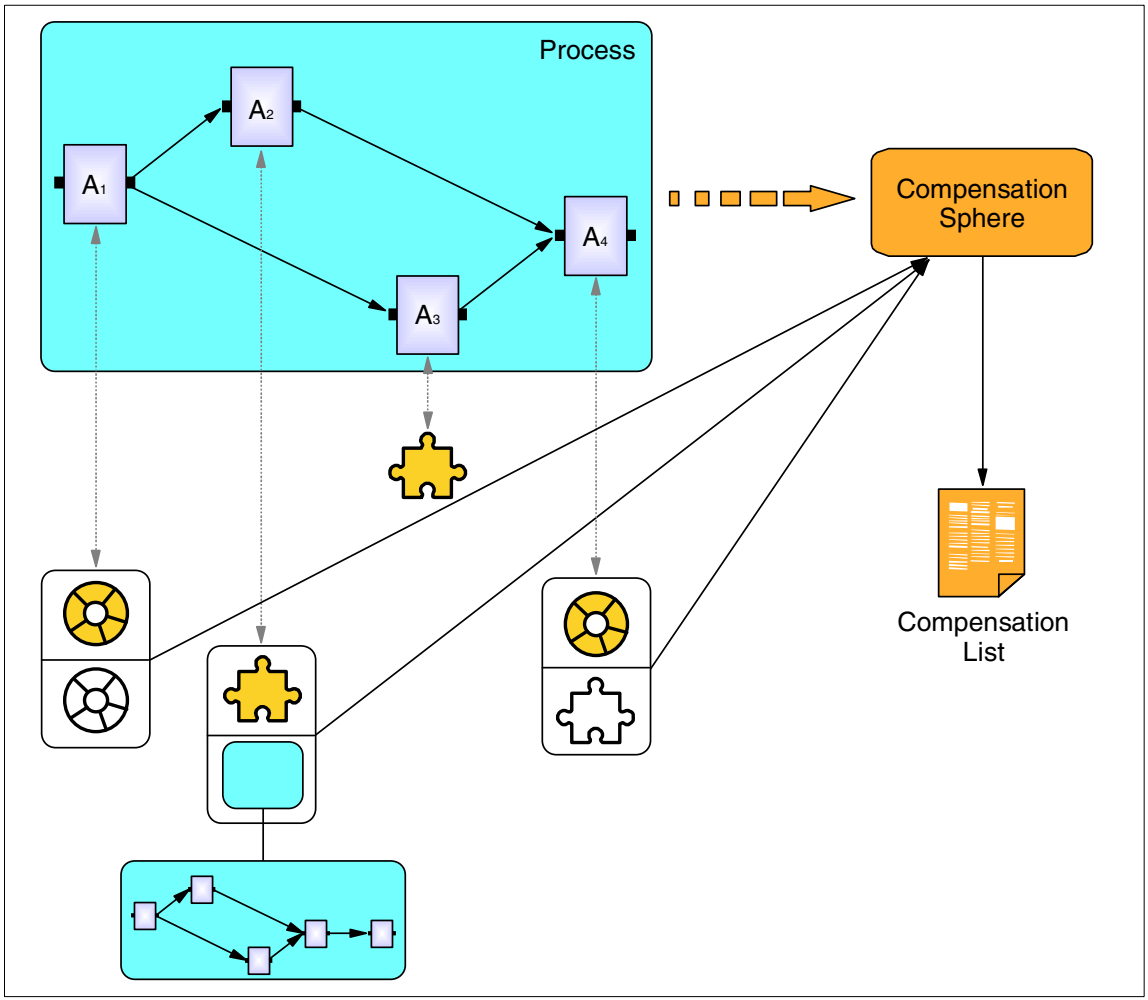


Figure 4-8 Logging of compensation pairs during forward process flow

The process runs using the forward operations. In addition, each invocation of a forward operation is logged with its input and output data in a compensation list. At any time during the execution of a process, you know which operations have been invoked and with which data.

If the process fails during its execution and has to be compensated, the compensation list is used to drive the backward execution of the process to re-establish the previous process state. For those operations that were successfully executed during the forward execution of the process, the associated undo operation is invoked, and the original data is passed. For example, the undo operation might send a compensation letter to tell the customer that a previous letter was in error and should be ignored.

Figure 4-9 shows the execution of the undo actions for the activities that were performed during the forward execution of the process. For activities that have no undo operations defined, and for activities that were not processed during the process' forward execution, nothing is done.

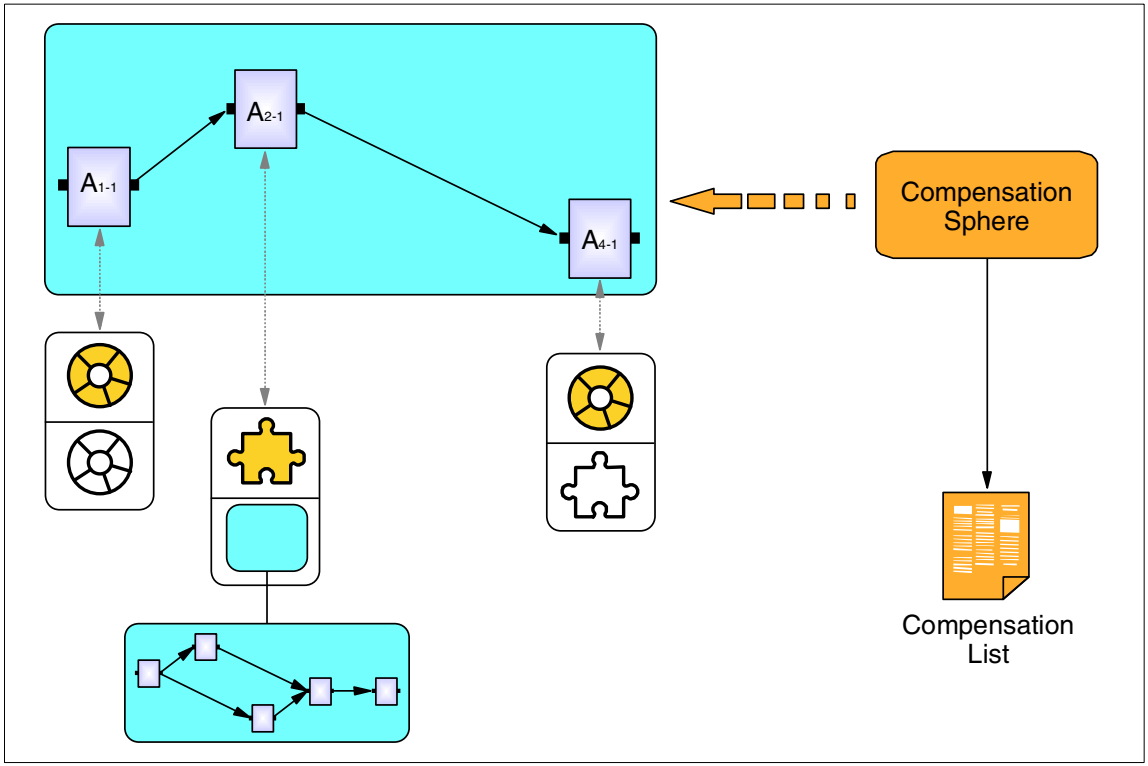


Figure 4-9 Execution of compensating services activities in reverse order in case of faults

4.2.5 Process modeling languages and standards

Process models are described in FDML (Flow definition markup language FDML). It is the markup language based on the process-related aspects of WSFL (Web Services Flow Language). WebSphere Studio IE provides

GUI-based wizards so that a programmer can develop business process applications through graphical maps without ever touching FDML. Hence the use of FDML as a modeling language is transparent to programmers.

Business Process Execution Language for Web Services (BPEL4WS) is positioned to become the Web services standard for business process composition. BPEL4WS provides a language for the formal specification of business processes and business interaction protocols. It extends the Web Services interaction model and enables it to support business transactions. It is being authored by a group of major users, led by IBM Corporation. At the time of this writing, the specification has officially been submitted as a standard specification to the Organization for the Advancement of Structured Information Standards (OASIS). Sun Microsystems Inc. and Novell have announced plans to support the specification.

Currently FDML supports many of the BPEL requirements, but it does not address the entire BPEL4WS standard, and vice versa. IBM plans to support BPEL4WS in the next major release of WebSphere.

4.2.6 External programming interfaces

Process Choreographer provides both general programming interfaces and programming interfaces specific to a process model. General programming interfaces, also referred to as Business Process Engine (BPE) external interfaces, provide programming interfaces independent of a specific process model.

Process-specific programming interfaces are generated by WebSphere Studio IE when you invoke the wizard by selecting **Enterprise Services -> Generate deployed code**.

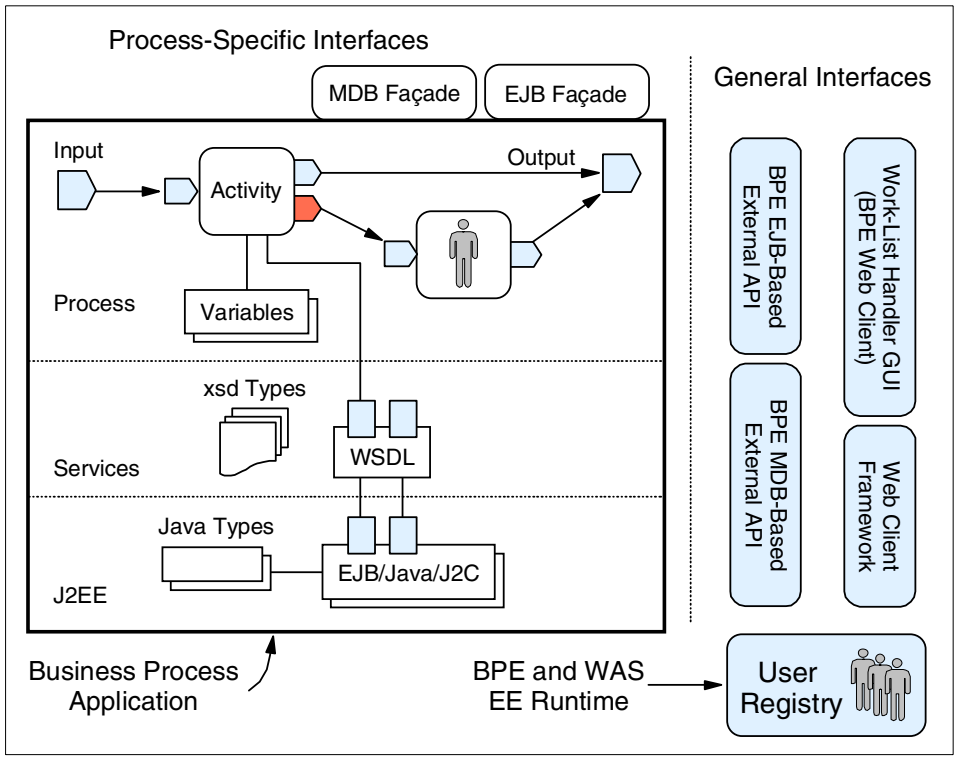


Figure 4-10 General and process-specific programming interfaces

General programming interfaces

For the latest description of these interfaces, refer to the complete JavaDoc for the BPE external interfaces in WebSphere Application Server InfoCenter. In the InfoCenter:

1. Click **Quick reference**.
2. Expand the link **javadoc**.
3. Click **Enterprise Extensions API**.

The interfaces for the business process engine are described in the package com.ibm.bpe.api. The interfaces are implemented by a core set of Java classes.

These general programming interfaces are made available in two forms:

- The complete set of functions and methods of the stateless session EJB that is provided by the business process container.

  This is the preferred method of interaction with the process engine from a client program, such as the supplied Web client.

- The functions for starting new process instances and sending events to running process instances are available via a message-driven bean (MDB). This MDB accepts JMS messages that pass the input or event data in the body of the message, and the additional information, such as the process template name or the event name, in their JMS properties.

  The process engine honors standard JMS properties, such as the specification of a reply queue. A reply to the start process message is sent upon completion of the process instance.

InfoCenter also includes an introduction to programming against the interfaces with example snippets. The JavaDoc can also be found with application server installation at <*WebSphere_root*>\Web\apidocs for Windows systems or <*WebSphere_root*>/web/apidocs for UNIX systems.

The general programming interfaces may be categorized into the following.

- Programming interfaces to work with processes.
- Programming interfaces to work with activities.
- Programming interfaces to work with work items.

Programming interfaces to processes

These are interfaces that allow interaction with processes.

- Start processes

  A new process instance can be started by specifying the name of the process template and the input data to be used for the new instance. In addition, you can associate the process instance with a user-defined ID. This ID can be used later as a secondary key to retrieve the process instance. You can also specify a callback that is invoked by the process engine when the process instance completes. There are two interfaces to start a process:

  - **call**, invoke process, pass inputs and receive result
  - **initiate**, invoke process, pass inputs, no result

- Send asynchronous events to running processes

  For processes instances containing receive event activities, the associated events can be sent using the programming interfaces. The parameters that have to be passed include the system-generated or user-defined ID of the target process instance, the event name, and the event data. The interface is:

  - **sendEvent**, pass asynchronous message into process instance

- Perform administrative tasks

  Programming interfaces are available to find out information about running process instances, to terminate a running instance, and to repair activities that have failed. The interface is:

  - **terminate**, administrative action to terminate process instance

Programming interfaces to activities

These interfaces allow persons with different roles to work with activities. These include programming interfaces to claim an activity for which a person received a work item, and to complete claimed activity. Example interfaces are:

- **claim**, get exclusive access to work with activity
- **complete**, let process continue from a claimed activity
- **cancelClaim**, return activity to other potential owners
- **forceRetry**, **forceComplete**, repair actions in case of fault

Programming interfaces to work items

These interfaces can be used to query work assigned to a person and to retrieve work items, using either SQL-like queries, or predefined worklists. Examples are:

- **query**, query work items, activities or processes
- **getActivityInstance**, retrieve activity associated with work item
- **getProcessInstance**, retrieve process associated with work item

The worklist handler GUI (BPE default Web client) is built on the BPE Web client framework, and the BPE external API provides additional graphical interfaces to administer and work on interruptible processes with events and staff activities.

The general programming interfaces apply mostly to macroflow or interruptible processes. A microflow or non-interruptible process is invoked synchronously in a single transaction. The only part of general programming interface that applies to microflow process is the call method of the Business Process interface that invokes a synchronous operation.

Process-specific programming interfaces (façades)

For each process model, you can use the WebSphere Studio IE tools to generate an associated EJB (EJB façade) and an associated MDB (MDB façade) that provide strongly typed interfaces for starting instances of the process and sending events to running instances. These façades simplify the coding of interactions with a specific process model.

A MDB façade allows clients to invoke the process or send an event to the process upon arrival of a message in a queue being listened to by the MDB through a listener port.

A EJB façade allows clients to invoke the process or send an event to the process by calling a corresponding business method in the EJB. While generating the above two façades, WebSphere Studio IE also generates corresponding WSDL files for these business methods.

The generated Web services interface invokes the process or send an event to the process through the EJB façade.

4.3 Development

This section describes the development steps for the Process Choreographer.

Business process development using WebSphere Studio Application Developer Integration Edition V5 is documented in the redbook *Exploring WebSphere Studio Application Developer Integration Edition 5.0*, SG24-6200.

This section covers a topic not found in the above-mentioned book, the Choreographer Web client.

4.3.1 Choreographer Web client

Business Process Choreographer provides a ready-to-use Web user interface based on JSPs and servlets. This interface allows users to access business processes that are relevant to them. This Web interface can be used as is, without having to implement or change any code, or can be adapted to fit your needs, or you can create a new client user interface from the ground up using the Process Choreographer APIs.

Architecture overview

As you can see in Figure 4-11 on page 65, the Web client architecture is based on the Model-View-Controller (MVC) pattern.

- **Model**: Executes the requested actions, such as starting new business process, claiming activities, and so on. It is called by command implementations and provides an abstracted interface between the WorkflowController and the WorkflowView JSPs.
- **View**: A set of defaults JSPs (WorkflowView JSPs) that provides the HTML response for a particular command implementation, such as displaying activities or presenting results when claiming activities, etc.
- **Controller**: A servlet (WorkflowController) and a set of command implementation classes. The servlet analyzes the command received and, based on the configuration.xml file, dispatches the command implementation class. On return of the command implementation class, based on the content

of the return and the configuration.xml file, it select the JSP that shows the result.

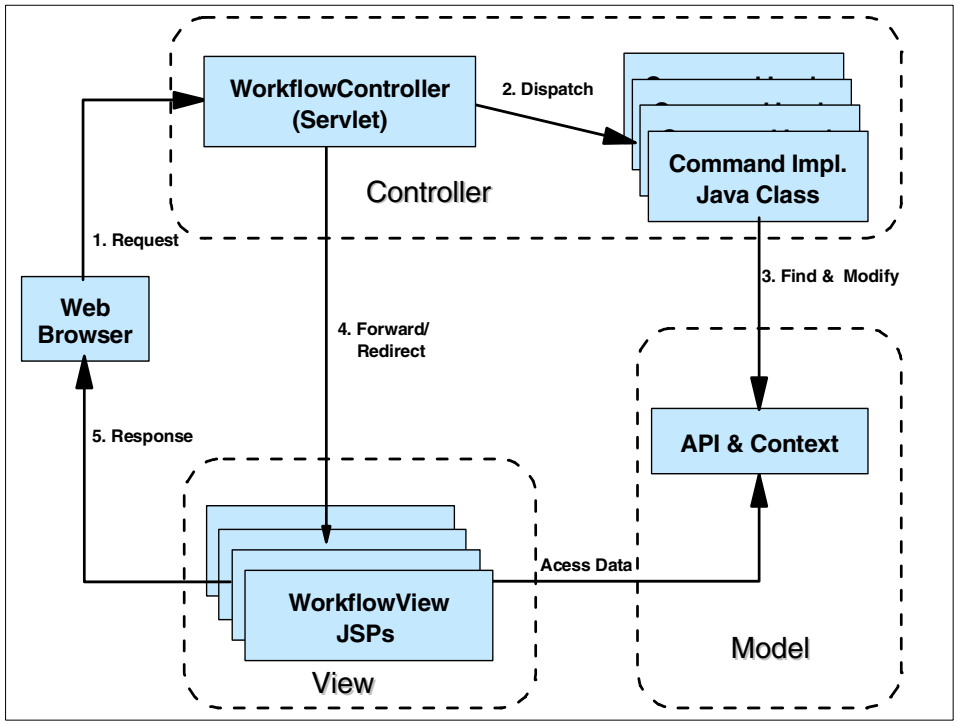


Figure 4-11 MVC architecture of Process Choreographer Web client

WorkflowController Servlet

This servlet handle all requests received from the browser to the Web client. With the HTTP requests it receives commands and parameters as follows:

```
http://localhost/bpe/webclient?WF_Command=ListDisplay&WF_ListName=MyActivities
```

The servlet uses the configuration.xml file to determine if it is an action command or a display command, and then executes the requested action.

- **Display commands**: The servlet uses the configuration.xml file to determine the JSP that serves the command using the parameters received (see Example 4-1). Then this JSP will render the information from ViewContext objects.

Example 4-1 Fragment of configuration.xml file, some display commands

```
<wf:views>
   <wf:definition>
      <wf:parameter name="WF_Command">ListDisplay</wf:parameter>
```

```
      <wf:parameter name="WF_Profile">WSWF 5.0 Default</wf:parameter>
      <wf:parameter name="WF_ListName">MyActivities</wf:parameter>
      <wf:parameter name="Command_Case">Success</wf:parameter>
      <wf:setting attribute="Setting_Type">JSP</wf:setting>
      <wf:setting attribute="Setting_Name">WorklistActivities.jsp</wf:setting>
   </wf:definition>
   <wf:definition>
      <wf:parameter name="WF_Command">ListDisplay</wf:parameter>
      <wf:parameter name="WF_Profile">WSWF 5.0 Default</wf:parameter>
      <wf:parameter name="WF_ListName">Generic</wf:parameter>
      <wf:parameter name="Command_Case">Success</wf:parameter>
      <wf:setting attribute="Setting_Type">JSP</wf:setting>
      <wf:setting attribute="Setting_Name">List.jsp</wf:setting>
   </wf:definition>
   <wf:definition>
      <wf:parameter name="WF_Command">ListDisplay</wf:parameter>
      <wf:parameter name="WF_Profile">WSWF 5.0 Default</wf:parameter>
      <wf:parameter name="Command_Case">Error</wf:parameter>
      <wf:setting attribute="Setting_Type">JSP</wf:setting>
      <wf:setting attribute="Setting_Name">Error.jsp</wf:setting>
   </wf:definition>
</wf:views>
```

- **Action commands**: The servlet uses the configuration.xml file to determine the command-implementation class that serves the command using the parameters received (see Figure 4-2). Then it calls the execute() method of this class.

Example 4-2 Fragment of configuration.xml file, some action commands

```
<wf:commands>
   <wf:definition>
      <wf:parameter name="WF_Command">ActivityCancel</wf:parameter>
      <wf:setting
attribute="Setting_Name">com.ibm.bpe.client.commands.wswf.ActivityCancel</wf:se
tting>
   </wf:definition>
   <wf:definition>
      <wf:parameter name="WF_Command">ActivityClaim</wf:parameter>
      <wf:setting
attribute="Setting_Name">com.ibm.bpe.client.commands.wswf.ActivityClaim</wf:set
ting>
   </wf:definition>
   <wf:definition>
      <wf:parameter name="WF_Command">ActivityComplete</wf:parameter>
      <wf:setting
attribute="Setting_Name">com.ibm.bpe.client.commands.wswf.ActivityComplete</wf:
setting>
   </wf:definition>
```

```
   <wf:definition>
      <wf:parameter name="WF_Command">ActivityForceRestart</wf:parameter>
      <wf:setting
attribute="Setting_Name">com.ibm.bpe.client.commands.wswf.ActivityForceRestart<
/wf:setting>
   </wf:definition>
   <wf:definition>
      <wf:parameter name="WF_Command">ActivityForceComplete</wf:parameter>
      <wf:setting
attribute="Setting_Name">com.ibm.bpe.client.commands.wswf.ActivityForceComplete
</wf:setting>
   </wf:definition>
<wf:commands>
```

Command implementations

Each action command has a command implementation class assigned to it. This class accesses the Process Choreographer API either to receive data or to perform an action. If data is received during the execution of a command implementation, it is stored as a Java object in the page context where it can be accessed by the JSP during the display command execution. A com.ibm.bpe.api.QueryResultSet object is an example of data that is created by the command implementation class and then accessed by a JSP.

Command implementation classes extend the abstract class com.ibm.bpe.client.CommandBase and implement the execute() method declared in the CommandBase. This method is used to implement the actions that should be performed when the corresponding action command is received. Any Process Choreographer API call can be made, such as executing queries, starting/stopping/deleting process, claim activities, etc. For display command implementations, the JSP corresponding to the command is determined and called.

Configuration file

For each command, the configuration.xml file defines the action, the JSP that shows the result of the action, and if cache is used for the command. A configuration.xsd file describes the syntax of this configuration.xml file. In WebSphere Application Server Enterprise, these files are located at <*WebSphere_root*>\installedApps\<*node_name*>\BPEContainer_*node_server*.ear\bpewebclient.war\config (where <*node_name*> and *node_server* are your installation names). In the WebSphere Studio IE they are located at <*WebSphere_Studio_IE_root*>\runtimes\ee_v5\installedApps\localhost\BPEContainer_localhost_server1.ear\bpewebclient.war\config, or if you create an EE Test Environment test server, you can locate these files at project bpewebclient in /Web Content/config.

Settings specific to a business process or a staff activity can overwrite the settings defined in the configuration.xml. You can specify these settings when you design a business process with WebSphere Studio IE:

- For the business process to add specific JSPs, open the business process editor for the process, select the **Client** tab and in the Definition section click **Add**.
- For the staff activity to add specifics JSPs, open the business process editor for the process, select the **Process** tab, right-click the staff activity you want to modify, and select **Properties**. In the Properties Approval window, select **Client** and click **Add**.

If a business process uses complex message types that cannot be displayed with the default JSPs, it is recommended that you write and add specifics JSPs for these business processes.

There are three sections in the configuration.xml file:

- <wf:views> specifies the JSPs that are displayed after each command. If the command is a display, the JSP is directly specified. For action commands, a subsequent command is defined that, in turn, specifies the appropriate JSP.
- <wf:cacheSettings> defines whether the cache is used for a command and if it should be cleared after the command finishes.
- <wf:commands> assigns a command implementation class to each action command.

These sections contains several <definition> elements. Each definition element is compound by <parameter> and <setting> elements.

Default JSPs

The default Web client that comes with Process Choreographer is implemented with a set of JSPs and Java classes. The configuration.xml file of the Web client contains a mapping between these JSPs and Java classes. However, process and staff activity specific JSPs can be specified when a business process is designed. These definitions override the settings in the configuration.xml.

The default JSPs provide the following functionality:

- Activities and process
  - Display attributes of activities and process
  - Enable users to modify attributes of activities and process
  - Display all actions that can be called, for example, claim, complete, save
  - Call these actions

- Simple messages
  - Display message attributes
  - Enable users to modify message attributes
- Worklists
  - Display a list of available worklists, including user-defined worklists
  - Display a worklist (that is, display the result of the worklist query)
  - Display a list of process templates that can be started
- Process templates
  - Show process templates attributes
  - Start a new process template from a process template
- Error
  - Display error messages

The default JSPs are packaged in the bpewebclient.war Web application. In the WebSphere Studio IE, these JSPs are in the bpewebclient project. However, this project is loaded only if you have created an EE Test Environment test server. That is a good place to go if you want to look at the JSP codes. In Table 4-3, you can see a short description of each of the default JSPs.

Table 4-3 Defaults JSPs

JSP	Description
Activity.jsp	Displays the most important activity attributes, requires user inputs and the input and output message of an activity. Action buttons are displayed based on the current user's rights.
ActivityInformation.jsp	Displays detailed information about an activity and the activity's process.
Error.jsp	Shows error message and some context information about the error.
Event.jsp	Shows the details of the event that occurred in the process.
Header.jsp	Is always displayed, shows logos and link to help.
List.jsp	Shows the result of a custom worklist query.

JSP	Description
Navigation.jsp	Like the Header.jsp, it is always displayed. Contains links to the standard worklists and a drop-down list with custom worklists and a button to view them. Depending on the user's rights, a drop-down list with process template names and buttons to view information about the process template, and start a new process.
PageExpired.jsp	Shows a message: `Page has expired`.
ProcessInformation.jsp	Shows detailed information about a process, such as a list of activities that belong to that process and some attribute of those activities.
ProcessInputMessage.jsp	Displays the most important process template attributes. If the current user is authenticated to start this process, the Start Process button is displayed. Provides entry fields for the process input message.
ProcessOutputMessage.jsp	Displays process template information and the process output message.
ProcessTemplates.jsp	Displays detailed process template attributes. If the current user is authenticated to start this process, a Start Process button is displayed.
Terminal.jsp	Terminates the process.
WorklistActivities.jsp	The default start page of the Web client. Shows a list of all activities in state READY, CLAIMED or STOPPED that the current user is allowed to see.
WorklistProcesses.jsp	Shows a list of process. Depending on the context in which it is called and the current user's rights, it shows different lists of processes.

Navigation.jsp and the Header.jsp are always visible, because other JSPs use the <jsp:include page = Header.jsp> and <jsp:include page = Navigation.jsp> to show them. Figure 4-12 on page 71 shows the layout of a displayed page. The information shown in the content pane depends on the JSP that is used to generate the page.

Figure 4-12 Layout of a Web client page

4.3.2 Customizing the choreographer Web client

In this section, we will start using our sample application to show how you can implement the Web client, including how to install and start WebSphere Application Server Enterprise, and install and start your business process application. Start the Web client by typing:

```
http://<Host_Name>/bpe/webclient
```

In a browser the first Web client User interface (after a splash UI that only remains a few seconds in the browser) will appear as shown in Figure 4-12, and you can start playing with your business process application. In Figure 4-13 on page 73 you can see the result of clicking the **Start** button in the Templates pane for CatalogUpdate. In the Process Input Message at the bottom of the window, type the information that the process needs to run, and click the **Start Process** button to run the business process. After it has finished, the returned results will be displayed. More information on this sample application can be found in Chapter 3, “Sample scenario” on page 25. Now the Web client is ready to use, but probably will need to change for many reasons:

- Its look and feel do not match your company standards; for example, see colors, style, banner, and so on in Figure 4-13 on page 73.

- It can display information about the business process itself, such as activity information, template information, process information, etc., but you may need to navigate following a different sequence, suppress some information, group information in different ways, and so on. For an example of how this kind of information is displayed, see Available Actions and Process Template Description in Figure 4-13 on page 73.
- It displays business process-specific information, that is the information that you have passed (typed) to a specific business process or the information that a business process returns (if any). Since it is designed to handle any business process, it uses the name of the property (the part name for the WSDL file) to name the information items in the UI. In the Process Input Message pane in Figure 4-13 on page 73, you can see the results of this approach. For example, in the catalogURL entry field, the comments of this field (string) are probably unintelligible to the end user. This must be changed before sending the application into production. Luckily, Web client can be easily customized to fix this, by implementing user-defined JSP for each business process.
- It displays staff activity-specific information, but it has the same issues of the business process-specific information: the Web client displays this data using part names and data types. We can have more than one staff activity in a business process, and this data will appear in many places. The staff activity is used for approval decisions by the managers, directors or presidents of companies. This must be changed before sending the application into production. Luckily, the Web client can be easily customized to fix this, by implementing user-defined JSPs for each staff activity.
- You may also need to take some of the Web client functions away from human users and put them into an application, such as starting a process passing its initial parameters, or inquiring about the state of an activity, and so on. For that you can use the business Process Choreographer API, the same API used by the Web client.

  Using this API you can, if you have budget and time, write your own Web client from scratch, although it is not recommended.

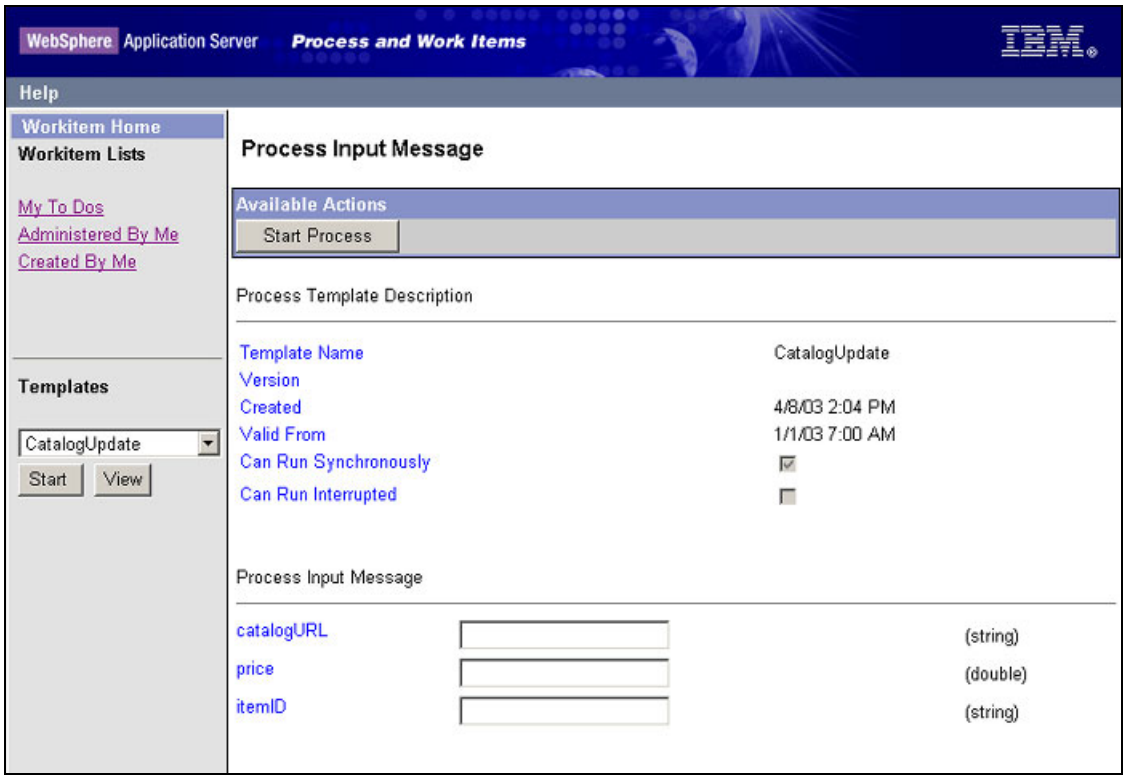


Figure 4-13 Default Web client started template for Catalog Update

Implementing business process-specific JSPs

Web client architecture allows you to enhance your presentation of business process-specific data without any change in the Web client default JSPs. You just have to code a JSP that shows the process-specific information in the way you like it. During the development of the business processes, you must add the JSP definitions where you want them in the business process, and at execution time the business process will pass this information to the Web client, which will call this JSP instead of using its default. We will show step by step how we implemented this modification in the Catalog Update and PO process of our sample application.

To implement user-defined JSPs, we will need a Web application to develop and deploy the JSPs. In theory you can use any existing Web application, but since you have to deploy it in every application server instance that the business process is running, some restrictions will apply. To make maintenance easier, we recommend that you have at least one Web application for each enterprise application that holds business process applications, just for the user-defined

JSPs. Therefore, when you deploy this EAR, everything you need to run your business process is there. We will need to add at least two libraries to this Web project: the bpe.jar and the wsfi.jar.

Restriction: In this version of the product, the Context root of the Web project for the user-defined JSPs must have the same name as the project created in WebSphere Studio IE to hold the JSPs. Otherwise, when you run the application you get the error:

```
[Servlet Error]-[JSP 1.2 Processor]: java.lang.NullPointerException
```

To make sure that the names are the same, in a Web perspective of the WebSphere Studio IE right-click the Web project, select **Properties**. On the Properties window, select **Web** and you will see the Context Root for this project.

Tip: In the Process Choreographer, if you need to have a Web application for the user-defined JSPs, at least in the production environment, with the Context root different from the project name, you will have to change the ABC.fdml file (where ABC is the name of the project). Search it for these strings: `MessageMapping`, `InputMessageJSP`, `OutputMessageJSP` and make the appropriate changes. You can find this file in the Enterprise application in the XYZ.far (where XYZ is the name of your service project). This XYZ.far can be opened with any unzip utility. In WebSphere Studio IE, you can find this file (ABC.fdml) in the Service project. Use a Java perspective to see it, but remember this file will be recreated each time you make a change and save the business process graphic tool. This approach is not recommended because it is very easy to make a mistake.

1. First, we will create the ACompanyProcessWeb Web project and add it to our business process enterprise application ACompanyProcess.

   To create a Web project in WebSphere Studio IE, select **File** from the menu, select **New** -> **Web Project**. On the Create Web Project window, enter the name in the Project name entry field, and click **Next**. In the Enterprise application project, select the **Existing** radio button, browse for the name, and select **ACompanyProcess**.

2. Right-click the project in the Web perspective, then select **Properties**.

3. On the Properties window, select **Web**. See the result in Figure 4-14 on page 75. Note that Context Root and the project name are the same (this is required).

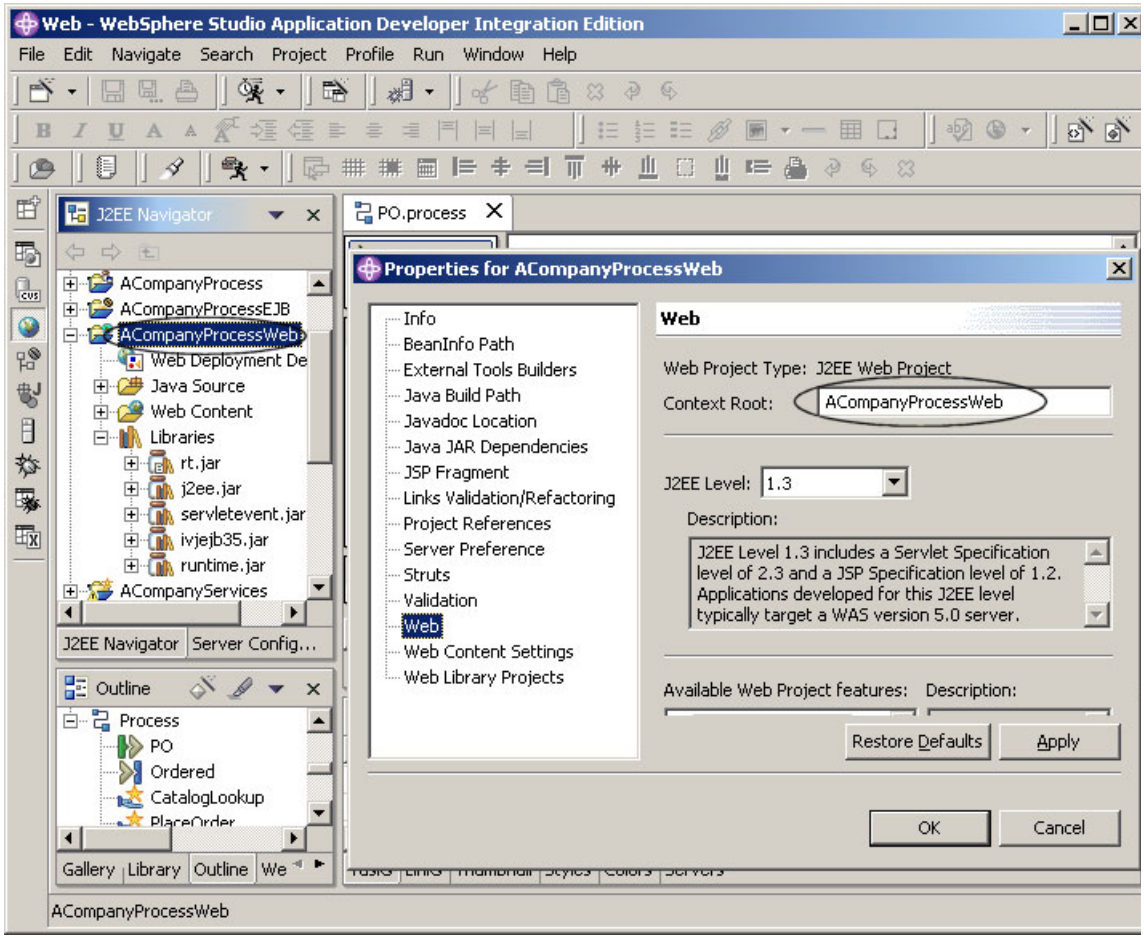


Figure 4-14 Properties of the Web project.

4. Now we will add the two libraries in the Web project for user-defined JSPs. In the Properties window (opened in previous steps), select the **Java Build Path**.
5. In the right part of this window, select the **Libraries** tab. In the Libraries pane, click the **Add Variable** button, and the New Variable Classpath Entry window will open.
6. Scroll down the list until you find the variable named WAS_EE_V5 (see Figure 4-15 on page 76). Select it and click **Extend**.
7. When the Variable Extension window opens, expand the lib folder. Scroll the list and select **bpe.jar**, then click **OK**.

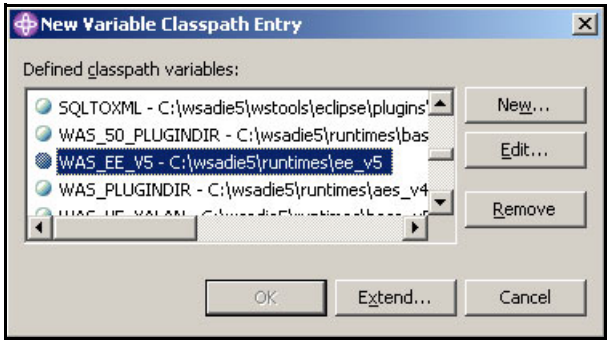


Figure 4-15 New Variable Classpath Entry

8. Repeat steps 1 on page 74 to 7 on page 75 for the wsif.jar. After that, the Web project will have two JARs in its Java Build Pass and Libraries. See Figure 4-16 on page 77.

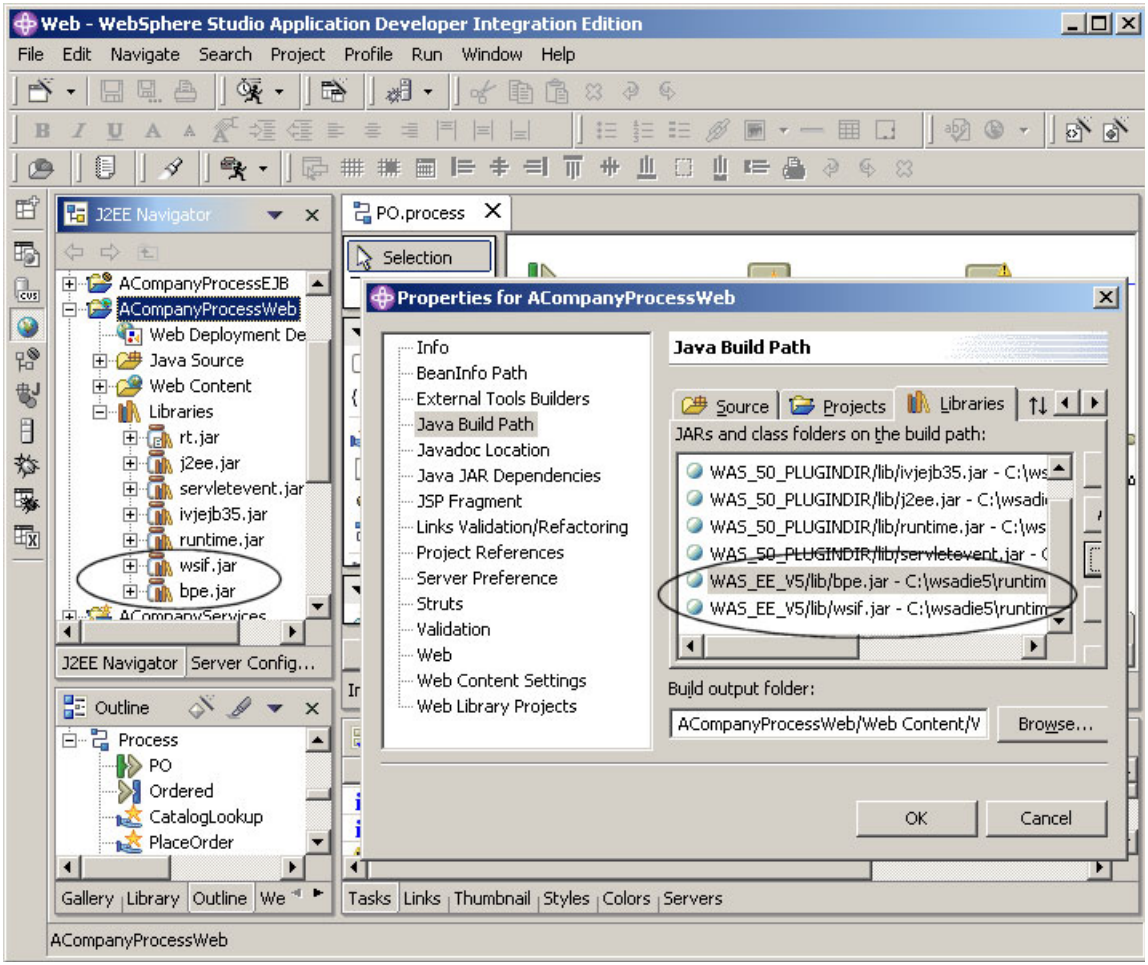


Figure 4-16 Libraries of Web project after we added bpe.jar and wsif.jar

Before we start coding the application JSPs for the business process, let's see where in the Web client they can be defined.

For the business process, see Table 4-4 on page 78.

Table 4-4 In the business process

Render points	User-defined JSP	Observations
ProcessInformation Display	Input Message JSP	Displays process input data information, usually when the process is waiting for some asynchronous activity to complete.
	Output Message JSP	Displays process output data information, return data if any, usually when the process is waiting for some asynchronous activity to complete.
ProcessInput MessageDisplay	Input Message JSP	Displays process input form, so the user can type the process input data needed to run the business process.
	Message Mapping JSP	Gets the data typed in the Input Message JSP, wraps it in a WSIF message, and forwards it to the flow. Optionally, we can validate the typed data before. The implementation is mandatory, if we implement the Input Message JSP.
ProcessOutput MessageDisplay	Output Message JSP	Displays process output data, returned at the end of business process, if any.

For the staff activity, see Table 4-5 on page 79. Remember that if you have more than one staff activity in a business process, you can define these JSPs for each activity if they deal with different data. But if you have lots of staff activity in your business process, it is better to build your data dictionary, which describes your data representations, and build a common JSP. You still need to declare this JSP in all staff activity, but at least the maintenance will be much easier with this approach.

Table 4-5 In staff activity

Render points	User-defined JSP	Observations
Activity Display	Input Message JSP	Displays activity input data information, during all activity states (claimed, pending, and so on.)
	Output Message JSP	Depending on the activity state, it just displays information. If the state is Claimed, it displays an input form so data, such as approve/reject and others needed to complete the activity, can be typed.
	Message Mapping JSP	Gets the data typed in the Output Message JSP, wraps it in a WSIF message and forwards it to the flow. Optionally we can validate the typed data before. The implementation is mandatory if we implement the Output Message JSP.

Overview the Catalog Update JSPs implementation points

Catalog Update process is non-interruptible (microflow), so the only user-defined JSP that is relevant in this case are the processes shown in Table 4-4 on page 78. Microflow does not have any asynchronous activity, so you just need to implement the ProcessInputMessageDisplay and the ProcessOutputMessageDisplay user-defined JSPs. In Figure 4-17 on page 80 in the upper-right pane, you can see the graphic representation of the Catalog Update process. In the upper-left pane, the two circled areas show the input and the output message of this process, which are defined in the CatalogUpdateInterface.wsdl file.

Tip: To open the business process graphic design in WebSphere Studio IE, in a Business Integration perspective, expand the Service Project (ACompanyServices) then expand the package where the business process have been created (com.acompany). Double-click the file **CatalogUpdate.process** to open it and select the **Process** tab.

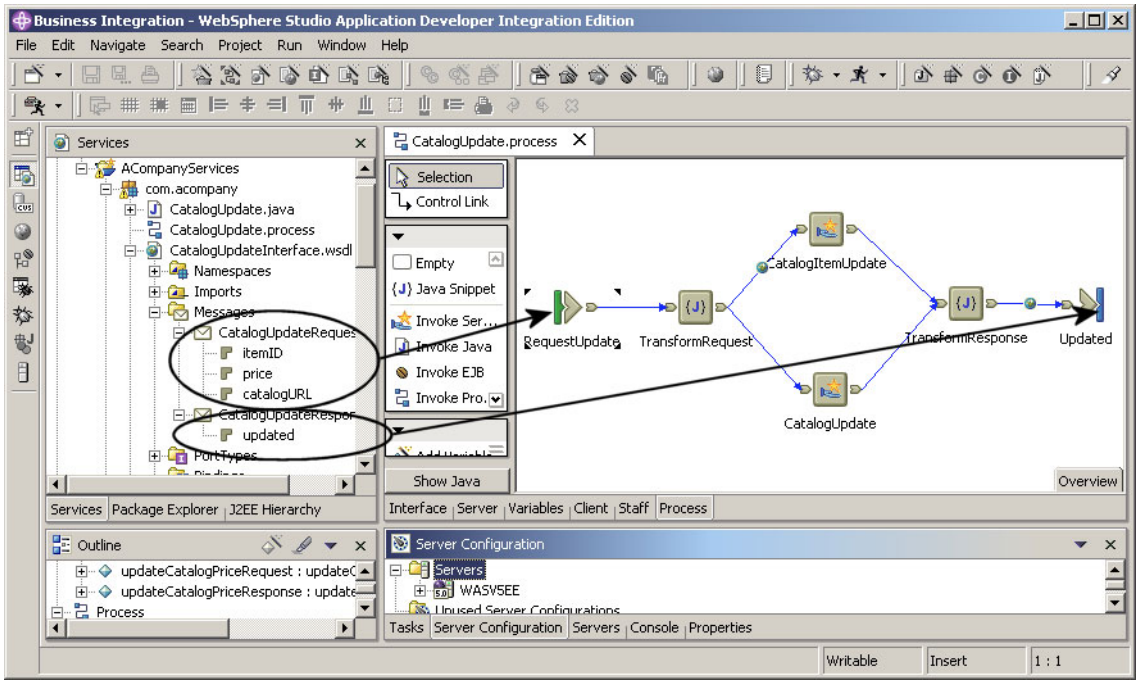


Figure 4-17 Catalog Update business process design

This file defines two messages the CatalogUpdateRequest and the CatalogUpdateResponse.

- The CatalogUpdateRequest message contains three parts, as shown in Table 4-6.

Table 4-6 CatalogUpdateRequest message parts

Part name	Type
itemID	xsd:string
price	xsd:double
catalogURL	xsd:string

Tip: To find the message part types, double-click the file **CatalogUpdateInterface.wsdl** in a Business Integration perspective, select the **Message** tab, select the message and the part, and the type is displayed in a pane below.

The message CatalogUpdateRequest is used to implement the input of the Catalog Update process (named RequestUpdate object in the process graphic design). It maps the data that Catalog Update business process needs to be executed.

Tip: To discover which file is used to map the message and they parts, select the visual object in the process graphic design, right-click **RequestUpdate**, select **Properties**, and in the Properties window select **Implementation**. The File entry field contains the location and name of the file. The Operation entry field information allows you to find the message.

- The CatalogUpdateResponse message contains just one part, as shown in Table 4-7.

Table 4-7 CatalogUpdateResponse message parts

Part name	Type
updated	xsd:int

This message CatalogUpdateResponse is used to implement the output of the Catalog Update process (named update object in the process graphic design). It is the data the business process will return when it finishes.

Based on the information above, the Web client will build the presentation. In Figure 4-18 on page 82, you can see the form generated by the Web client for the process input data.

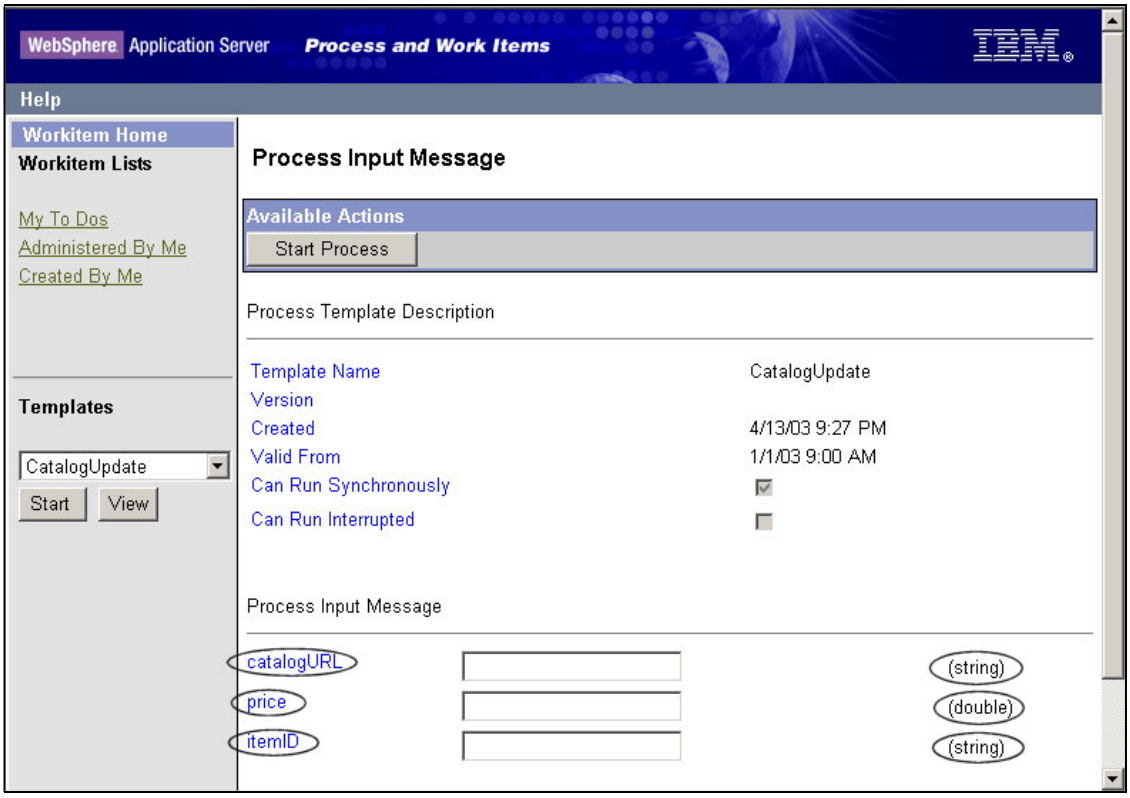


Figure 4-18 Process input message generated by the default Web client

Figure 4-19 on page 83 and Figure 4-20 on page 83 shown the Process Output Message.

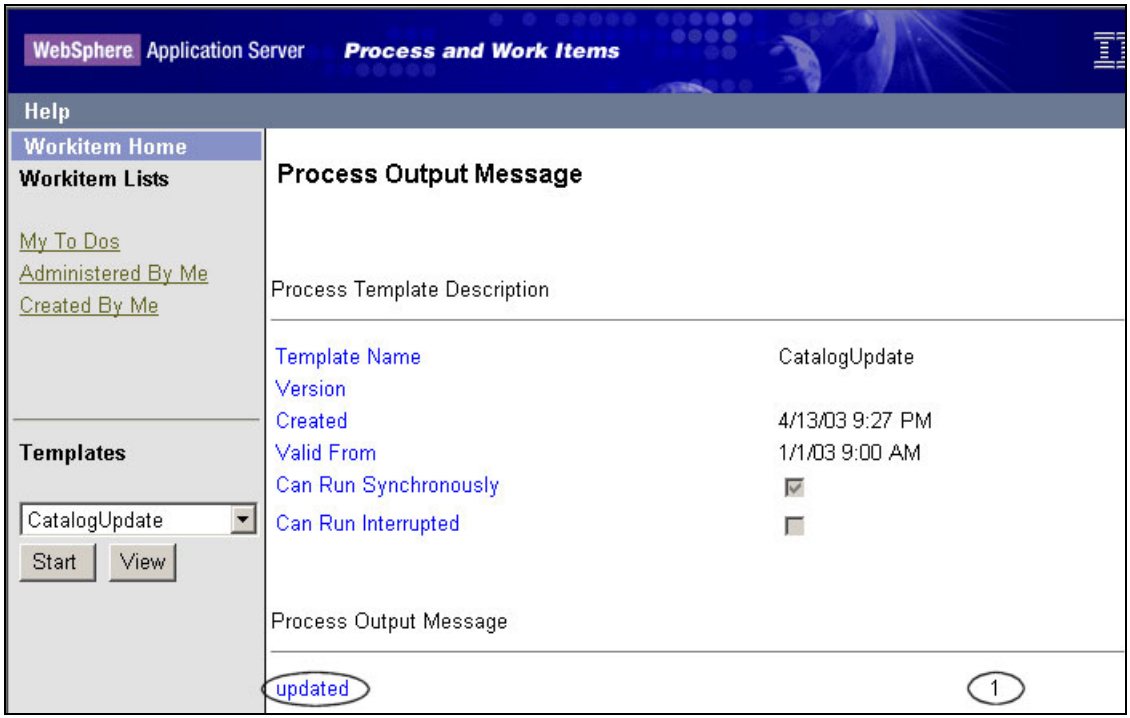


Figure 4-19 Successful process output message generated by the default Web client

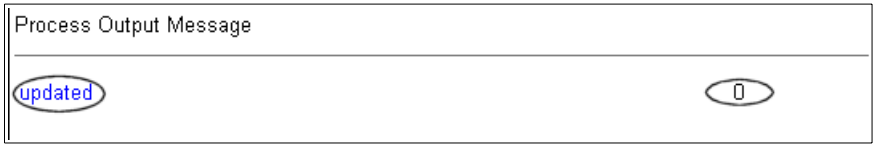


Figure 4-20 Unsuccessful process output message generated by the default Web client

Implementing user-defined JSPs in the Catalog Update

To change the Process Input Message, we must provide to the Web client a new form to use instead of its default rendering. The end user will type data into this form. For every user-defined JSP implementation for which the end user enters data, it is mandatory that we provide a JSP to pack the typed data into a format that the business process engine will understand. We also can use this last JSP to contain the typed data before wrapping it. In this case, we will need a third JSP to show an error message to the end user if the same typed data is not consistent. We will create three JSPs in the Web project ACompanyProcessWeb, already created in “Implementing business process-specific JSPs” on page 73. Their names are shown in Table 4-8 on page 84.

Table 4-8 User-defined JSPs for Catalog Update

JSP Names	Comments
CatalogUpdateProcessInput.jsp	To render the Process Input Message form
CatalogUpdateProcessInputMap.jsp	To contain the user data and wrap it for the business process engine
error.jsp	To display error messages to the end user detected by the CatalogUpdateProcessInputMap.jsp

Tip: To create the JSPs in the WebSphere Studio IE, right-click the **Web Content** folder in the ACompanyProcessWeb, then select **New** and **JSP File**, on the File Name of the New JSP File window enter the JSP name, then click the **Finish** button.

Example 4-3 shows the code of CatalogUpdateProcessInput.jsp. Note that we do not have <html>, <head>, or <body> tags. User-defined JSPs are not self-contained Web pages, but are only fragments of the Web client pages. In addition, JavaScript functions cannot be invoked before a form that contains user-defined JSPs is submitted. Since the <form> tag that encloses the input fields of the user-defined JSP and its Submit button is an element of the standard Web client, you cannot add attributes such as `onSubmit="return validate()"` to the form tag. Remember, you should do the validation in the message mapping JSP, in this case the CatalogUpdateProcessInputMap.jsp. Our implementation is quite simple: just an HTML table with two labels with a correlated input element and a comment. The input element can have any name here, but in the CatalogUpdateProcessInputMap.jsp you have to wrap this elements with the names that the business process knows. See Table 4-7 on page 81 for the part names. The message part catalogURL will be suppressed in the form, so that the end user does not have to deal with it.

Example 4-3 Code of CatalogUpdateProcessInput.jsp

```
<%--**************************************************************************--%>
<%-- This  JSP  will be rendered to get the information needed for at the  --%>
<%-- process start, in this case the "Catalog Update Process". That means, --%>
<%-- every time the Web client need to get the input information for to    --%>
<%-- start this process it calls this JSP instead of using its default     --%>
<%-- render.The data got here must be verified, packed into an             --%>
<%-- ClientObjectWrapper and forward to the flow. This is done in a        --%>
<%-- correspondent message mapping JSP, that in this case is the           --%>
<%-- "CatalogUpdateProcessInputMap.jsp", where we will use the input (like --%>
<%-- names "itemID_Input") to get the typed value.                         --%>
```

```
<%--************************************************************************--%>
<table>
   <colgroup>
      <col width="20%" span="2">
      <col width="60%" span="1">
   </colgroup>
   <tr class="marked1">
      <td class="fw_label">Part number</td>
      <td> <input type="text" name="itemID_Input"> </td>
      <td>Enter the part number you want to change the price.</td>
   </tr>
   <tr class="marked2">
      <td class="fw_label">New price</td>
      <td> <input type="text" name="price_Input"></td>
      <td>Enter the new price you want to the part number.</td>
   </tr>
</table>
```

Example 4-4 shows our CatalogUpdateProcessInputMap.jsp implementation code. In the page directive, we declare the Error.jsp as the error page for this JSP. Therefore, every time we raise ServletException (three times in the code) the Error.jsp will be invoked to show the error message, and guide the user in retrying the operation. Also, we must declare the Java packages that we need to import. The following are some important details about the code:

1. `String itemID_Input = request.getParameter("itemID_Input");`

   The name "itemID_Input" is the same used to name the input element in the CatalogUpdateProcessInput.jsp.

2. `message.setObjectPart("itemID", itemID_Input);`

   This line builds the input message (add the first part value).The `"itemID"` is the name of the part that is defined in the input of the business process. See Table 4-7 on page 81 and/or the upper-left pane of Figure 4-17 on page 80 for the name of the input message parts.

3. `message.setObjectPart("catalogURL", "");`

   This part will not be used now. As we said before, we will remove it from the presentation, but it needs to be added to the message, even with no value. Otherwise we will get a nullpointException.

Example 4-4 CatalogUpdateProcessInputMap.jsp

```
<%--************************************************************************--%>
<%-- This is the map JSP for the "CatalogUpdateProcessInput.jsp" so here we--%>
<%-- parse the typed data if it is invalid throws a ServletException with a--%>
<%-- description of the error. In this case the "error.jsp" will be called,--%>
<%-- because we have it in the page directive of this JSP, to show the error-%>
```

```
<%-- message and how to retry the operation. When all data is correct, we --%>
<%-- wraps it in ClientObjectWrapper and forward to the flow.             --%>
<%--************************************************************************--%
>
<%@ page
   language="java"
   contentType="text/html;charset=UTF-8"
   errorPage="./Error.jsp"
   import="java.util.*,
          com.ibm.bpe.api.ClientObjectWrapper,
          com.ibm.bpe.client.*,
          org.apache.wsif.base.WSIFDefaultMessage"%>
<%
// Create an instance of WSFI message class
WSIFDefaultMessage message = new WSIFDefaultMessage();

// check part number input
String itemID_Input = request.getParameter("itemID_Input");
if (itemID_Input.equals("")) {
   throw new ServletException("Please enter a Part number.");
}
//add the itemID data to the WSFI message
message.setObjectPart("itemID", itemID_Input);

// check new price input
String price_Input = request.getParameter("price_Input");
if (price_Input.equals("")) {
   throw new ServletException("Please enter a New price.");
}
//check new price for a valid data, and if ok add to the WSFI message
try {
   message.setDoublePart("price",
(Double.valueOf(price_Input).doubleValue()));
} catch (Exception ex) {
   throw new ServletException("The New price '"+price_Input+"' is not valid.
");
}

// set catalogURL input to nothing,business process need it in the message
// (as it is one of input part) even if we will not use it for this operation.
message.setObjectPart("catalogURL", "");

//Wrap the WSFI message in the business process message wrap class
ClientObjectWrapper messageObject = new ClientObjectWrapper(message);

//Create an vector and add the request parameters to be excluded
Vector excludeRequestParameters = new Vector();
excludeRequestParameters.add("itemID_Input");
excludeRequestParameters.add("price_Input");
```

```
//forward message with the user typed data to the flow
MessageUtilities.forwardMessageToController(request, response, messageObject,
null, excludeRequestParameters);
%>
```

Example 4-5 is a simple code for the Error.jsp. It just displays the error message from the exception attribute. Note that it is a complete Web page; it includes the necessary <html>, <body>, and so on... tags.

Example 4-5 Error.jsp

```
<%--************************************************************************--%>
<%-- This JSP will show a error message in case of some exception in the -%>
<%-- Map JSP, CatalogUpdateProcessInputMap.jsp and the PoProcessInputMap.jsp-%>
<%-- so when we raise Servlet exceptions because of typing errors this JSP --%>
<%-- will show the message that we passed and instruct the user how do retry-%>
<%--************************************************************************--%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Error</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<style type="text/css">
p,a,font {
   font-family:Verdana, Arial, Helvetica, sans-serif;
   font-size: 10pt;
   }
</style>

<%@ page import="java.io.*"
   isErrorPage="true"
   contentType="text/html;charset=UTF-8"
%>

<body>
<p>An error occured while processing data:</p>

<font color="red"><B> <% out.println(exception.getMessage());%> </B></font><br>
<br>
<H4>Please use the 'back' button of your browser to return to the previous page
and complete / correct your input data. </H4>

</body>
</html>
```

The only thing missing now for the Catalog Update user-defined JSP to work is to add them to the business process. To do that:

1. In the WebSphere Studio IE, switch to the Business Integration perspective, then expand the **ACompanyServices** project.
2. Double-click the **CatalogUpdate.process** file under the com.acompany package.
3. Select the **Client** tab.

At the end of these steps you should see something like Figure 4-21.

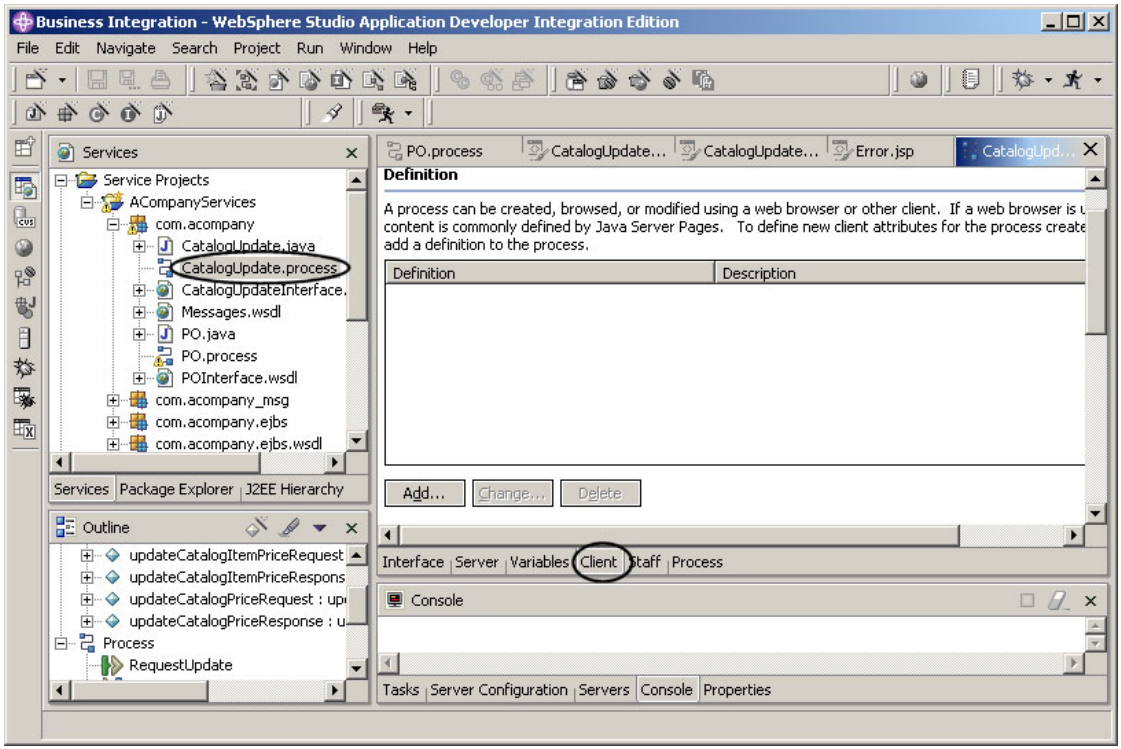


Figure 4-21 Catalog Update process client page

To add the CatalogUpdateProcessInput.jsp:

1. On the Client tab, click **Add**.
2. In the Add definition window, select **ProcessInputMessageDisplay** from the Action list drop-down list.
3. In the JSP Page Settings, select the **Value** cell in front of Input Message JSP.
4. Click the button with three dots next to the cell.

5. On the File Selection window, expand the **ACompanyProcessWeb**. Expand the **Web Content** folder, select the **CatalogUpdateProcessInput.jsp** and click the **OK** button.

To add the CatalogUpdateProcessInputMap.jsp:

1. In the JSP Page Settings, click **Message Mapping JSP**.
2. Click the button with three dots next to the cell.
3. On the File Selection window, expand the **ACompanyProcessWeb**, then expand the **Web Content**, select the **CatalogUpdateProcessInputMap.jsp** and click the **OK** button.

   At the end it should look like Figure 4-22.

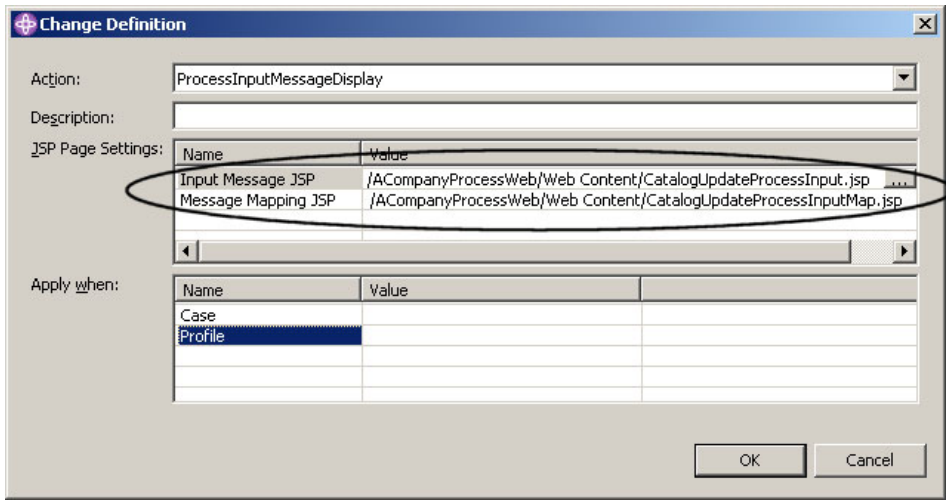


Figure 4-22 Catalog Update ProcessInputMessageDisplay configuration

4. To finish, click **OK**, then save the changes. Figure 4-23 on page 90 shows the results of the implementation when we run the Web client with the user-defined JSP for the process input message. Compare this with Figure 4-18 on page 82, which shows the same Web client UI, but without the user-defined JSP.

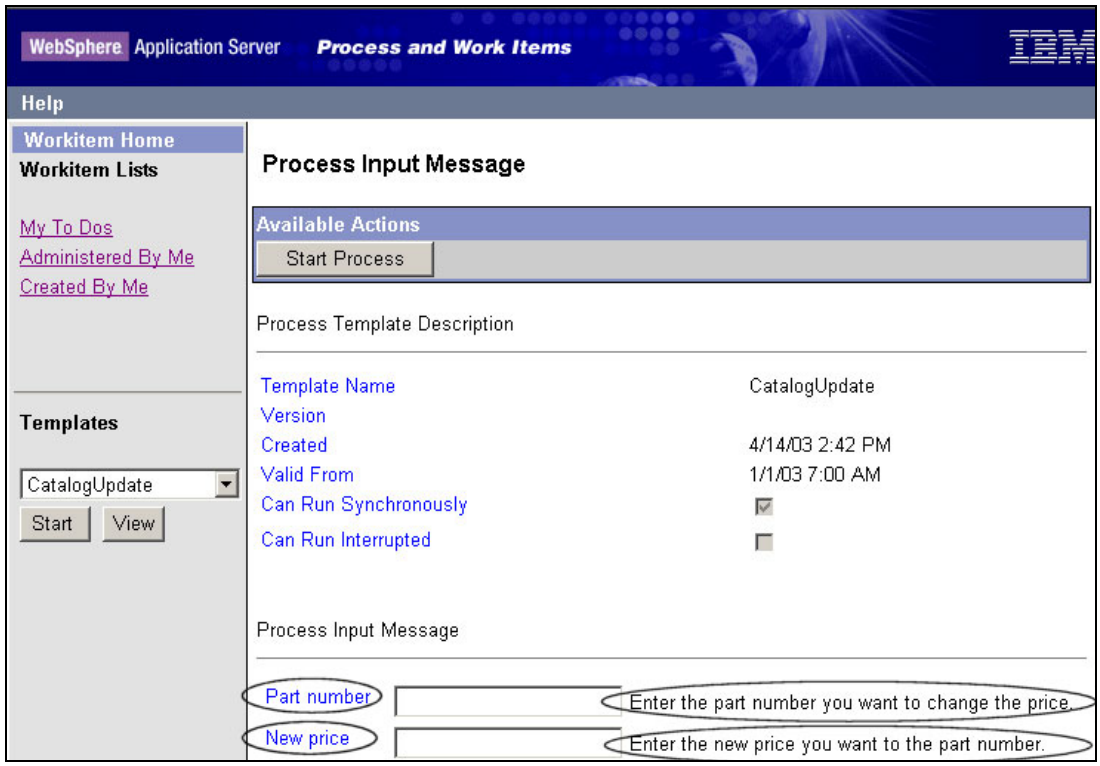


Figure 4-23 Catalog Update Process Input Message with user-defined JSP

The next step is to implement the Catalog Update Process Output Message, to provide to the Web client the new user-defined output JSP that it is going to use instead of its default rendering. This JSP only displays data, so we do not need the mapping. First let's create a new JSP file named CatalogUpdateProcessOutput.jsp in the Web Content folder of the ACompanyProcessWeb project.

Example 4-6 shows the sample code for the CatalogUpdateProcessOutput.jsp. As you can see it is very simple. We receive the output message in the request, get the value of the output part, in this case updated (remember the name of the message parts are defined in the output of the business process, as shown in Table 4-8 on page 84 and the upper-left upper pane of Figure 4-17 on page 80 for the name of the message parts) and display information depending on its value.

Example 4-6 CatalogUpdateProcessOutput.jsp

```
<%--**************************************************************************--%>
<%-- This is the JSP that will be rendered, at the end of Catalog Update    --%>
```

```
<%-- Process to show the returned results if any. That means every time the --%>
<%-- Web client need to show this information for this process it will      --%>
<%-- call this JSP instead of using its default render                      --%>
<%--************************************************************************--%>
<%@ page
   language="java"
   contentType="text/html;charset=UTF-8"
   import="com.ibm.bpe.api.*,
          com.ibm.bpe.client.*,
          org.apache.wsif.base.WSIFDefaultMessage"%>

<%--Getting a reference to the output message  --%>
<jsp:useBean id="outputMessage" class="java.lang.Object" scope="request"> </jsp:useBean>
<%
//Casting to the business process wrap class
ClientObjectWrapper cMsg = (ClientObjectWrapper)outputMessage;

//Extract the WSFI message from the wrap class
WSIFDefaultMessage outMsg = (WSIFDefaultMessage)cMsg.getObject();

//Testing the updated attribute to show the right information
if ((outMsg.getIntPart("updated")) == 1){%>
<H3 align="center">The part number price was successfully updated.</H3>
<%}
else {%>
<H3 align="center">The part number price was not updated.</H3>
<%}%>
```

Now you need to add it to the Catalog Update process, as we did for the Input message. Open the CatalogUpdate.process in a Business Integration perspective, and select the Client tab.

1. In the Client tab, click **Add...**
2. In the Add definition window, select the **ProcessOutputMessageDisplay** from the Action drop-down list.
3. In the JSP Page Settings, click **Output Message JSP**.
4. Click the button with three dots next to the cell.
5. On the File Selection window, expand **ACompanyProcessWeb**, expand **Web Content**, select **CatalogUpdateProcessOutput.jsp**, then click **OK**.
6. Click **OK** then save the changes.

Figure 4-23 on page 90 shows the results of the implementation, Compare this with Figure 4-18 on page 82, which is the same UI without the user-defined JSP.

We can see the result of this implementation when we run the Web client with the process output user-defined JSP. Figure 4-24 shows a successful result and Figure 4-25 an unsuccessful result. Compare with the Web client default renderings in Figure 4-19 on page 83 and Figure 4-20 on page 83.

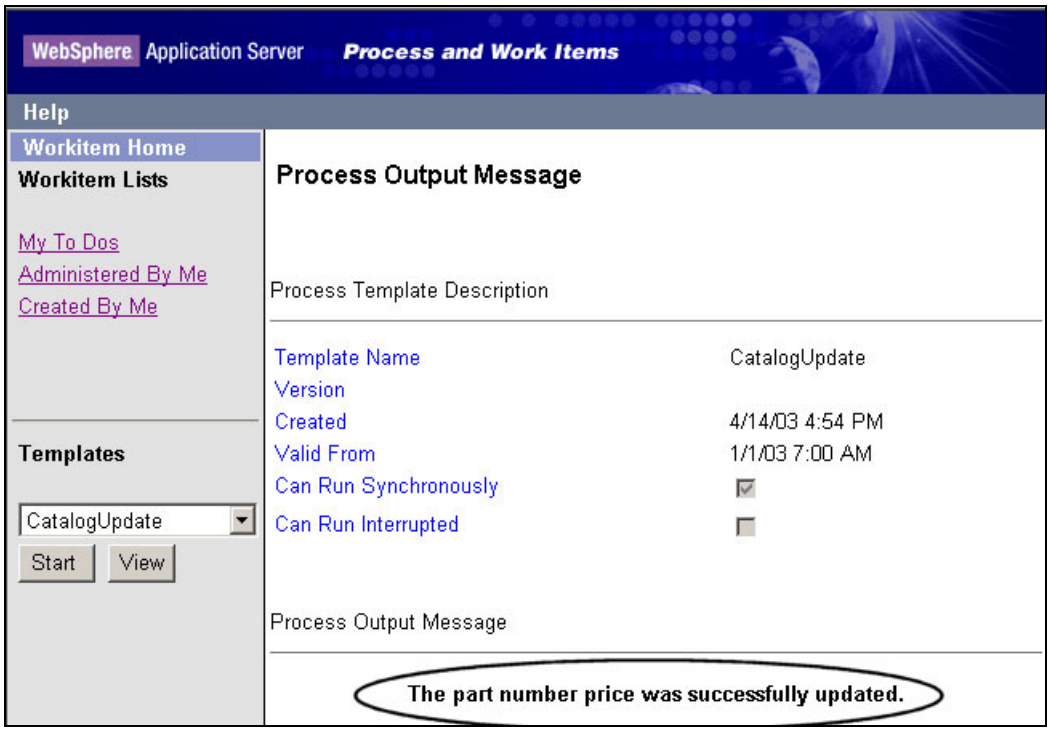


Figure 4-24 Successful process output message generated by the Web client with defined JSP

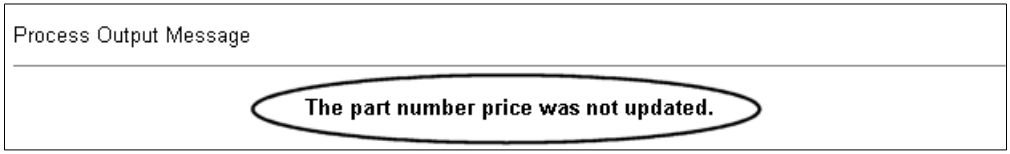


Figure 4-25 Unsuccessful process output message generated by the Web client with defined JSP

Overview of the PO JSPs implementation points

The PO business process is interruptible (macroflow). Since it has a staff activity both the business process (see Table 4-4 on page 78) and the staff activity (see Table 4-5 on page 79) user-defined JSPs are relevant for this business process. But as this business process does not return any result, the ProcessOutputMessageDisplay and the ProcessInformationDisplay - Output Message JSP user-defined JSPs are not relevant (there will not be any message

to show). In the upper-right pane of Figure 4-26, we can see the graphic representation of PO business process. In the upper-left upper pane, starting from the top down, the first and second circled areas show the input and output message parts of the Approval staff activity. These definitions are in the Messages.wsdl file. The third circled area shows the input message parts of the PO business process. These definitions are in the POInterface.wsdl file.

Tip: To open the business process graphic design in WebSphere Studio IE, in a Business Integration perspective expand **Service Project (ACompanyServices)** then expand the package where the business process has been created (**com.acompany**), then double-click the file **PO.process** and select the **Process** tab.

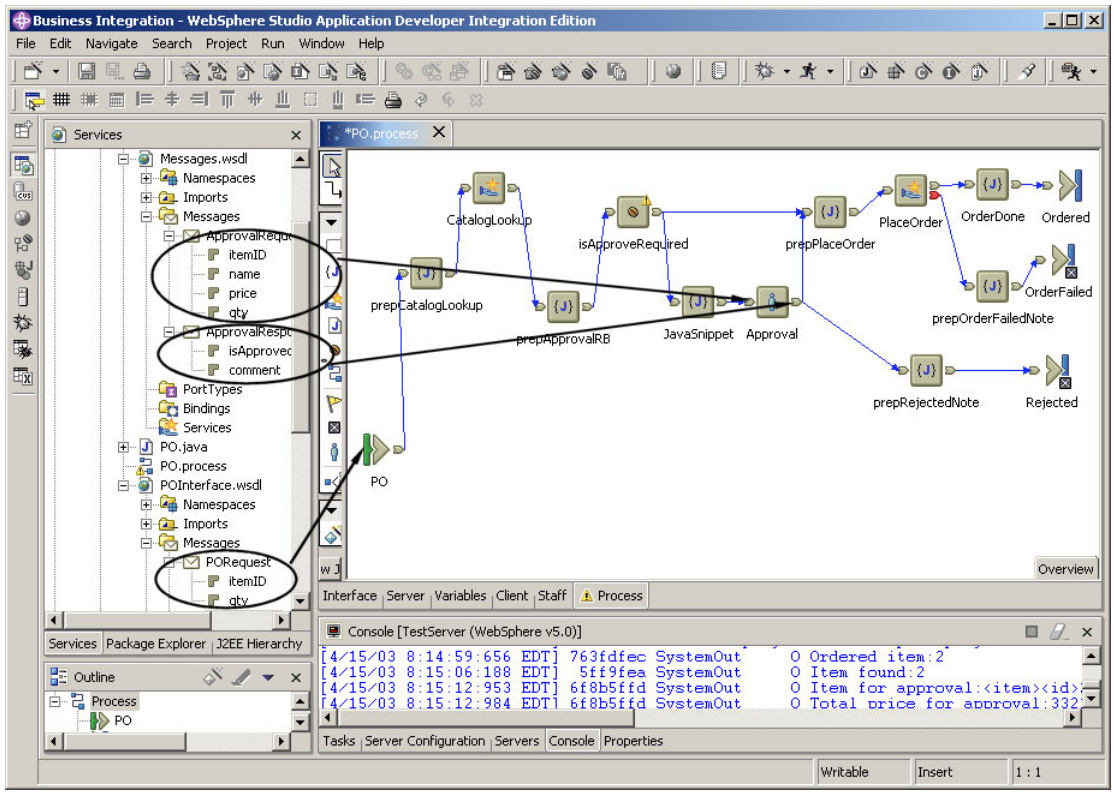


Figure 4-26 PO business process graphic design

The POInterface.wsdl file defines the PORequest message that contains two parts shown in Table 4-9 on page 94.

Table 4-9 PORequest message parts

Part name	Type
itemID	xsd:string
qty	xsd:int

To find the message part types, double-click the file **POInterface.wsdl** in a Business Integration perspective, select the **Message** tab, and select the message and the part. The type is displayed in a pane below.

The PORequest message is used to implement the Input of the PO business process (named PO object in the process graphic design). It maps the data that PO business process needs to be executed.

Tip: To discover which file is used to map the message and the parts, select the visual object in the process graphic design, right-click **PO** in this case, select **Properties**, on the Properties window select **Implementation**. The File entry field contains the location and name of the file. The Operation entry field information allows you to find the message.

The Messages.wsdl file defines two messages: the ApprovalRequest and the ApprovalResponse.

The ApprovalRequest contains four parts, as listed in Table 4-10.

Table 4-10 ApprovalRequest message parts

Part name	Type
itemID	xsd:string
name	xsd:string
price	xsd:double
qty	xsd:int

This message ApprovalRequest is used to implement the input of the Approval staff activity of the PO business process (named Approval object in the process graphic design). It is the data the Approval staff activity will receive from the flow.

The ApprovalResponse contains two parts, as listed in Table 4-11 on page 95.

Table 4-11 ApprovalResponse message parts

Part name	Type
isApproved	xsd:boolean
comment	xsd:string

The ApprovalResponse message is used to implement the output of the Approval staff activity of the PO business process (named Approval object in the process graphic design). It is the data the Approval staff activity will return to the flow when it is completed.

Based on the information above, the Web client builds the presentation. Instead of showing how the default Web client presentations looks now, we will first see the implementation of the user-defined JSPs for the PO business process. Then we show all the before and after presentations in sequence, so it will be easier for you to see the improvements in the implementation.

Implementing user defines JSPs - Process page

For the process user-defined JSPs, we will implement the ProcessInputMessageDisplay and the ProcessInformationDisplay -Input Message JSP. Once the PO business process does not return any information, there are no message for it to display for the ProcessOutputMessageDisplay or ProcessInformationDisplay - Output Message JSP. The ProcessInputMessageDisplay gets a data JSP; that means it renders a form so the user can type the input data needed by the PO business process. Therefore, we will need a correlated mapping JSP to validate the typed data and build a WSIF message with this data. If a user types data that is not valid and we detect it at the mapping JSP, we will use the Error.jsp that we created before for the Catalog Update, to show the error information. We will create the JSP files shown in Table 4-12 in the ACompanyProcessWeb.

Table 4-12 Process user-defined JSPs for PO

JSP Names	Comments
PoProcessInput.jsp	To render the Process Input Message form
PoProcessInputMap.jsp	To consist the user data and wrap it for the business process engine
PoProcessInfoInput.jsp	To display the Process Input Message

Example 4-7 on page 96 shows the implementation of the Process Input Message JSP. It is a very simple HTML table with the fields we want the user to

enter as process input data. For more details and some restrictions, see this implementation for the Catalog Update in Example 4-3 on page 84.

Example 4-7 PO Process Input user-defined JSP (PoProcessInput.jsp)

```
<%--*************************************************************************--%>
<%-- This  JSP  will be rendered to get the information needed for at the  --%>
<%-- process start, in this case the "PO Process". That means, every       --%>
<%-- time the Web client need to get the input information for to start    --%>
<%-- this process it calls this JSP instead of using its default render.The--%>
<%-- data got here must be verified, packed into an ClientObjectWrapper and--%>
<%-- forward to the flow. This is done in a correspondent message mapping  --%>
<%-- JSP, that in this case is the  "PoProcessInputMap.jsp",where          --%>
<%-- we will use the input (like names "itemID_Input")to get the typed value-%>
<%--*************************************************************************--%>
<table>
   <colgroup>
      <col width="20%" span="2">
      <col width="60%" span="1">
   </colgroup>
   <tr class="marked1">
      <td class="fw_label">Part number</td>
      <td> <input type="text" name="itemID_Input"> </td>
      <td>Enter the part number you want to order.</td>
   </tr>
   <tr class="marked2">
      <td class="fw_label">Quantity</td>
      <td> <input type="text" name="qty_Input"></td>
      <td>Enter the quantity you want to order.</td>
   </tr>
</table>
```

Example 4-8 shows our POProcessInputMap.jsp implementation code. Remember that the Error.jsp in the page directive is the same created for Catalog Update. This JSP will validate the information from the PoProcessInput.jsp. It will pack the information using the part names defined in the business process (see Table 4-9 on page 94 or Figure 4-26 on page 93) on a WSIF message, and forward it to the flow. For more details, see this implementation for the Catalog Update in Example 4-4 on page 85.

Example 4-8 PO Process Input mapping user-defined JSP (PoProcessInputMap.jsp)

```
<%--*************************************************************************--%>
<%-- This is the map JSP for the "PoProcessInput.jsp" so here we           --%>
<%-- parse the typed data if it is invalid throws a ServletException with a--%>
<%-- description of the error. In this case the "error.jsp" will be called,--%>
<%-- because we have it in the page directive of this JSP, to show the error-%>
<%-- message and how to retry the operation. When all data is correct, we  --%>
<%-- wraps it in ClientObjectWrapper and forward to the flow. --%>
```

```
<%--**************************************************************************--%>
<%@ page
   language="java"
   contentType="text/html;charset=UTF-8"
   errorPage="./Error.jsp"
   import="java.util.*,
          com.ibm.bpe.api.ClientObjectWrapper,
          com.ibm.bpe.client.*,
          org.apache.wsif.base.WSIFDefaultMessage"%>
<%
// Create an instance of WSFI message class  and wrap it in
// the ClientObjectWrapper
WSIFDefaultMessage msg = new WSIFDefaultMessage();
ClientObjectWrapper wrapMsg = new ClientObjectWrapper(msg);

// check part number input
String itemID_Input = request.getParameter("itemID_Input");
if (itemID_Input.equals("")) {
   throw new ServletException("Please entrer a part number.");
}
msg.setObjectPart("itemID", itemID_Input);

// check quantity input
String qty_Input = request.getParameter("qty_Input");
if (qty_Input.equals("")) {
   throw new ServletException("Please enter a quantity.");

//check qty for a valid data, and if ok add to the WSFI message
}
try {
   msg.setIntPart("qty", (Integer.valueOf(qty_Input).intValue()));
} catch (Exception ex) {
   throw new ServletException("The quantity '"+qty_Input+"' is not valid. ");
}

//Wrap the WSFI message in the business process message wrap class

//Create an vector and add the request parameters to be excluded
Vector excludeRequestParameters = new Vector();
excludeRequestParameters.add("itemID_Input");
excludeRequestParameters.add("qty_Input");

//forward message with the user typed data to the flow.
MessageUtilities.forwardMessageToController(request, response, wrapMsg, null,
excludeRequestParameters);
%>
```

The Web client will call the ProcessInformationDisplay - Input Message user-defined JSP every time it needs to display (read only) the value of the input parameters for a specific instance of the PO business process. You can see in Example 4-9 the implementation code for this JSP, We do not receive the input message in the request; instead we receive the process ID (PIDD) that identifies this specific instance of the PO business process. We use the PIDD to get the input message for this instance of the business process. For the name of the parts of this message, see Table 4-9 on page 94.

Example 4-9 PO Process Input information user-defined JSP (PoProcessInfoInput.jsp)

```
<%--************************************************************************--%>
<%-- This is the JSP that will render the input information of the process --%>
<%-- "PO Process" just for show (read-only). That means ever time the Web  --%>
<%-- Client need to show this information for this process it calls this JSP-%>
<%--  instead of using its default render.                                  --%>
<%--************************************************************************--%>
<%@ page
   language="java"
   contentType="text/html;charset=UTF-8"
   import="com.ibm.bpe.api.*,
          com.ibm.bpe.client.*,
          org.apache.wsif.base.WSIFDefaultMessage"%>
<%
//get the processor ID string
String pidd = request.getParameter(Constants.WF_PIID);

//get the local reference of the business process EJB
LocalBusinessProcess process = MessageUtilities.getProcess(request);

//get the business process wrapped message
ClientObjectWrapper cMsg = (ClientObjectWrapper)process.getInputMessage(pidd);

//Extract the WSFI message from the wrap class
WSIFDefaultMessage inMsg = (WSIFDefaultMessage)cMsg.getObject();

//as you can see in the HTML below the dynamic data is obtained from
//the input message - inMsg.getXxxxx("xxxxx")-
%>
<table width="100%">
   <colgroup>
      <col width="20%" span="2">
      <col width="60%" span="1">
   </colgroup>
   <tr class="marked2">
      <td class="fw_label" height="25pt">Part number</td>
      <td><%= inMsg.getObjectPart("itemID")%></td>
      <td>The part number of the ordered item.</td>
   </tr>
```

```
   <tr class="marked1">
      <td class="fw_label" height="25pt">Quantity</td>
      <td><%= inMsg.getIntPart("qty")%></td>
      <td>The number of ordered items.</td>
   </tr>
</table>
```

To add this JSP to the PO business process, do as follows:

1. Open the **PO.process** in a Business Integration perspective and select the **Client** tab.
2. To add the PoProcessInput.jsp, on the Client tab, click **Add**.
3. In the Add Definition window, select **ProcessInputMessageDisplay** from the Action drop-down list.
4. From the JSP Page Settings, select **Input Message JSP**.
5. Click the button with three dots next to the cell.
6. On the File Selection window, expand **ACompanyProcessWeb**, expand **Web Content**, select **PoProcessInput.jsp** and click **OK**.
7. Add the PoProcessInputMap.jsp. In the JSP Page Settings, select **Message Mapping JSP**.
8. Click the button with three dots next to the cell.
9. On the File Selection window, expand **ACompanyProcessWeb**, expand **Web Content**, select **POProcessInputMap.jsp** and click **OK**.
10. To add the PoProcessInfoInput.jsp, on the Client tab pane click **Add**.
11. In the Add Definition window, select **ProcessInformationDisplay** from the Action drop-down list.
12. In the JSP Page Settings, select **Input Message JSP**.
13. Click the button with three dots next to the cell.
14. On the File Selection window, expand **ACompanyProcessWeb**, expand **Web Content**, select **PoProcessInfoInput.jsp** and click **OK**.
15. At the end, click **OK** and save the changes.

Implementing user defines JSPs - Activity page

To implement the user-defined JSPs for the Approval staff activity, we need to create three new JSP files in the project ACompanyProcessWeb, as listed in Table 4-13 on page 100.

Table 4-13 Process user-defined JSPs for PO

JSP Names	Comments
PoActivityInput.jsp	To display activity Input Message.
PoActivityOutput.jsp	Depending on the activity state, it displays information. If the state is Claimed, it displays an input form so data such as approve/reject, etc., needed to complete the activity can be entered.
PoActivityOutputMap.jsp	To contain the user data and wrap it for the business process engine

The Web client will invoke the staff activity input JSP every time it needs to display (read only) the data that this activity received when it started. This information is very important for the owner of that activity. Based on this data, the process owner will take action before completing the activity (in our case approve or reject an order). Example 4-10 shows the implementation code of this user-defined JSP. It is very simple. The input message is received in the request, so we just need to get the data we want to show using the input message part names. See Table 4-10 on page 94. To help the user make a decision, we also add in the presentation a new entry that shows the total price. In the input message, we have the unit price and the number of items. A snippet of this code is as follows:

```
<td class="fw_label" height="25pt">Total price</td>
<td><%= ((inMsg.getDoublePart("price"))*(inMsg.getIntPart("qty")))%></td>
<td>The total cost in US$.</td>
```

We can add extra fields or remove fields, as we did in the Catalog Update business process in Example 4-4 on page 85.

Example 4-10 PO Approval staff activity Input user-defined JSP (PoActivityInput.jsp)

```
<%--************************************************************************--%>
<%-- This is the JSP that will render the input information of the         --%>
<%-- "Approval" activity of the process "PO Process". That means every time--%>
<%-- the Web client need to show the information for this activity/process --%>
<%-- it calls this JSP instead of using its default render.                --%>
<%--************************************************************************--%>
<%@ page
   language="java"
   contentType="text/html;charset=UTF-8"
   import="com.ibm.bpe.api.*,
          com.ibm.bpe.client.*,
          org.apache.wsif.base.WSIFDefaultMessage"%>
```

```
<jsp:useBean id="inputMessage" class="java.lang.Object" scope="request">
</jsp:useBean>

<%
//Casting to the business process wrap class
ClientObjectWrapper cMsg = (ClientObjectWrapper)inputMessage;

//Extract the WSFI message from the wrap class
WSIFDefaultMessage inMsg = (WSIFDefaultMessage)cMsg.getObject();

//as you can see in the HTML below the dynamic data is obtained from
//the input message - inMsg.getXxxxx("xxxxx")-
%>
<table width="100%">
   <colgroup>
      <col width="20%" span="2">
      <col width="60%" span="1">
   </colgroup>
   <tr class="marked1">
      <td class="fw_label" height="25pt">Name</td>
      <td><%= inMsg.getObjectPart("name")%></td>
      <td>The name of the order' request.</td>
   </tr>
   <tr class="marked2">
      <td class="fw_label" height="25pt">Part number</td>
      <td><%= inMsg.getObjectPart("itemID")%></td>
      <td>The part number of the ordered item.</td>
   </tr>
   <tr class="marked1">
      <td class="fw_label" height="25pt">Unit Price</td>
      <td><%= inMsg.getDoublePart("price")%></td>
      <td>The part number price in US$.</td>
   </tr>
   <tr class="marked1">
      <td class="fw_label" height="25pt">Quantity</td>
      <td><%= inMsg.getIntPart("qty")%></td>
      <td>The number of ordered itens.</td>
   </tr>
   <tr class="marked2">
      <td class="fw_label" height="25pt">Total price</td>
      <td><%=  ((inMsg.getDoublePart("price"))*(inMsg.getIntPart("qty"))
)%></td>
      <td>The total cost in US$.</td>
   </tr>
</table>
```

Now let's see the most complex implementation. The PoActivityOutput.jsp is a user-defined JSP. This JSP builds different presentations based on the activity states, which can be:

- **Ready**: The activity was just created, and is waiting for some authorized user to claim it.
- **Claimed**: Some authorized user has already claimed it, but has not completed it.
- **Finished**: The activity has been completed.

When the activity is in the Claimed state, it can have saved data or not. The owner of the activity needs to leave the activity after entering data but before completing it, he has the option to save the entered data by clicking the Save button, and when he returns to this activity he will receive the form with the data he had entered before. The implementation code in Example 4-11 shows how to deal with the details. From the request we receive the activity ID (AIID), which represent a specific instance of that activity. We use that to obtain the output message to get saved data if any. See Table 4-11 on page 95 for the message part names. Other objects that we need to receive in the request include the ActivityInstanceData, which we use to check the state of the activity. The following is an important line of this code:

```
if(activity.getExecutionState() == ActivityInstanceData.STATE_READY)
```

This line shows how you test the activity state (Ready in this line). Based on the result you build the presentation. In this case, it just indicates that this order has to be approved.

```
if (outMsg != null) {
   commentInputValue = (String)outMsg.getObjectPart("comment");
   approvalInputValue = outMsg.getBooleanPart("isApproved");}%>
   ...
```

This code runs when the activity is in a Claimed state. Here we test for saved data (outMsg is not null), so if we have previous saved data, we load it in some variables to show that in the form. These variables were loaded before with some defaults. As you can see in the source code above, in this case we do not have any saved data (outMsg is null).

Example 4-11 PO Approval staff activity Output user-defined JSP (PoActivityOutput.jsp)

```
<%--**************************************************************************--%>
<%-- This JSP will render output information of "Approval" activity of PO  --%>
<%-- Process depending on the state of the activity. That means, every time--%>
<%-- the Web client needs to get or show  output information for this      --%>
<%-- activity/process it calls this JSP instead of using its default render.-%>
<%-- In case of getting data it must be verified, packed into an           --%>
<%-- ClientObjectWrapper and forwarded to the flow. This is done in a      --%>
```

```
<%-- correspondent message mapping JSP, that in this case is the           --%>
<%-- "PoActivityOutputMap.jsp",where we will use the input names to get the--%>
<%-- typed value.                                                          --%>
<%--*************************************************************************-->
<%@ page
   language="java"
   contentType="text/html;charset=UTF-8"
   import="com.ibm.bpe.client.*,
         java.util.*,
         org.apache.wsif.base.WSIFDefaultMessage,
         com.ibm.bpe.api.*"%>
<%--Getting a reference to the activity instance  --%>
<jsp:useBean id="activity" class="com.ibm.bpe.api.ActivityInstanceData"
scope="request">
</jsp:useBean>

<%
//get the activity ID string
String aidd = request.getParameter(Constants.WF_AIID);

//get the local reference of the business process EJB
LocalBusinessProcess process = MessageUtilities.getProcess(request);

//get the business process wrapped message
ClientObjectWrapper wrapOutMsg =
(ClientObjectWrapper)process.getOutputMessage(aidd);

//Extract the WSFI message from the wrap class
WSIFDefaultMessage outMsg = (WSIFDefaultMessage)wrapOutMsg.getObject();

//show message for activity READY
if(activity.getExecutionState() == ActivityInstanceData.STATE_READY) {%>
   <H4>This order has to be approved. </H4><%}

//show message for activity CLAIMED
if(activity.getExecutionState() == ActivityInstanceData.STATE_CLAIMED) {
   //Set default value for activity output input data
   boolean approvalInputValue = false;
   String commentInputValue = "";

   //Load value for activity output input data from an previous save, in case
   //the data was saved before (save button in the Activity Available Actions
form)
   if (outMsg != null) {
      commentInputValue = (String)outMsg.getObjectPart("comment");
      approvalInputValue = outMsg.getBooleanPart("isApproved");}%>
      <H4>What do you want to do?</H4>
      <blockquote>
      <TABLE width="100%">
```

```
         <TBODY>
            <TR class="marked1">
               <P><LABEL> <INPUT type="radio" name="approval_Input"
value="true"<% if (approvalInputValue)
               {%>checked<%}%>> Approve the order.</LABEL> <BR>
               <LABEL> <INPUT type="radio" name="approval_Input" value="false"
<% if (!approvalInputValue)
               {%>checked<%}%>> Reject the order.</LABEL></P>
            </TR>
            <TR class="marked1">
               <TABLE>
                  <TR class="marked1">
                     <P>Enter your comments:</P>
                  </TR>
                  <TR class="marked1">
                     <TEXTAREA rows="4" cols="30" name="comment_Input"><%=
commentInputValue %></TEXTAREA>
                  </TR>
               <TABLE>
            </TR>
         </TBODY>
      </TABLE>
   </blockquote>
 <%}

//show message for activity FINISHED
if(activity.getExecutionState() == ActivityInstanceData.STATE_FINISHED) {
   if (outMsg.getBooleanPart("isApproved")) {%>
      <H4>This order has been <b>approved</b>.</H4>
      <H4>With this comments: <%= (String)outMsg.getObjectPart("comment")
%></H4> <%}
   else {%>
      <H4>This order has <b>not been approved</b>.</H4>
      <H4>With this comments: <%= (String)outMsg.getObjectPart("comment")
%></H4><%} %>
<%} %>
```

Example 4-12 shows the implementation of the PoActivityOutputMap.jsp. The only difference in this code is the test to see if we already have a WSIF message object (be sure to save before). If so, we reuse it just to avoid unnecessary instance creation.

Example 4-12 PO Approval staff activity Output user-defined JSP (PoActivityOutputMap.jsp)

```
<%--************************************************************************--%>
<%-- This is the map JSP for the "PoActivityOutput.jsp" so here we         --%>
<%-- parse the typed data if it is invalid throws a ServletException with a--%>
<%-- description of the error. In this case the "error.jsp" will be called,--%>
```

```
<%-- because we have it in the page directive of this JSP, to show the error-%>
<%-- message and how to retry the operation. When all data is correct, we  --%>
<%-- wraps it in ClientObjectWrapper and forward to the flow.              --%>
<%--************************************************************************-->
<%@ page language="java"
   contentType="text/html;charset=UTF-8"
   import="java.util.*,
      com.ibm.bpe.client.*,
      com.ibm.bpe.api.*,
      com.ibm.bpe.api.ClientObjectWrapper,
      com.ibm.bpe.client.MessageUtilities,
      org.apache.wsif.base.WSIFDefaultMessage"%>
<%
//get the activity ID string
String aidd = request.getParameter(Constants.WF_AIID);

//get the local reference of the business process EJB
LocalBusinessProcess process = MessageUtilities.getProcess(request);

//get the business process wrapped message
ClientObjectWrapper wrapOutMsg =
(ClientObjectWrapper)process.getOutputMessage(aidd);

//Extract the WSFI message from the wrap class
WSIFDefaultMessage outMsg = (WSIFDefaultMessage)wrapOutMsg.getObject();

//if we do not have an previous output message create one and wrap it in
// the ClientObjectWrapper
if (outMsg == null) {outMsg = new WSIFDefaultMessage();
   wrapOutMsg = new ClientObjectWrapper(outMsg);}

//convert the string "true" or "false" to a Boolean type and add it to the
message
outMsg.setBooleanPart("isApproved", (new
Boolean(request.getParameter("approval_Input")).booleanValue()));

//add typed comment to the message
outMsg.setObjectPart("comment", (String)request.getParameter("comment_Input"));

//create an vector and add the request parameters to be excluded
Vector excludeRequestParameters = new Vector();
excludeRequestParameters.add("isApproved");
excludeRequestParameters.add("comment");
//forward message with the user typed data to the flow.
MessageUtilities.forwardMessageToController(request, response, wrapOutMsg,
null, excludeRequestParameters);
%>
```

Now we that we have all the files, we will add them to the Approval staff activity in the PO business process. In a Business Integration perspective of WebSphere Studio IE:

1. Open the PO business process, the PO.process file, and select the **Process** tab.
2. Right-click the **Approval** (staff activity) object and select **Properties**.
3. On the Properties for Approval window, select **Client** and click **Add**.
4. On the Add Definition window, select the action **ActivityDisplay** from the drop-down action list.
5. Add the activity user-defined JSPs to the ActivityDisplay points as shown in Figure 4-27, then click **OK**.

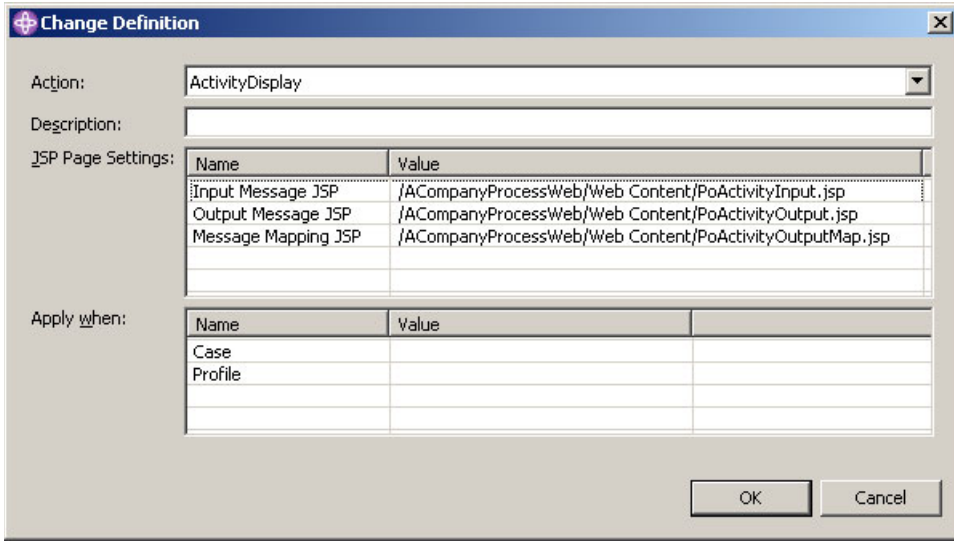


Figure 4-27 ActivityDisplay user-defined JSPs definition

6. Click **OK** on the Properties for Approval window and save the changes.

Now let's see the differences in the presentations of the PO business process with the user-defined JSPs.

Important: You cannot use the ViewContext in your user-defined JSPs without some lib changes. We did not need it in our sample because it is very simple. However, if you need internationalizing in your user-defined JSPs, the locale is in the context object. But because the ViewContext class is in the library (bpewebclient.jar) located inside of the Web client application bpewebclient (lib directory of the WAR), if you try to use it in the user-defined JSPs, you will receive the error:

```
Class com.ibm.bpe.client.ViewContext not found
```

If you make a copy of this JAR to the lib dir of your user-defined JSP Web project ACompanyProcessWeb, you will receive:

```
java.lang.ClassCastException: com.ibm.bpe.client.ViewContextImpl
```

If you really need to use this object, copy the bpewebclient.jar from the Web client application lib to WebSphere Application Server Enterprise lib (<*WebSphere_root*>\lib), or in WebSphere Studio IE test environment copy to <*Studio_root*>\runtimes\ee_v5\lib. So both Web applications bpewebclient and ACompanyProcessWeb will load it from a unique file system place.

Figure 4-28 on page 108 and Figure 4-29 on page 109 show the difference made by PoProcessInput.jsp. This JSP displays a better name for the data and changes the type for an explanation of the data to be entered.

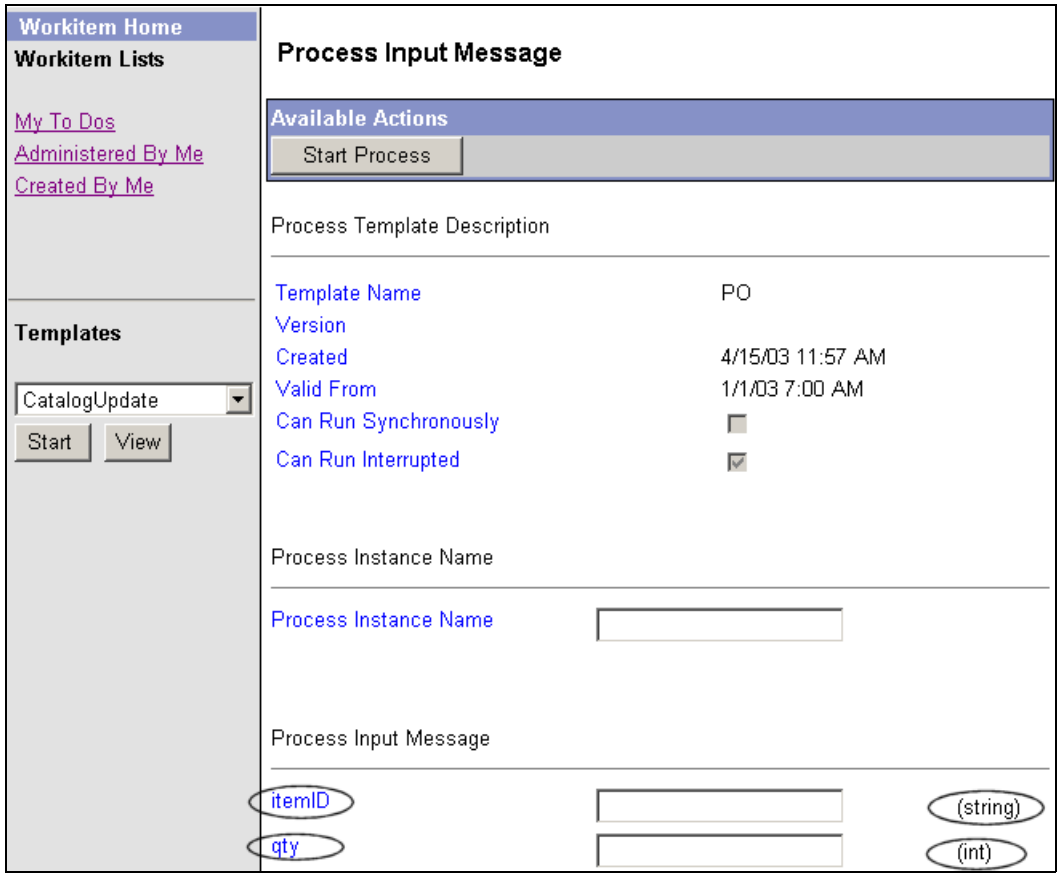


Figure 4-28 PO process input message without user-defined JSPs

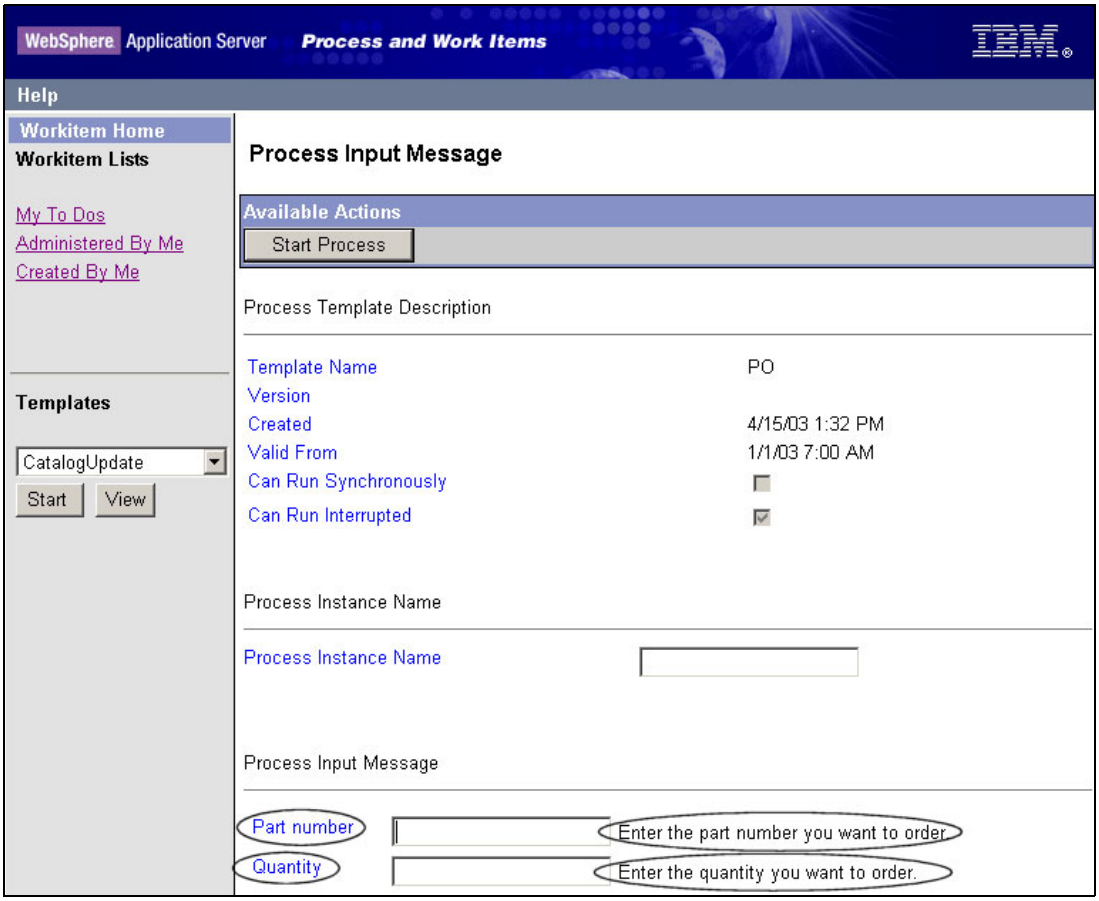


Figure 4-29 PO process input message without user-defined JSPs

Figure 4-30 on page 110 and Figure 4-31 on page 111 show the difference made by PoProcessInfoInput.jsp. This JSP displays a better name for the data and added an explanation of the data displayed.

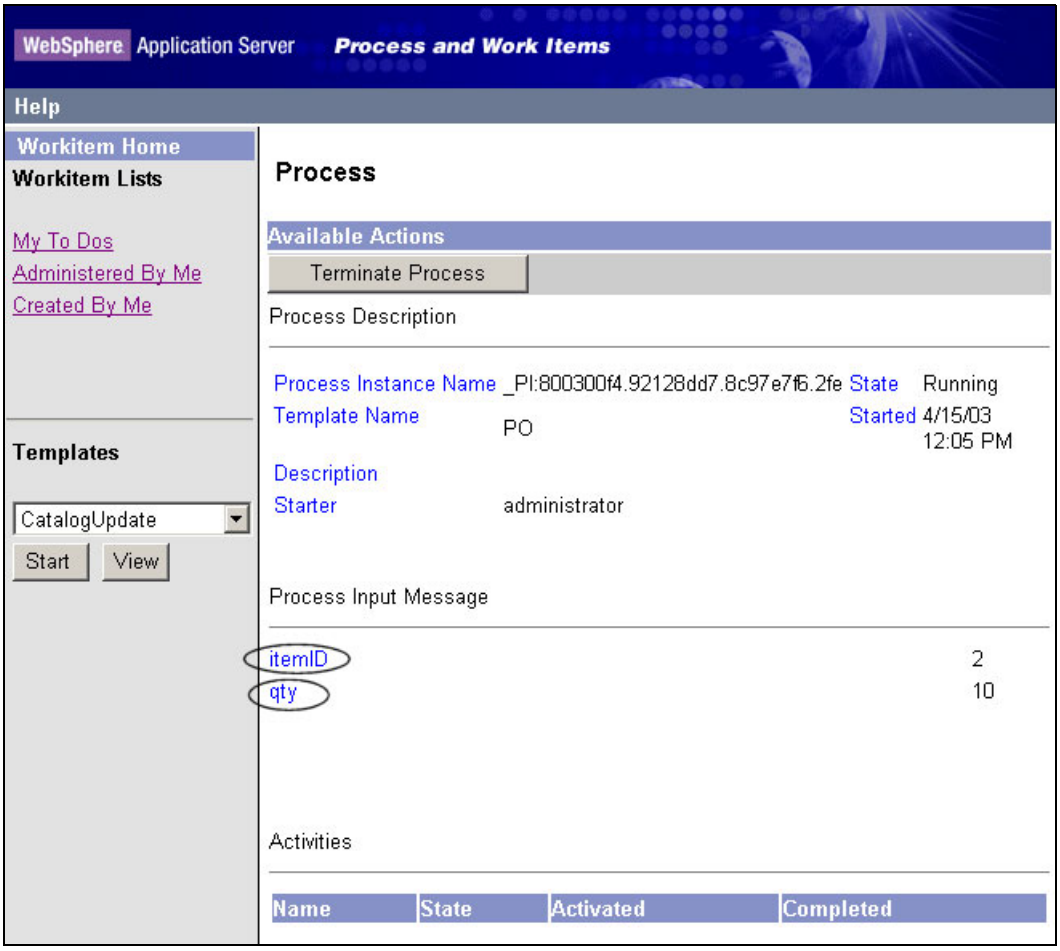


Figure 4-30 PO process input information without user-defined JSPs

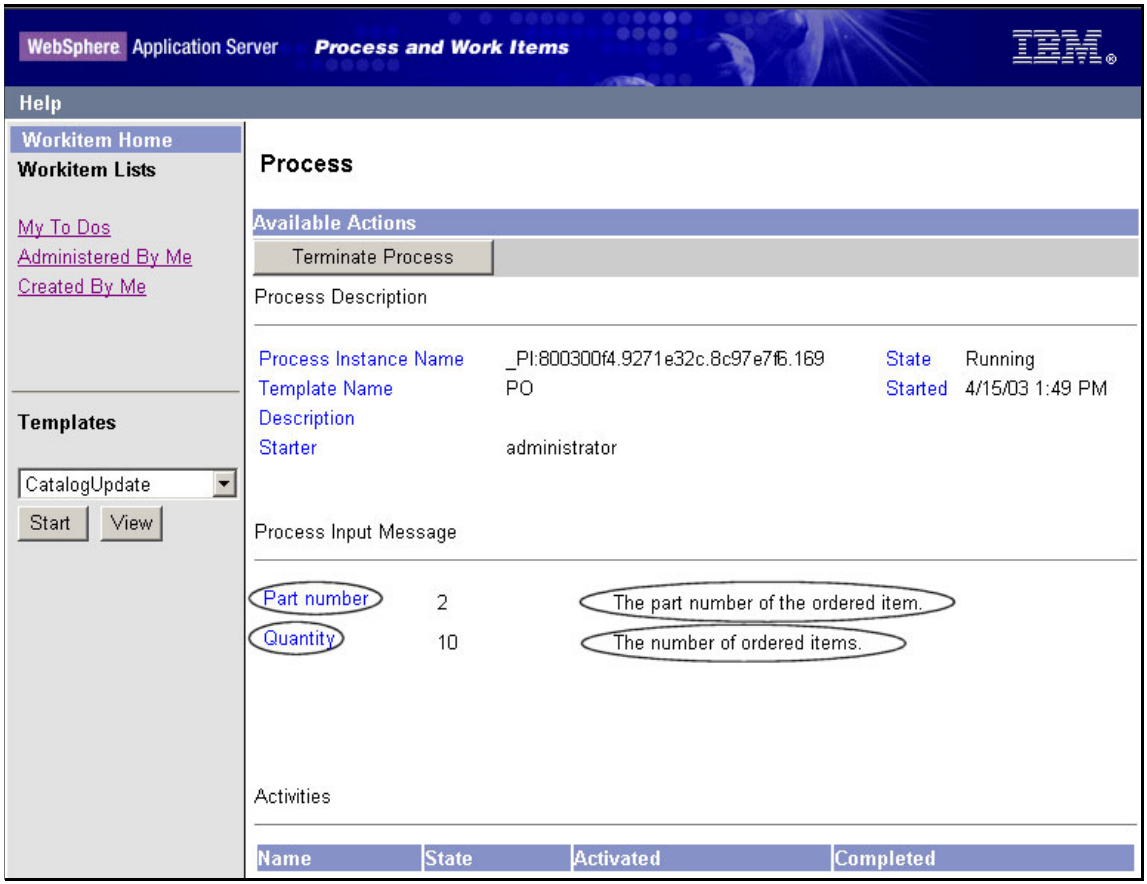


Figure 4-31 PO process input information with user-defined JSPs

Figure 4-32 on page 112 and Figure 4-33 on page 113 show the difference made by PoActivityInput.jsp for the activity input message, and PoActivityOutput.jsp for the activity output message for an ready activity.

The PoActivityInput.jsp JSP just displays a better name for the data and added an explanation of the displayed data.

The PoActivityOutput.jsp shows the output part names with no value. Once the activity is started, it shows a message saying that the other needs to be approved.

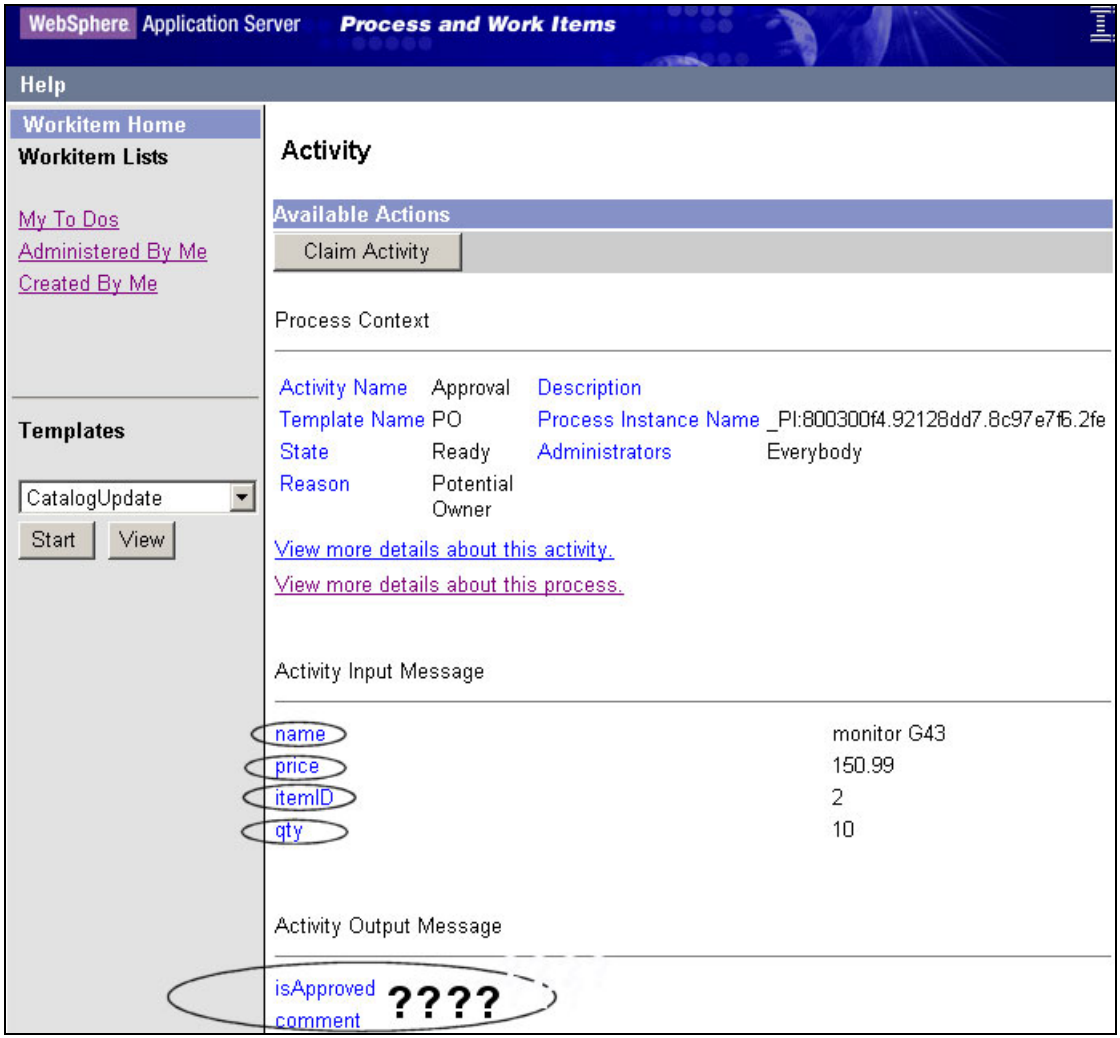


Figure 4-32 PO process ready Approval activity input/output message without user-defined JSPs

Figure 4-33 PO process ready approval activity input/output message with user-defined JSPs

Figure 4-34 on page 114 and Figure 4-35 on page 115 show the difference made by PoActivityInput.jsp for the activity input message, and PoActivityOutput.jsp/PoActivityOutputMap.jsp for the activity output message for a Claimed activity.

The PoActivityInput.jsp JSP displays a better name for the data and adds an explanation of the displayed data.

The PoActivityOutput.jsp, instead of showing an output part named isApproved and an entry field where the user should type true or false (a boolean field), it shows a message reminding the user of what he wants to do and two radio

buttons appear to approve or reject. By default, reject is always selected. In addition, instead of one entry field for the comments there is a text box.

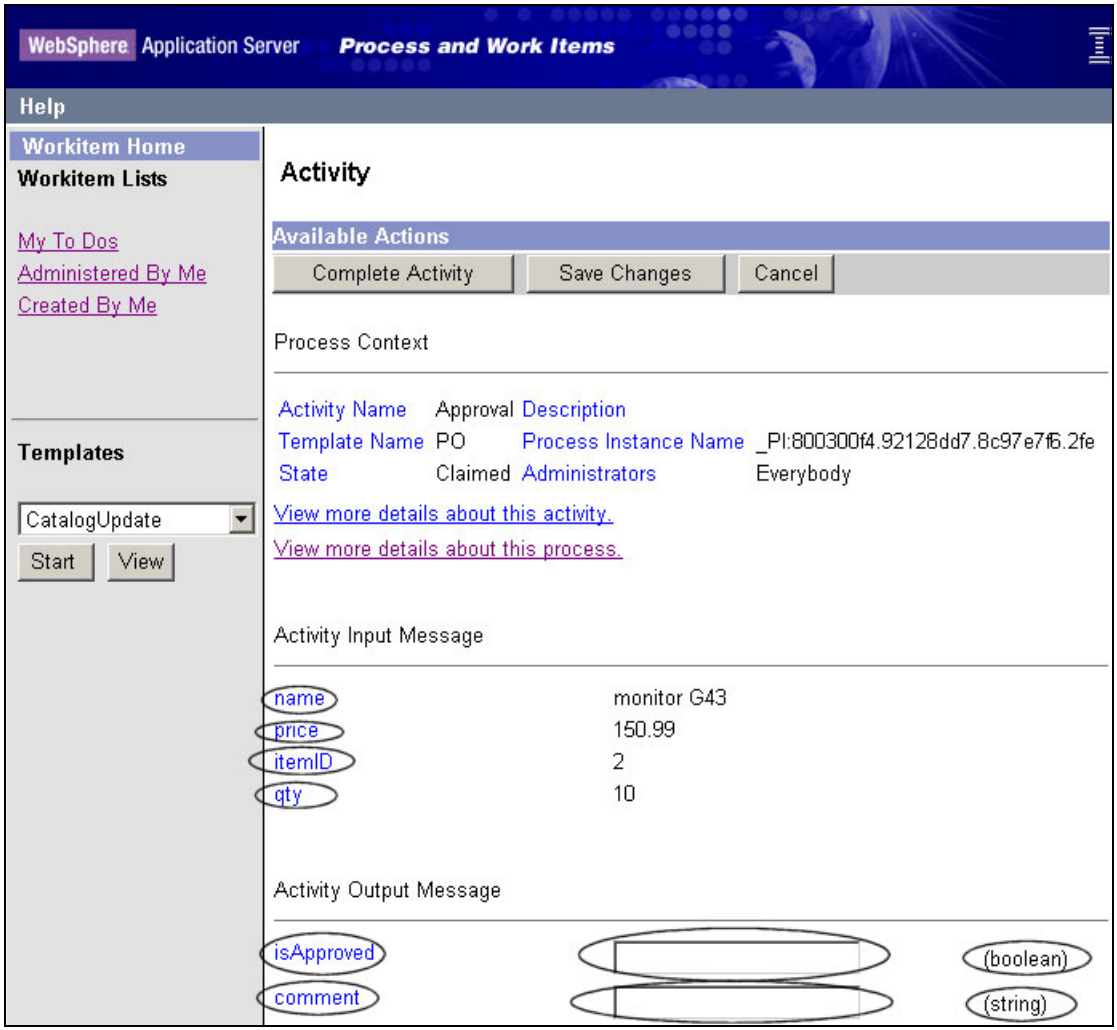


Figure 4-34 PO process claimed Approval activity input/output message without user-defined JSPs

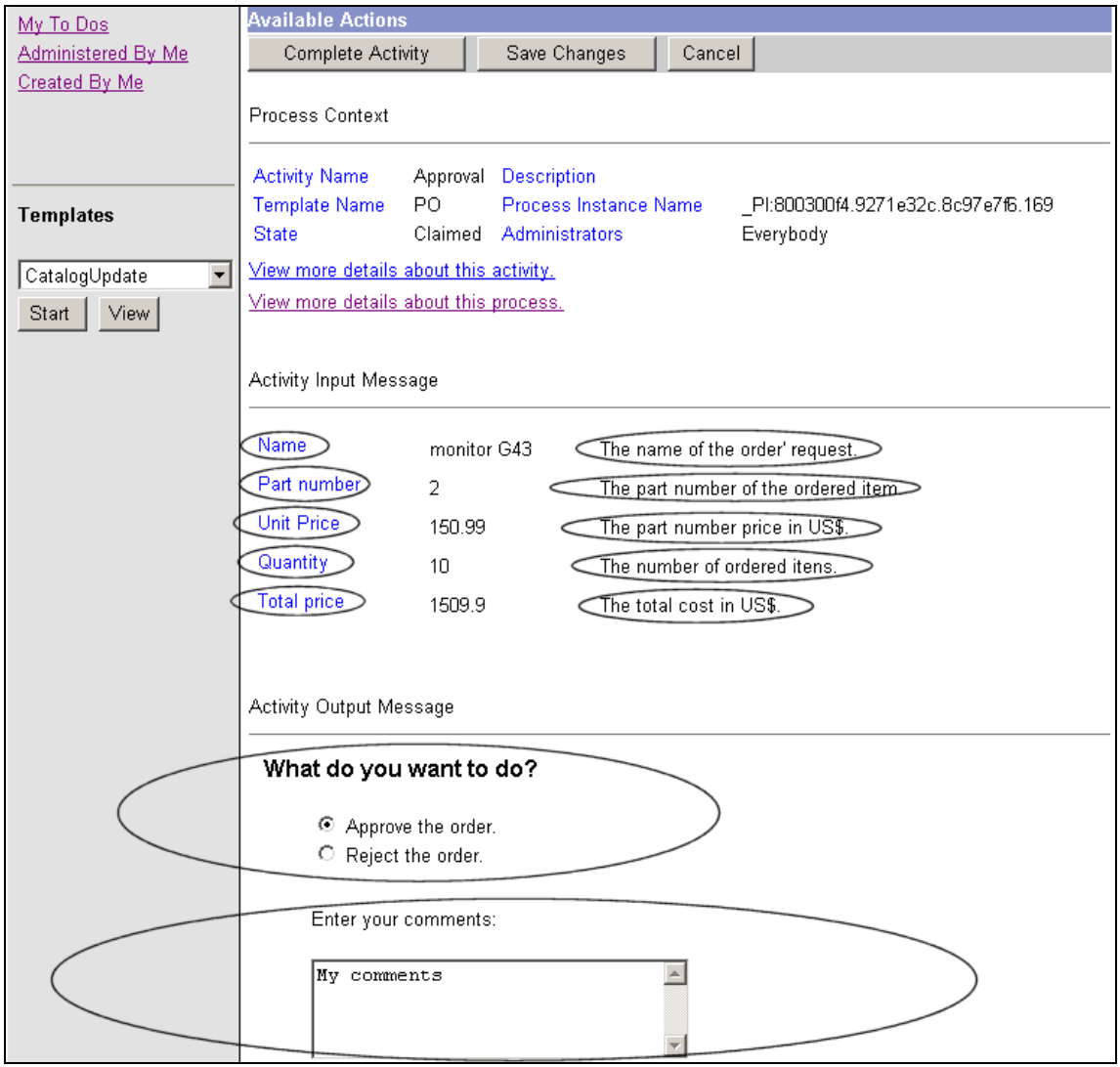


Figure 4-35 PO process claimed Approval activity input/output message with user-defined JSPs

Figure 4-36 on page 116 and Figure 4-37 on page 117 show the difference made by PoActivityInput.jsp for the activity input message, and PoActivityOutput.jsp/PoActivityOutputMap.jsp for the activity output message for a finished activity.

The PoActivityInput.jsp JSP displays a better name for the data and adds an explanation of the displayed data.

The PoActivityOutput.jsp instead shows the output part names with their value. It also shows a message saying that the other has been approved or rejected and provides some comments.

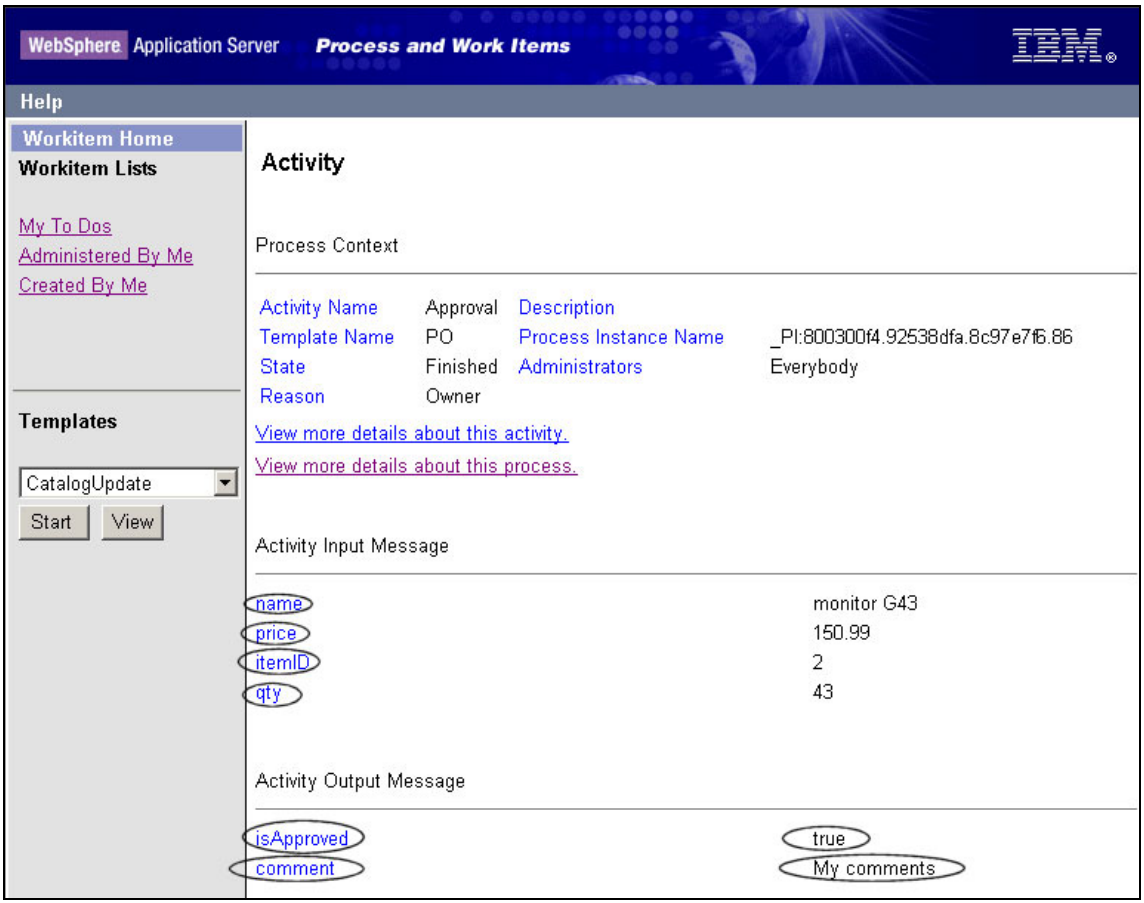


Figure 4-36 PO process finished approval activity input/output message without user-defined JSPs

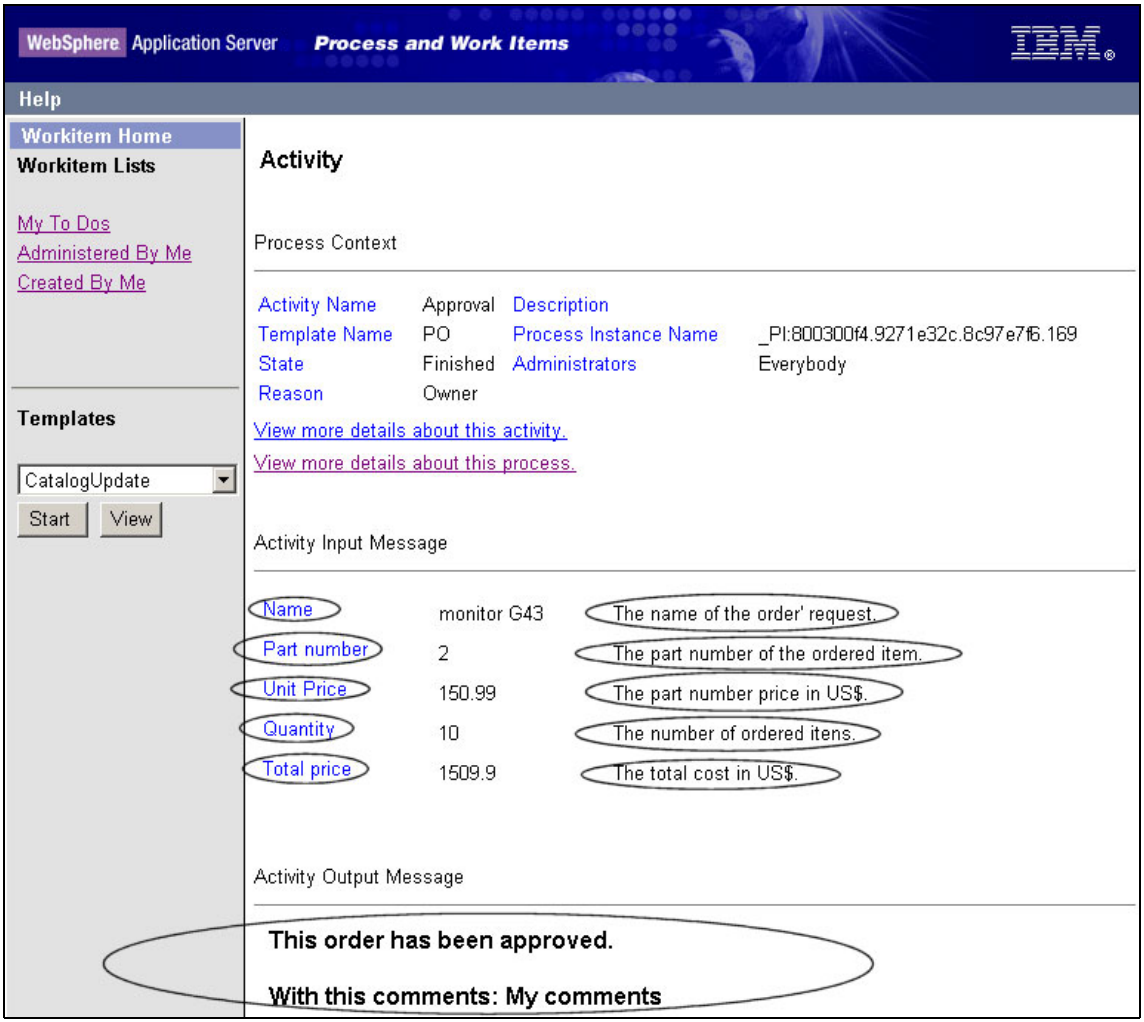


Figure 4-37 PO process finished Approval activity input/output message with user-defined JSPs

Changing the look and feel

To make any change in the look and feel, other than implementing the user-defined JSPs (see “Implementing business process-specific JSPs” on page 73) will require changes in the Web client default JSPs (see Table 4-3 on page 69), Cascading Style Sheets (dwc.css), configuration file (Configuration.xml), etc. The best approach to avoid problems is to create a copy of the Process Choreographer Web client application. We will create a new project in the WebSphere Studio IE and bring all the definitions, codes, and pages of the Web client (bpewebclient) to it.

Creating a copy of the Web client project

First, create a new EE Test Environment Server in WebSphere Studio Application Server Integration Edition.

During the creation of this server, the following projects of Process Choreographer will be loaded so we can run and debug process business applications:

- BPEContainer (EAR)
- bpecontainer_ejb (EJB)
- BPERemoteDeploy (EAR)
- bperemotedeploy_ejb (EJB)
- bpesoapclient (WEB)
- bpewebclient (WEB)
- compensate_ejb (EJB)

For more details, see 4.4, “Testing and debugging” on page 148.

To copy the entire Web client application to a new project:

1. In a Web perspective, right-click the Web project **bpewebclient**, then select **Copy**.
2. Right-click again and select **Paste**.
3. On the Copy Project window, enter the name of the new project as `MyBpewebclient`.
4. In the Enterprise application Project, select **New** and enter `MyBpewebclientEAR` for project name.

   Creating a new enterprise application for just the MyBpewebclient is a good idea, as it will be common for all business process.
5. Enter a new context root so we don’t need to remove the default Web client from the WebSphere Application Server Enterprise or from WebSphere Studio IE.
6. Click **OK**.

Two new projects were created in our sample application: the Web MyBpewebclient and the enterprise application MyBpewebclientEAR.

Important: Just remember that this new Web client project MyBpewebclient must be deployed in all instances of WebSphere Application Server Enterprise, because it uses local interfaces to access the business process AIP (EJB).

The MyBpewebclient gives a warning with the following description in the Task list:

```
IWAE0034W EJB link element ejb/local/BusinessProcessHome is unresolvable
in......
```

Do not worry about this, because this reference will be solved in the runtime. This message does not show in the BPEContainer.ear, because of a trick within the project. Try to rebuild this project, and you will see that you are not allowed. The trick will be discovered later in this section.

To finish with this project, we need to configure a security role:

1. Open the Deployment Descriptor of the MyBpewebclientEAR.
2. Select the **Security** tab and click **Add**.
3. Add the security role WebClientUser and click **Finish**.
4. On the Security tab, select the recently created **WebClientUser**.
5. On the WebSphere Bindings tab, check the **All authenticated users** box.
6. Save the changes and close the file.

We are finished with the MyBpewebclientEAR. Now let's work with the MyBpewebclient:

1. Select and delete the **imported_classes** folder.
2. The next steps will show how to remove a trick from the bpewebclient project that comes with the copy. Just to show the problem in the MyBpewebclient project, double-click the Activity.jsp in a Web perspective and select the Source tab and find the following lines of code:

   ```
   if (mode == null) {
      mode = Constants.CONFIG_MODE_DISPLAY;
   }
   ```

   Move the cursor after `Constants.` (just after the dot) and press and hold the Ctrl key and press the Space key (code assist). As you can see, code assist is not working, or you would see a pop-up with the variables and methods signatures of the Constants class. You can even type any wrong code and save this file without seeing any error.

   Important: Make sure you do the steps below when you copy the bpewebclient. Otherwise, this new project (MyBpewebclient) will miss the code assist syntax check support. You will only detect errors at runtime.

   Lets fix it. Close WebSphere Studio IE and using Windows Explorer, go to the WebSphere Studio IE repository where you build this project. Usually

WebSphere Studio IE shows this directory in the startup. Select the directory MyBpewebclient. In this directory, you should find a file named .project.

3. Now insert the lines that are boldfaced in Example 4-13.

Example 4-13 .project file of MyBpewebclient after the changes

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
   <name>MyBpewebclient</name>
   <comment></comment>
   <projects>
   </projects>
   <buildSpec>
      <buildCommand>
         <name>com.ibm.etools.webtools.LibDirBuilder</name>
         <arguments>
         </arguments>
      </buildCommand>
      <buildCommand>
         <name>org.eclipse.jdt.core.javabuilder</name>
         <arguments>
         </arguments>
      </buildCommand>
      <buildCommand>
         <name>com.ibm.etools.webtools.additions.linksbuilder</name>
         <arguments>
         </arguments>
      </buildCommand>
      <buildCommand>
         <name>com.ibm.etools.validation.validationbuilder</name>
         <arguments>
         </arguments>
      </buildCommand>
      <buildCommand>
         <name>com.ibm.etools.j2ee.LibCopyBuilder</name>
         <arguments>
         </arguments>
      </buildCommand>
      <buildCommand>
         <name>com.ibm.etools.ctc.serviceprojectbuilder</name>
         <arguments>
         </arguments>
      </buildCommand>
   </buildSpec>
   <natures>
      <nature>com.ibm.etools.beaninfo.BeaninfoNature</nature>
      <nature>com.ibm.etools.j2ee.WebNature</nature>
      <nature>org.eclipse.jdt.core.javanature</nature>
      <nature>com.ibm.etools.ctc.javaprojectnature</nature>
```

```
   </natures>
</projectDescription>
```

4. Restart WebSphere Studio IE again. In a Web perspective, right-click the **MyBpewebclient** project and select **Rebuild Project**. After it finishes, you will see lots of errors in this Task list. Almost all of them are unresolved references. To fix them, you need to add the bpe.jar and the wsif.jar to the project.
5. After you have added the libraries, you still have five errors and one warning. To fix the errors, double-click them in the Task list one-by-one. The file with an error opens. Then simply save it. The errors will disappear. This is caused by a known refresh problem in WebSphere Studio IE.
6. The warning message is a limitation in page designing. It is complaining of a broken link in the <a href="#skiptomain"> because it is looking only in its file (Header.jsp) for that, but you can find it defined in other JSP files, for example ActivityInformation.jsp.

Now our Web client is ready to run. To test it, add the MyBpewebclientEAR to your server in the WebSphere Studio IE. In a browser, type:

`http://localhost/mybpe/webclient`

Note: Note that the context root is `mybpe` this time.

Going around the Web client Splash screen

The splash UI is hard-coded in the com.ibm.bpe.client.CommandHandler class in the method showStatus(request, response, servlet). The only easy change is if you want to change the GIF image. It is located in the images/startAnimation.gif in the MyBpewebclient, but if you want a different layout or nothing at all, try the JSP shown in Example 4-14. This JSP can show any HTML you want for the time you set and bypass the default splash. The way it is set now, it will show a blank page for 0 milliseconds.

Example 4-14 index.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
```

```
<TITLE>index.jsp</TITLE>
<%
String st = request.getContextPath() + "/webclient?WF_Command=ListDisplay";
%>
<SCRIPT language = "JavaScript">
function setClientTimeZone() {
   var date = new Date();
   var clientTimeZone = date.getTimezoneOffset();
   document.CommandHandler.WF_ClientTimeZoneOffset.value=clientTimeZone;
<%--Line below last parameter 0 means wait for 0 mileseconds, if you want --%>
<%--to show some HTMLimage just put the time you want this message to stay--%>
   window.setTimeout("document.CommandHandler.submit()",0);
}
</SCRIPT>
</HEAD>
<BODY onLoad= setClientTimeZone()>
<FORM name= CommandHandler method=post action=<%= st %>>
<INPUT type=hidden name=WF_ClientTimeZoneOffset value="">
</FORM>
<%-- Insert any HTML/Image you want here or live it blank --%>
</BODY>
</HTML>
```

You can also add the index.jsp to the Welcome Pages of the MyBpewebclient, so you will only need to enter:

```
http://localhost/mybpe
```

Changing the Header

The header is generated by the Header.jsp. The image it shows is in the images/banner.jsp, so you can easily change your company banner. But the header.jsp also shows the Help menu, which points to the WebSphere Application Server Enterprise control center on the Internet:

```
http://publib7b.boulder.ibm.com/webapp/wasinfo1/index.jsp?deployment=Enterprise
&file=wfclient/concepts/c7webclt
```

You may want to remove it or change to your own.

Changing layout

Figure 4-38 on page 123 is a fragment of HTML that builds the default Web client layout. This fragment of HTML code is in almost all JSPs of the Web client. As you can see, each page of the default Web client layout consists of one main HTML table and the Header.jsp pane. The main table has just one row and one column and also includes the Navigation.jsp pane. The provided table cell is used to render the content, which can contain tables, forms, labels, inputs, images, and user-defined JSPs, etc. Therefore, with the exception of the

Header.jsp that renders the Header pane and the Navigation.jsp that renders the navigation bar pane, all other default JSPs (see Table 4-3 on page 69) render the content pane for a specific content.

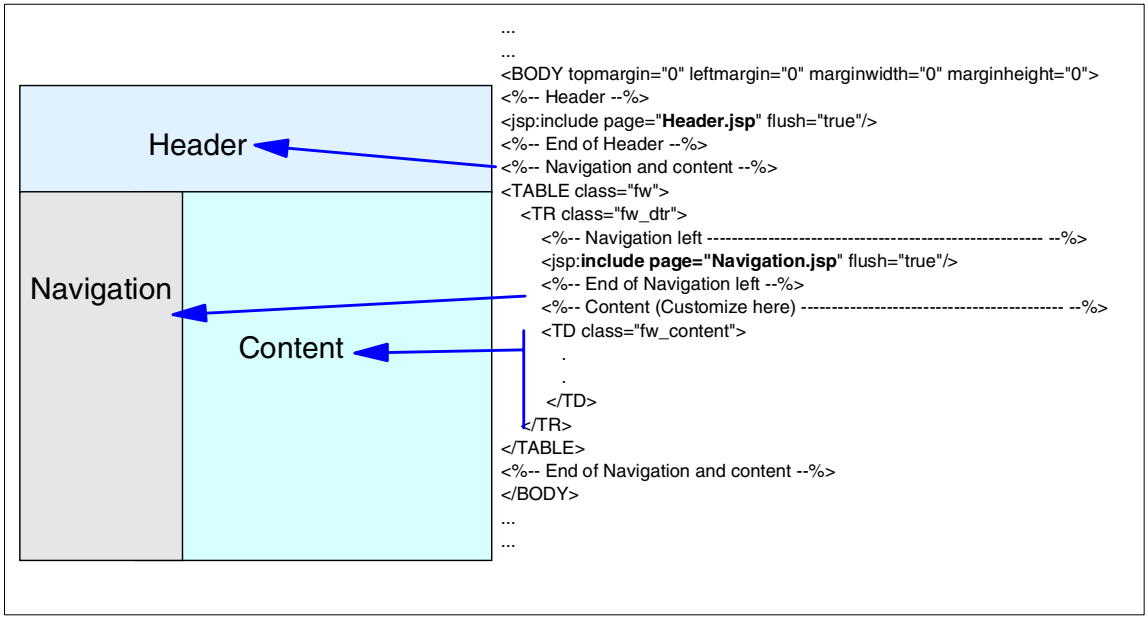


Figure 4-38 HTML template for Web client layout

Changing the content

One thing that we cannot change with the user-defined JSPs is the Process Instance Name window. It is always there for interruptible processes such as the PO process (see Figure 4-29 on page 109). If you enter it, the process instance will have this name. If you do not enter it, Process Choreographer will build one for the process instance; it must be unique. The problems, beside the pane layout, are:

- Usually the default will be acceptable. This name will appear in the UI.
- For process that you need a name that usually will be related with the physical process, like an document number that your company uses, or some information generated in other system, the name Process Instance Name probably will not better description for this information.
- As it is related to the process instance, even if it is not input data to this process, it is more natural if it is grouped with the process input data.

We will change the default ProcessInputMessage.jsp of our Web client application that we created in the project MyBpeWebClientWeb to change the behavior. The change we will do will allow the ProcessInputMessage.jsp to

behave like its default if you do not implement the ProcessInputMessage user-defined JSP, but if you implement one, it will be your responsibility to get this information for the Process Choreographer or not depending on your processing need. Example 4-15 shows the changes we had made to the default ProcessInputMessage.jsp (the changes are in bold). We can suppress the Process Instance Name if there is a user-defined JSP for the process input message:

```
if (inputMessageJSP == null) {
```

and set the process name and the message in the request so we can handle it in the process input message user-defined JSP:

```
request.setAttribute( "NOT_UNIQUE_MESSAGE", message);
request.setAttribute( "PROCESSNAME", value);
```

Example 4-15 Changes in the default ProcessInputMessage.jsp to suppress the Process Instance Name panel

```
<TABLE width="100%">
<% if (!processTemplate.getCanRunSynchronously()) { %>
<%-- Change to allows user-defined JSP, if any, to render the Process instance --%>
<%-- name, so you can give it name/format you want or even suppress it if it   --%>
<%-- is not meaningful for your process and the Choreographer will generat the --%>
<%-- default unique name.                                                      --%>
<%String value = "";
String message = null;
if (context.getViewAttribute(Constants.NOT_UNIQUE_MESSAGE) != null) {
   message = (String) context.getViewAttribute(Constants.NOT_UNIQUE_MESSAGE);
}
if (context.getViewAttribute(Constants.WF_PROCESSNAME) != null) {
   value = (String) context.getViewAttribute(Constants.WF_PROCESSNAME);
   value = new String (value.getBytes(request.getCharacterEncoding()),"UTF-8");
}
//Set this attribute so you can get them in the user-defined jsp ProcessInput
request.setAttribute( "NOT_UNIQUE_MESSAGE", message);
request.setAttribute( "PROCESSNAME", value);

if (inputMessageJSP == null) { %>
   <TR>
      <TD COLSPAN="3" ><%=dictionary.getString("PROCESS_NAME")%></TD>
   </TR>
   <TR>
      <TD COLSPAN="3">
         <hr noshade size="1">
      </TD>
   </TR>
   <TR>
      <TD class="fw_label"><%= dictionary.getString("PROCESS_NAME") %></TD>
```

```
      <TD><input name="<%= Constants.WF_PROCESSNAME %>" type="text" value="<%= value %>"
size="20"></TD>
<% if (message != null) { %>
      <TD CLASS="fw_warning"><%= message %></TD>
<% } else { %>
      <TD> </TD>
<% } %>
   </TR>
   <TR>
      <TD COLSPAN="3"> </TD>
   <TR>
   <TR>
      <TD COLSPAN="3"> </TD>
   <TR>
<%} // end of if (inputMessageJSP != null) %>
<%-- end of process instance name change --%>
<%}; // end of if (!processTemplate.getCanRunSynchronously %>
<% if (((attributeInputMessageNames != null) && (attributeInputMessageNames.size() != 0)) ||
(inputMessageJSP != null)) { %>
...
```

After these changes, if you run the PO process without a ProcessInputMessage user-defined JSP, you will see the same result rendered by the default Web client (see Figure 4-28 on page 108). But if you run the PO process with our ProcessInputMessage user-defined JSP PoProcessInput.jsp, the Process Instance Name pane will be suppressed and the default name will be generated for this instance (see Figure 4-39 on page 126).

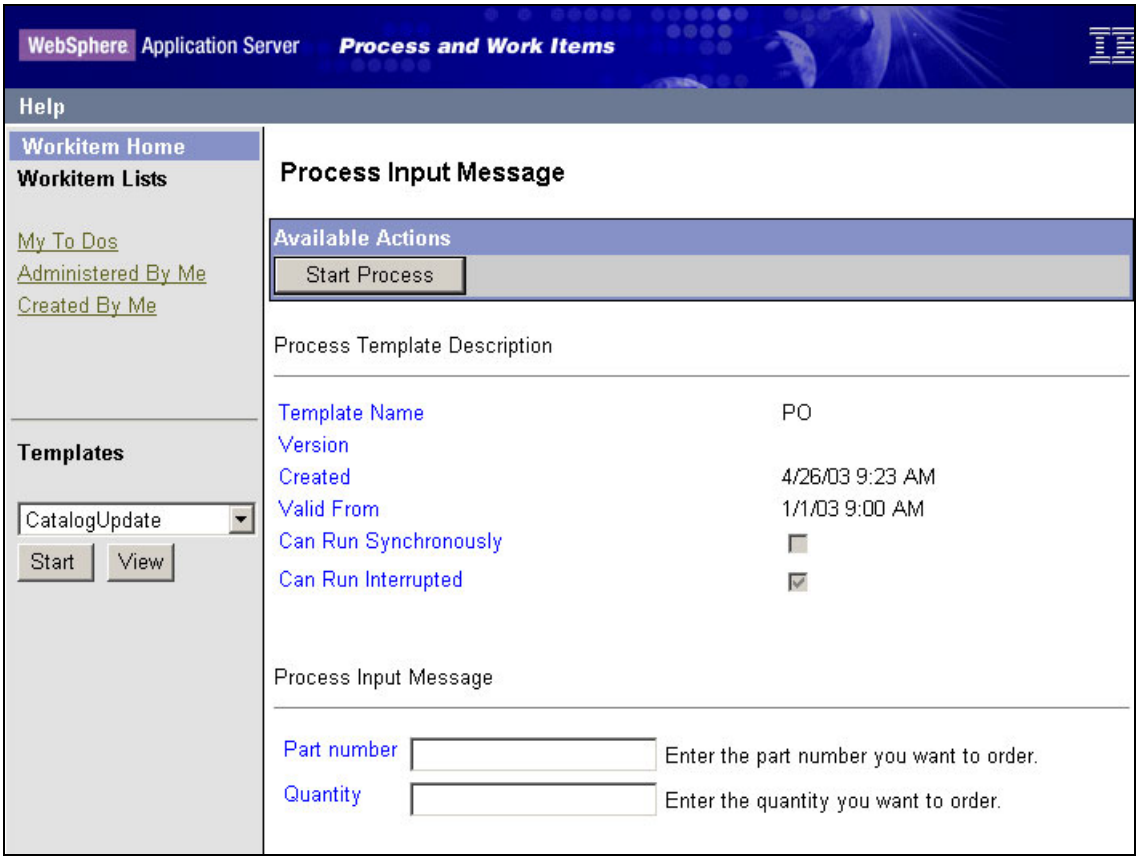


Figure 4-39 Process Instance Name suppressed with the new ProcessInputMessage.jsp.

Example 4-16 shows the new PO process input JSP that puts a new entry in the form, named Order Number, where we can enter the name we want for the process instance. We also check for non-unique names and receive a message if there are any duplicate process instance names. Figure 4-40 on page 128 shows the result of this new user-defined JSP, PoProcessWithOrderNumInput.jsp.

Example 4-16 PoProcessWithOrderNumInput.jsp new PO process input user-defined JSP

```
<%--************************************************************************--%>
<%-- This  JSP  will be rendered to get the information needed for at the   --%>
<%-- process start, and the process instance name called Order name in the  --%>
<%-- form  for the "PO Process". That means, every                          --%>
<%-- time the Web client need to get the input information for to start     --%>
<%-- this process it calls this JSP instead of using its default render. The--%>
<%-- data got here must be verified, packed into an ClientObjectWrapper and --%>
<%-- forward to the flow. This is done in a correspondent message mapping   --%>
```

```
<%-- JSP, that in this case is the  "PoProcessInputMap.jsp",where           --%>
<%-- we will use the input (like names "itemID_Input")to get the typed value--%>
<%--************************************************************************--%>
<%@ page
   language="java"
   contentType="text/html;charset=UTF-8"
   import="com.ibm.bpe.client.*,
      com.ibm.bpe.api.*"%>
<table>
<% //Test for duplicate Processor instance name error
if (request.getAttribute("NOT_UNIQUE_MESSAGE") != null) {
   //Get the Process Choreographer defaul message
   String msg = (String) request.getAttribute("NOT_UNIQUE_MESSAGE");
   //Change generic part of the message, to better fit our Application
   msg = "Order number "+msg.substring(msg.indexOf('\''));%>
   <%-- Show the error message --%>
   <tr CLASS="fw_warning"><td><%= msg %></td></TR>
</table>
<table> <%}
if (request.getAttribute("PROCESSNAME") != null) {%>
   <colgroup>
      <col width="20%" span="2">
      <col width="60%" span="1">
   </colgroup>
   <tr class="marked1">
      <td class="fw_label">Order Number</td>
<%-- The name of this input MUST BE Constants.WF_PROCESSNAME for the --%>
<%-- Process Choreographer process it right --%>
      <td> <input type="text" name="<%= Constants.WF_PROCESSNAME %>" value="<%=
request.getAttribute("PROCESSNAME") %>"> </td>
      <td>Enter Number for this order.</td>
   </tr> <%}%>
   <tr class="marked1">
      <td class="fw_label">Part number</td>
      <td> <input type="text" name="itemID_Input"> </td>
      <td>Enter the part number you want to order.</td>
   </tr>
   <tr class="marked1">
      <td class="fw_label">Quantity</td>
      <td> <input type="text" name="qty_Input"></td>
      <td>Enter the quantity you want to order.</td>
   </tr>
</table>
```

Figure 4-40 on page 128 shows the new PO process input message window.

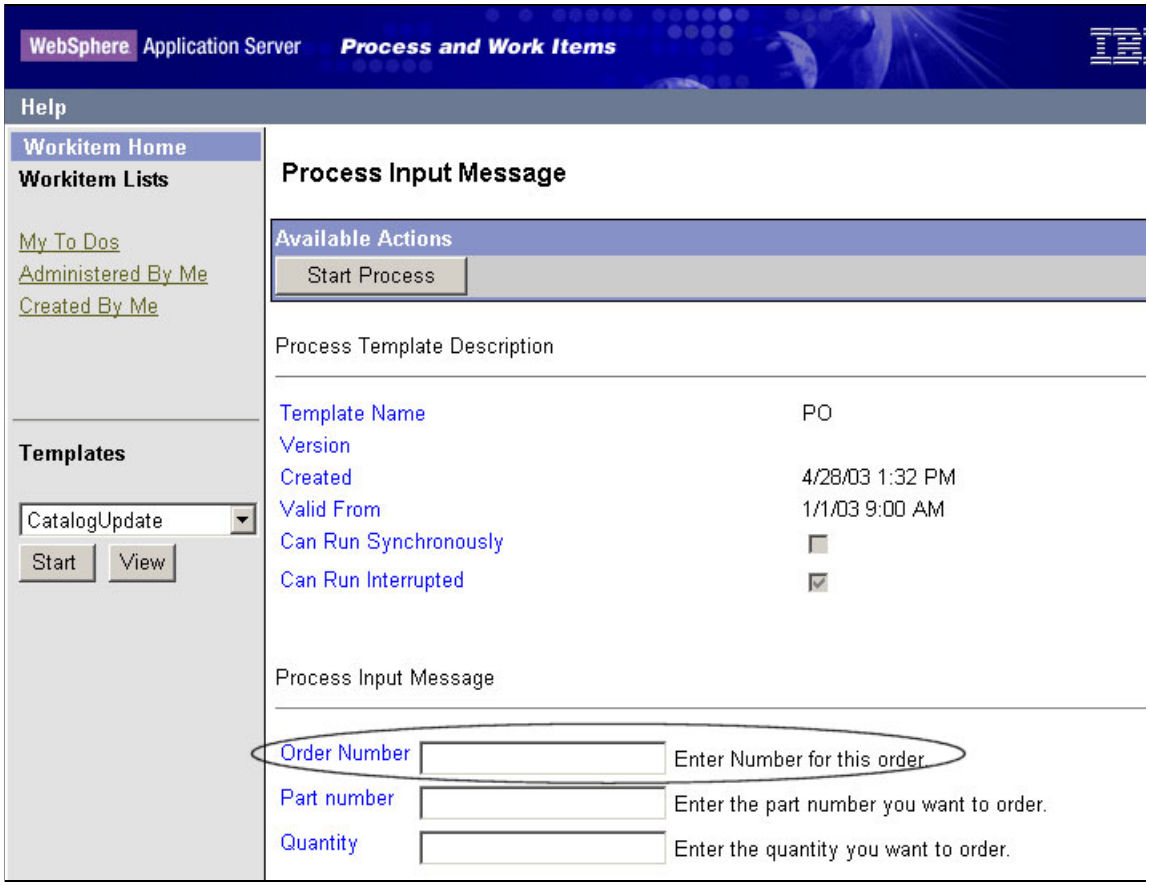


Figure 4-40 Result of the new PO process input message user defines JSP PoProcessWithOrderNumInput.jsp

Process Choreographer API

In this section, we will see how to call the business Process Choreographer API from a Java application, to start a business process, make a query, and so on. See Table 4-14 for more details about this API.

Table 4-14 Process Choreographer API function summary.

Api Functions	Description
call	Creates and executes a process instance from the specified process template and synchronously waits for the result

Api Functions	Description
callWithReplyContext	Creates and executes a process instance from the specified process template and asynchronously waits for the result
cancelClaim	Cancels the claim of an activity instance
claim	Claims a ready activity instance for user processing
complete	Completes a claimed activity instance
createMessage	Creates a message defined by the specified process template
delete	Deletes the specified top-level process instance and its subprocesses from the database
deleteWorkList	Deletes the specified worklist from the database
executeWorkList	Executes the query defined by the worklist and returns the qualifying object properties
forceComplete	Forces the completion of a stopped activity instance
forceRetry	Forces the repetition of a stopped activity instance
forceTerminate	Terminates the specified top-level process instance, its subprocesses, and its running or claimed activities and receive event
getActivityInstance	Retrieves the specified activity instance
getCustomAttribute	Retrieves the named custom attribute of the specified activity instance
getEventNames	Retrieves the receive event names of the specified process instance
getFaultMessage	Retrieves the specified fault message of the specified activity instance
getFaultTerminalNames	Retrieves the fault terminal names of the specified activity instance
getInputMessage	Retrieves the input message of the specified activity instance
getOutputMessage	Retrieves the output message of the specified activity instance
getOutputTerminalNames	Retrieves the output terminal names of the specified activity instance

Api Functions	Description
getProcessInstance	Retrieves the specified process instance
getProcessTemplate	Retrieves the specified process template
getUISettings	Retrieves user interface settings for the specified activity instance
getUserInput	Retrieves user input for the specified activity instance
getVariable	Retrieves the specified variable of the specified process instance
getWorkItems	Retrieves work item assignments for the logged-on user and the specified activity instance
getWorkList	Retrieves the specified worklist definition from the database
getWorkListNames	Retrieves the names of worklists persistently stored in the database
initiate	Creates a process instance from the specified process template, passes the specified input message, and initiates processing of the process instance
newWorkList	Creates a worklist and persistently stores it in the database
query	Retrieves selected object properties persistently stored in the database
queryProcessTemplates	Retrieves process templates persistently stored in the database
sendEvent	Sends the specified event to the specified process instance
setCustomAttribute	Stores custom-specific values for the specified activity instance
setFaultMessage	Stores the specified fault message for the specified activity instance in the database
setOutputMessage	Stores the output message of the specified activity instance in the database
setUserInput	Stores user input for the specified activity instance

This API is based on a session EJB, as you can see in Figure 4-41, that implements the remote interface:

- BusinessProcessHome
- BusinessProcess

and the local interface:

- LocalBusinessProcessHome
- LocalBusinessProcess

and the JNDI name:

- com/ibm/bpe/api/BusinessProcessHome.

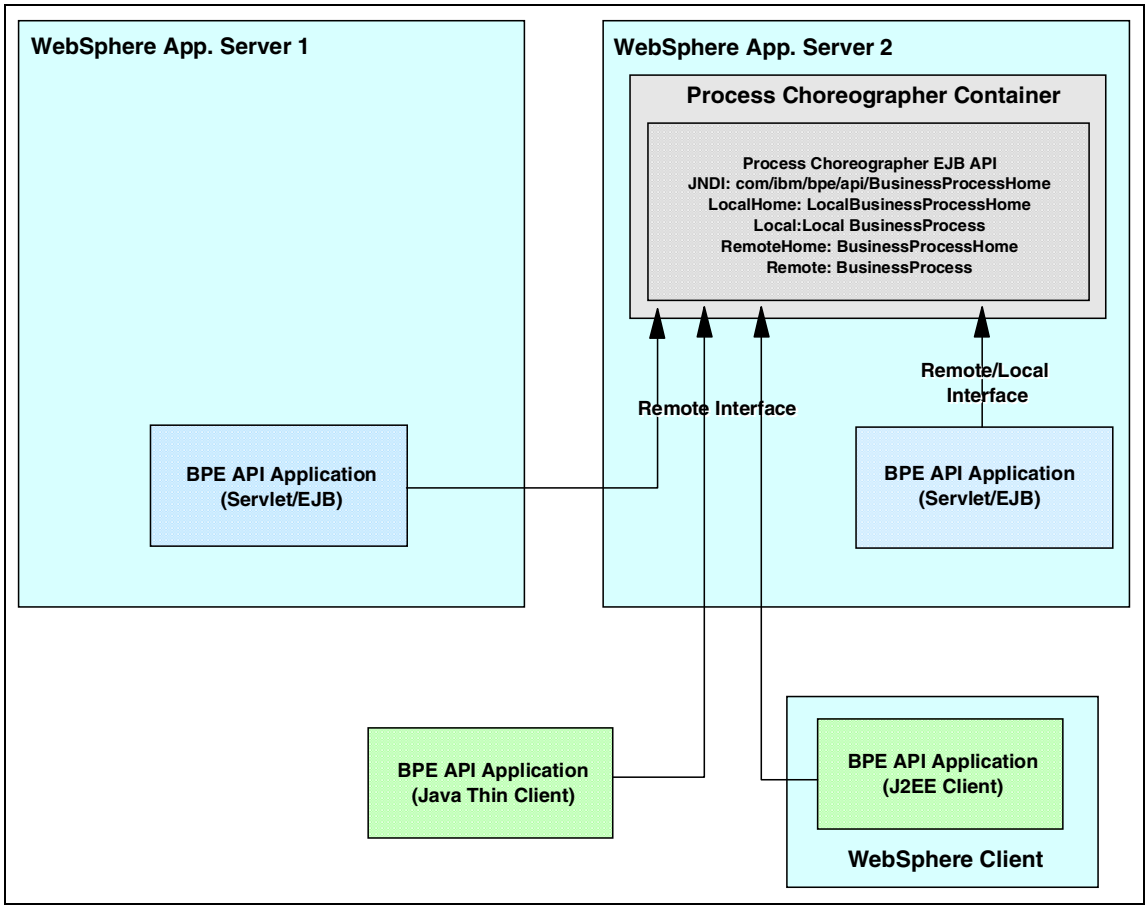


Figure 4-41 Business Process Choreographer API interfaces

The local interface is faster. That is why Process Choreographer Web client uses the local interface, but an application can only use this interface when it runs in the WebSphere Application Server Enterprise with the Process Choreographer installed. In all other cases the remote interface must be used. As you can see in Figure 4-41 on page 131, the difference between the J2EE client and the thin client is just the environment it runs. For the J2EE client, we need to install the WebSphere Client that came with WebSphere Application Server. The thin client runs over an IBM JVM plus some JARs, as you will see.

We have two samples: one Java client using a remote interface that will call the CatalogUpdate business process (using the call function off the API), and one servlet using a local interface that will start the PO business process (using the initiate function off the API). In the case of the Java client, we will show how to test it under the WebSphere Studio IE, and how to deploy and run it in the WebSphere Client and as a Java thin client.

Process Choreographer API remote interface Java client

This application is in the application client project named ACompanyClient. It has two Java classes: the Client.java in the default package, and the CatalogUpdateAPI.java in the ACompanyGUI package. You need to add two libraries to our project: the bpe.jar and the wsif.jar.

Example 4-17 shows the Client.java. It has the main() method that creates an instance of CatalogUpdateAPI class (our GUI implementation class). Add an window event listener to it, so when it closes, the JVM is ended. After that, the window appears.

Example 4-17 Client.java class

```
import ACompanyGUI.CatalogUpdateAPI;
public class Client {
   public Client() {
      super();
   }
   public static void main(String[] args) {
   try {
     // Creates an instance of the GUI class
     CatalogUpdateAPI aCatalogUpdateAPI = new CatalogUpdateAPI();
     // Add a windowListener for the windowClosingEvent
     aCatalogUpdateAPI.addWindowListener(new java.awt.event.WindowListener() {
      public void windowOpened(java.awt.event.WindowEvent e) {
      }
      public void windowClosing(java.awt.event.WindowEvent e) {
      //Exit the JVM when the window closes
            System.exit(0);
      }
      public void windowClosed(java.awt.event.WindowEvent e) {
```

```
        }
        public void windowIconified(java.awt.event.WindowEvent e) {
        }
        public void windowDeiconified(java.awt.event.WindowEvent e) {
        }
        public void windowActivated(java.awt.event.WindowEvent e) {
        }
        public void windowDeactivated(java.awt.event.WindowEvent e) {
        }
    });
       //Turn the GUI (window) visible
      aCatalogUpdateAPI.setVisible(true);
     } catch (Throwable exception) {
       System.err.println("Exception occurred in main() of aCatalogUpdateAPI");
       exception.printStackTrace(System.out);
     }
    }
}
```

We have to register the Client class as the main class. In a Java perspective, expand the **ACompanyClient** project and its META-INF folder, and open the **MANIFEST.MF** file. In the JAR Dependency Editor pane, select the **Dependencies** tab, and in the Main-Class field, enter **Client**. Save the changes and close the file.

The CatalogUpdateAPI was created using the WebSphere Studio IE Java Visual Editor. With this tool you can create a Java visual class, with Swing or AWT components, using drag-and-drop of components.

Building a Java GUI application is beyond the scope of this book. In Figure 4-42 on page 134, you can see the finished Swing GUI. Let's add the code to access the CatalogUpdate process.

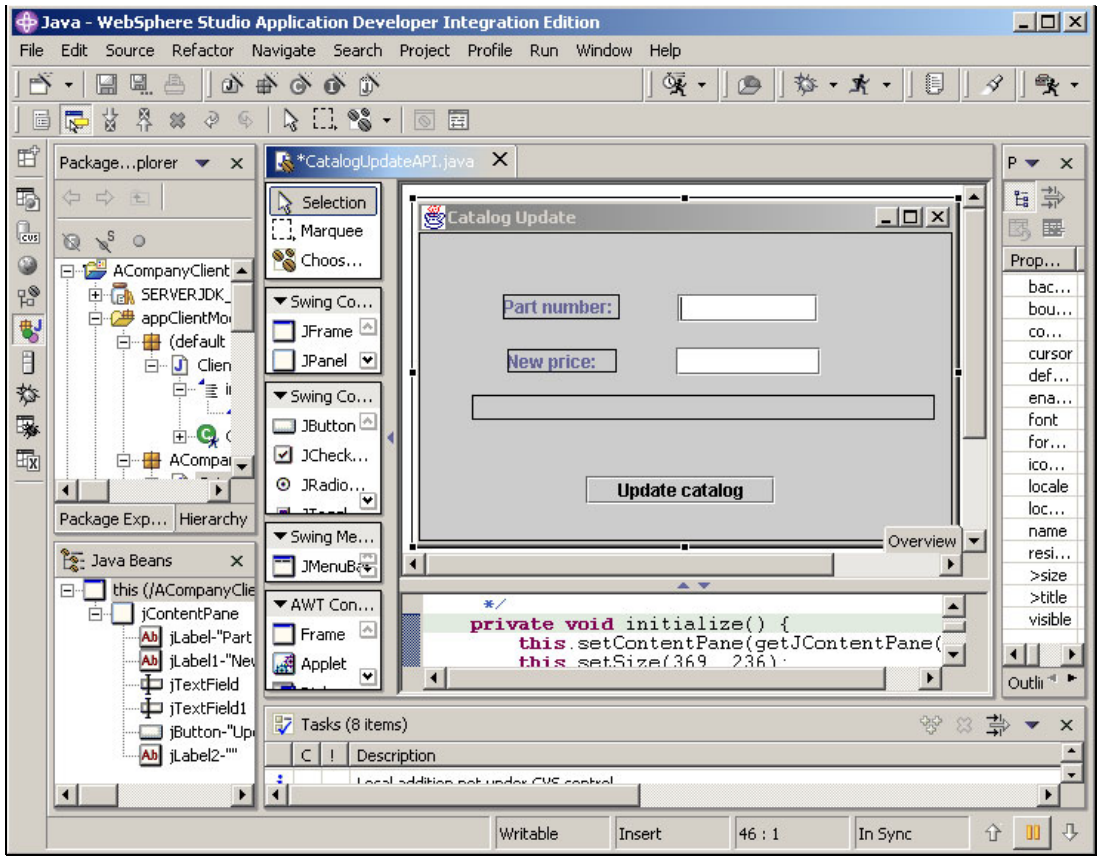


Figure 4-42 Visual Editor for class CatalogUpdateAPI

To understand the code of CatalogUpdateAPI callCatalogUpdateProcess() method (Example 4-18 on page 135), we need to know the meaning of the following GUI variables:

- jTextField: Instance of javax.swing.JTextField() for the part number.
- jTextField1: Instance of javax.swing.JTextField() for the new price.
- jLabel2: Instance of javax.swing.JLabel for the messages.

This code is called when you click the **Update catalog** button on the GUI (ActionEvent).

1. Validate the entered data in the Swing GUI.
2. Get the initial context to do the JNDI lookup.
3. Do the lookup for the com/ibm/bpe/api/BusinessProcessHome, which is the JNDI name of the Business Process API EJB.

4. Narrow the returned object to the BusinessProcessHome, which is the Home remote interface for the business process, and create an instance of BusinessProcess Bean remote interface.
5. Create a message with the data entered in the Swing GUI.
6. Call the business process API for the CatalogUpdate process passing the message created with the Swing GUI data. Since this is a synchronous process, we have to wait for the response.
7. Test the response and set the right response message in the Swing GUI.

Example 4-18 Method callCatalogUpdateProcess() of CatalogUpdateAPI class

```
/**
    * This method calls the Remote Process Choreographer, as this client
    *  will run outside the WebSphere Application Server, API to start
    * the CatalogUpdate process with GUI data for the input message
    */
   private void callCatalogUpdateProcess(){

    InitialContext context=null;
   BusinessProcessHome processHome=null;
   BusinessProcess processServ = null;

   // Validate the typed data
   try {
     // check new part number input
     if (jTextField.getText().equals("")){
     jLabel2.setText("Please enter a Part number.");
      return;}

     // check new price input
     if (jTextField1.getText().equals("")) {
      jLabel2.setText("Please enter a New price.");
      return;}

     //check new price for a valid data type
     try {
      Double.valueOf(jTextField1.getText()).doubleValue();}
     catch (Exception ex) {
     jLabel2.setText("The New price " + jTextField1.getText()+
                     " is not valid.");
      return;}

     // Get properties for the JNDI initial context
     Properties p = new Properties();;

     //Supply the provider url of the server
     p.put(javax.naming.Context.PROVIDER_URL,"IIOP://localhost");
```

```
 //Supply the initial context factory
 p.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
       "com.ibm.websphere.naming.WsnInitialContextFactory");

 // Obtain the initial JNDI context
 context = new InitialContext(p);

 // Lookup the remote home interface of the BusinessProcess bean
Object result = context.lookup("com/ibm/bpe/api/BusinessProcessHome");

 // Convert the lookup result to the proper type (IIOP cast)
processHome = (BusinessProcessHome)
                  javax.rmi.PortableRemoteObject.narrow(result,
                 BusinessProcessHome.class);

// Create the BusinessProcess session bean's remote interface
processServ = processHome.create();

//Create a WSIF message
WSIFDefaultMessage aMessage = new WSIFDefaultMessage();

 //Add the typed part number to the process message
aMessage.setObjectPart("itemID", jTextField.getText());

 //Add the typed new price to the process message
aMessage.setDoublePart("price",
            (Double.valueOf(jTextField1.getText()).doubleValue()));
//Add the catalogURL to the process message, we do not need it now
//but as it is defined in the input it must be in the message
aMessage.setObjectPart("catalogURL", "");

 //Call the API to start the "CatalogUpdate" template with the as
//aMessage input data and wait for the return result in the resp
ClientObjectWrapper resp = processServ.call("CatalogUpdate",
                                new ClientObjectWrapper(aMessage));
if (result != null)
 {
 //Unpack the message
 WSIFMessage rtnMsg = (WSIFMessage) resp.getObject();

 //Get the part name updated from the message
 java.lang.Integer rc  = (java.lang.Integer)
                         rtnMsg.getObjectPart("updated");

 //Test the return to set the right complition message in the gui
 if (rc.intValue() == 1) {
     jLabel2.setText("Successfully updated");}
 else  {jLabel2.setText("not updated");}}}
```

```
  catch (Exception e)
     { e.printStackTrace();
      jLabel2.setText(e.getMessage());}}
}  //  @jve:visual-info  decl-index=0 visual-constraint="-4,-2"
```

Before you can test the client, you need to start the Test Environment server of the sample application. Once the server is running, follow the steps below:

1. In a Java perspective, select **Run -> Run...** from the menu.
2. On the Launch Configurations window, right-click the WebSphere V5 Application Client and select **New**.
3. Enter a name, for example `ACompanyClient`.
4. Select **WebSphere v5 EE** for the server type.
5. Select **ACompany** for the enterprise application.
6. Click **Apply** then click **Run**.
7. The Catalog Update window should appear, as shown in Figure 4-43.
8. Enter `1` for the part number and `1.2` for the new price, then click **Update catalog**.
9. In the login window, type a valid user name and password for the application server.

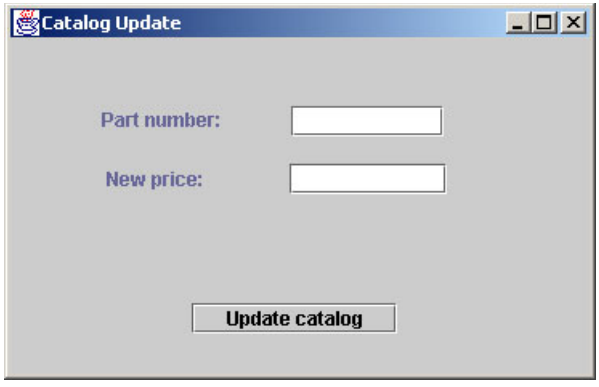


Figure 4-43 Catalog Update GUI

10. After a successful update, a message appears, as shown in Figure 4-44 on page 138.

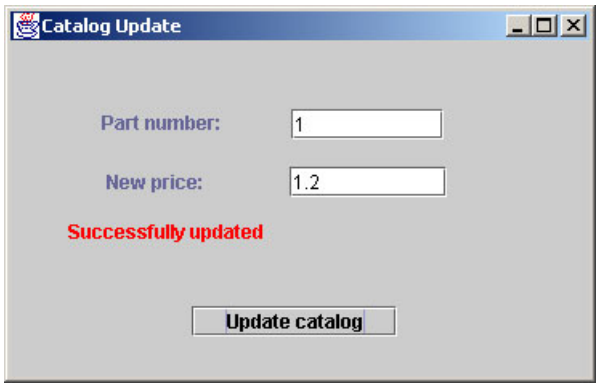


Figure 4-44 Catalog Update GUI after a successful update

Deploying the ACompanyClient in WebSphere Client

Install the WebSphere Client. Export the ACompany enterprise application as EAR file to the file system, let's say c:\acompany\ACompany.ear.

Important: You need to copy two library files, bpe.jar and bpe137650.jar, from <*Studio_root*>\runtimes\ee_v5\lib to <*WebSphere_client_root*>\lib.

Before testing the client, we need to start the Test Environment server of our sample application. After that, open a command prompt and change to the <*WebSphere_client_root*>/bin directory. Start a WebSphere client for the ACompany application with the following command:

```
launchclient c:\apompany\ACompany.ear -CCBootstrapHost=<AppServer_hostname>
-CCBootstrapPort=<AppServer_port>
```

If you do not specify the host name and a port number, the client will connect to the localhost using the default port 2809.

After a little while the Catalog Update window should appear. See Figure 4-43 on page 137.

Important: If you receive the following error:

```
java.io.InvalidClassException: javax.xml.namespace.QName; Local class not
compatible: stream classdesc serialVersionUID=-9120448754896609940 local
class serialVersionUID=1
        at
java.io.ObjectStreamClass.validateLocalClass(ObjectStreamClass.java:565)
        at java.io.ObjectStreamClass.setClass(ObjectStreamClass.java:609)
        at
java.io.ObjectInputStream.inputClassDescriptor(ObjectInputStream.java:981)
```

Copy the qname.jar library from <*Studio_root*>\runtimes\ee_v5\lib to the <*WebSphere_client_root*>\lib.

Deploying the ACompanyClient in WebSphere Client

Important: This procedure will work only if you use the IBM JDK.

Let's create the directory structure to build our Java thin client for the ACompanyClient application C:\acompanythin\libs.

Copy the following library files from <*Studio_root*>\runtimes\ee_v5\lib to C:\acompanythin\libs:

- bpe.jar
- bpe137650.jar
- commons-logging-api.jar
- ecutils.jar
- ffdc.jar
- idl.jar
- iwsorb.jar
- j2ee.jar
- naming.jar
- namingclient.jar
- qname.jar
- ras.jar
- tx.jar
- txPrivate.jar
- utils.jar
- wsdl4j.jar
- wsexception.jar
- wsif.jar

Copy the implfactory.properties file from <*WebSphere_Studio_IE_root*>\runtimes\ee_v5\properties to the c:\acompanythin, as follows:

1. In WebSphere Studio IE at the Java perspective, expand the **ACompanyClient** project, right-click **appClient Module** and select **Export**.
2. On the Export window, select **File system** and click **Next**.
3. Browse for the directory and select **c:\acompanythin**.
4. Make sure that **Create only selected directories** is selected, and click **Finish**.

Now you should have all classes, files, and library files that we need to run the client. We create a catUudate.bat file in the c:\acompanythin (see Example 4-19) to make it easy run the thin client. As you can see, we include in the Java classpath the library files of c:\acompanythin\libs, the property file directory, and the ACompanyClient classes directory. We also change to the <*WebSphere_root*>\java\bin directory to use IBM JDK.

Example 4-19 .bat file to run the ACompanyClient as a Java thin client

```
SET MY_CLASSPATH=.;C:\acompanythin;C:\acompanythin\appClientModule;C:\acompanythin\libs\idl.jar
;C:\acompanythin\libs\naming.jar;C:\acompanythin\libs\ras.jar;C:\acompanythin\libs\wsexception.
jar;C:\acompanythin\libs\j2ee.jar;C:\acompanythin\libs\ffdc.jar;C:\acompanythin\libs\iwsorb.jar
;C:\acompanythin\libs\bpe13765O.jar;C:\acompanythin\libs\bpe.jar;C:\acompanythin\libs\tx.jar;C:
\acompanythin\libs\utils.jar;C:\acompanythin\libs\txPrivate.jar;C:\acompanythin\libs\wsif.jar;C
:\acompanythin\libs\wsdl4j.jar;C:\acompanythin\libs\commons-logging-api.jar;C:\acompanythin\lib
s\qname.jar;C:\acompanythin\libs\namingclient.jar;C:\acompanythin\libs\ecutils.jar;
c:
cd "\Program Files\WebSphere\AppServer\java\bin"
java -classpath %MY_CLASSPATH%  Client
```

Before testing the client, we need to start the Test Environment server of our sample application. After that, open a command prompt, change to c:\acompanythin directory, and run the catUpdate.bat file.

After a little while the Catalog Update window should appear, as shown in Figure 4-43 on page 137.

Process Choreographer API Local interface servlet client

This application will start a PO process of our sample. Remember that the PO process is an interruptible flow so it does not return. That means we just start it and it goes asynchronous, and we are finished. For this application, we will have one HTML file that will render a form to get and validate (JavaScript) the PO process input data, and on the submit it will call a servlet that will invoke the Process Choreographer API (local interface). If the servlet can start the PO

process, it forwards a JSP and passes in the request an affirmative message. In case of an exception, the same JSP is forwarded but with an error message.

Create the project ACompanyWeb in the ACompany enterprise application and ACompany as context root. Since we will access the Process Choreographer API, we need to add two library files to it: the bpe.jar and the wsif.jar. Let's create in the ACompanyWeb project two files:

- poInputData.html
- infoPage.jsp

In Example 4-20 is the code for poInputData.html file. As you can see, it is just a form to get the part number and the quantity (PO process input data). We also have a JavaScript (doSubmit()) called by using the Submit button. This JavaScript validates the entered data and if the data is valid, it submits the form, which will call the ProcessPoApi servlet. If the data is not valid, the JavaScript sends an alert to the user with information about the error. When you save the file, we will see a warning about a broken link. This is because the servlet ProcessPoApi is not created yet.

Example 4-20 poInputData.HTML code

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>PO Submit</TITLE>
</HEAD>
<SCRIPT language="JavaScript">
function doSubmit(pn,qty) {
   if (pn.value == "") {
      alert('Please enter a part number.');
      return true;
   } else {
      if (qty.value == "") {
         alert('Please entrer a quantity.');
         return true;
      } else {
         if ((parseInt(qty.value) <= 0) || isNaN(qty.value)) {
            alert('Please entrer a valid quantity.');
            return true;
         } else {
            document.poInput.submit();
         }
      }
   }
```

```
}
</SCRIPT>
<BODY>
<FORM name=poInput action="/ACompany/ProcessPoApi">
<TABLE border="1">
   <TBODY>
      <TR>
         <TD>
            <table>
               <colgroup>
                  <col width="20%" span="2">
                  <col width="60%" span="1">
               </colgroup>
               <tr class="marked1">
                  <td class="fw_label">Part number</td>
                  <td> <input type="text" name="itemID_Input" > </td>
                  <td>Enter the part number you want to order.</td>
               </tr>
               <tr class="marked2">
                  <td class="fw_label">Quantity</td>
                  <td> <input type="text" name="qty_Input" ></td>
                  <td>Enter the quantity you want to order.</td>
               </tr>
         </table>
         </TD>
      </TR>
      <TR>
         <TD></TD>
      </TR>
      <TR>
         <TD align="center"><INPUT type="button" name="Submit" value="Submit"
onclick="doSubmit(itemID_Input,qty_Input)"></TD>
      </TR>
   </TBODY>
</TABLE>
</FORM>
</BODY>
</HTML>
```

In the Example 4-21 you see the code of the infoPage.jsp, which shows the info attribute received in the request.

Example 4-21 infoPage.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <head>
      <title>Information</title>
      <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

```
   </head>
   <style type="text/css">
      p,a,font {
      font-family:Verdana, Arial, Helvetica, sans-serif;
      font-size: 10pt;
      }
   </style>
<%@ page import="java.io.*"
   isErrorPage="true"
   contentType="text/html;charset=UTF-8"
%>
<body>
   <H3><% out.println(request.getAttribute("info"));%></H3><br>
   <br>
</body>
</html>
```

To define the servlet ProcessPoApi:

1. In a Web perspective, expand the **ACompanyWeb** project and open the Web Deployment Descriptor file.
2. Select the **Servlet** tab, and click **New**.
3. In the New Servlet window, enter `poBusinessProcess` for the package name, `ProcessPoApi` for the Class Name, then click **Finish**.

At this point, the file ProcessPoApi.java is created and opened, so let's add the code in the servlet. First, we will add the method processRequest(req, resp), the code for this method is in Example 4-22. The main difference from the remote interfaces are:

- You have to do lookup using the java:comp/env. Therefore, we need to define the reference name ejb/local/BusinessProcessHome in the EJB Local reference in the Web Deployment Descriptor, as follows:

  ```
  context.lookup("java:comp/env/ejb/local/BusinessProcessHome",...
  ```

- We just cast the lookup result to the local type. In the remote interface, we need to narrow (IIOP) the new type to do this cast.

  ```
  processHome = (LocalBusinessProcessHome)result
  ```

- As PO is an interruptible process. we do not wait for any response.

  ```
  processServ.initiate("PO",new ClientObjectWrapper(aMessage))
  ```

Example 4-22 Method processRequest(req, resp)

```
/**
* This method calls the Local Process Choreographer API , as servlet will run
* in the WebSphere Application Server Enterprise , to start the
```

```
* PO process with poInputData.html data for the input message
*/
public void processRequest(HttpServletRequest req,HttpServletResponse resp)
   throws ServletException, IOException{

   InitialContext context=null;
   LocalBusinessProcessHome processHome=null;
   LocalBusinessProcess processServ = null;

   // get the Business Process EJB
   try {
      // Obtain the initial JNDI context
      context = new InitialContext();

      // Lookup the local home interface of the BusinessProcess bean
      Object result =
      context.lookup("java:comp/env/ejb/local/BusinessProcessHome");

      // Convert the lookup result to the proper type
      processHome = (LocalBusinessProcessHome)result;

      // Access the BusinessProcess session bean's remote interface
      processServ = processHome.create();

      //Create a WSIF message
      WSIFDefaultMessage aMessage = new WSIFDefaultMessage();

      //Add the typed part number to the process message
      aMessage.setObjectPart("itemID", req.getParameter("itemID_Input"));

      //Add the typed quantity to the process message
      aMessage.setIntPart("qty",
      Integer.valueOf(req.getParameter("qty_Input")).intValue());

      //Call the API to start the "PO" template with the aMessage as input
data
      processServ.initiate("PO",new ClientObjectWrapper(aMessage));

      //Send the complete message using the InfoPage.jsp
      req.setAttribute("info", "PO Process has been started");
      req.getRequestDispatcher("InfoPage.jsp").forward(req, resp);
   } catch (Exception e) {
        //Print the error and the stack
      System.out.println("--------- "+ e.getMessage()+
                                          "/"+e.fillInStackTrace());
      //Send the error message using the InfoPage.jsp
       req.setAttribute("info",
                   "An Exception occured when starting the PO process");
       req.getRequestDispatcher("InfoPage.jsp").forward(req, resp);
```

```
}}
```

We need to add a call to the processRequest(req,resp) in the doPost(req,resp) and doGet(req,resp) as follows:

```
public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
   processRequest(req, resp);
   public void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
       processRequest(req, resp);
```

1. To define the local EJB reference ejb/local/BusinessProcessHome, open the Web Deployment Descriptor of ACompanyWeb:
2. Select the **References** tab and **EJB Local** (at the top of the pane).
3. Click **Add**, and override the New EJB Local Ref with ejb/local/BusinessProcessHome.
4. Select **session** for the type.
5. Set the JNDI Name to com/ibm/bpe/api/BusinessProcessHome.
6. Now we have to enter the Local: and the Local home: values. But we are not allowed to type in these fields, and the Browse only looks for local interfaces defined in the WebSphere Studio IE projects. Since the interface we need to define is in the library file bpe.jar, we do not see it using the Browse button. To bypass this limitation, see the following restriction.

Restriction: In the Web Deployment Descriptor Editor on the References tab and EJB local, we can add local references, select its type (Session/Entity), and enter the JNDI name, but it is not possible to add the Local home or the Local interface names if these interfaces are not defined in any WebSphere Studio IE project. To bypass this limitation, add the local EJB reference (with its type and JNDI name), and save this in the Deployment Descriptor Editor (press and hold the Ctrl key and press the s key). On the Source tab, your local EJB definition should looks like this:

```
<ejb-local-ref>
   <ejb-ref-name>ejb/local/BusinessProcessHome</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
   <local-home></local-home>
   <local></local>
</ejb-local-ref>
```

Enter your local - home and local names in the file. You will see result below:

```
<ejb-local-ref>
   <ejb-ref-name>ejb/local/BusinessProcessHome</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
   <local-home>com.ibm.bpe.client.api.LocalBusinessProcessHome</local-home>
   <local>com.ibm.bpe.client.api.LocalBusinessProcess</local>
</ejb-local-ref>
```

Now if you return to the References tab, you can see the names there.

Once the descriptor is saved, WebSphere Studio IE will generate an ID for the ejb-local-ref tag and will insert it as an attribute.

To test the application, you have to publish it and start the Test Environment server of the sample application. Open a browser with the following URL:

```
http://localhost:9080/ACompany/poInputData.html
```

You can see the result in Figure 4-45 on page 147. To test, enter `2` for the part number and `6` for the quantity. You will receive a confirmation as shown in Figure 4-46 on page 148.

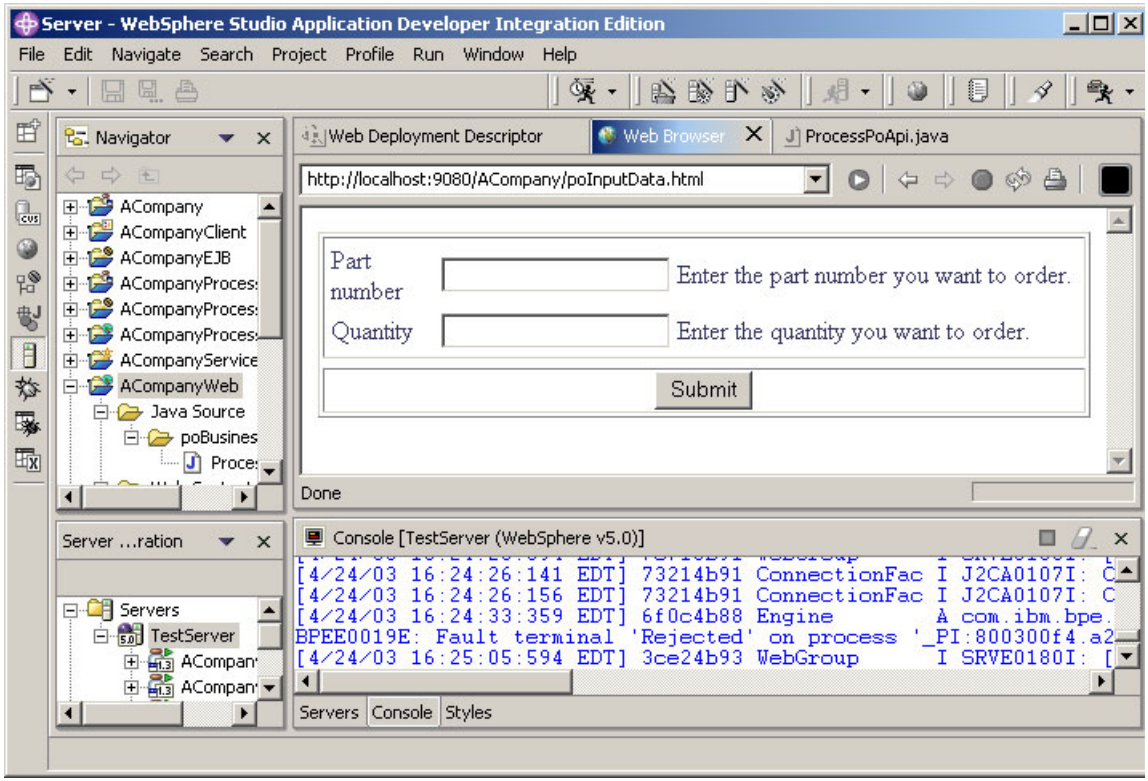


Figure 4-45 PO process start form

Now if you start the Web client the order request we have created is waiting for your approval.

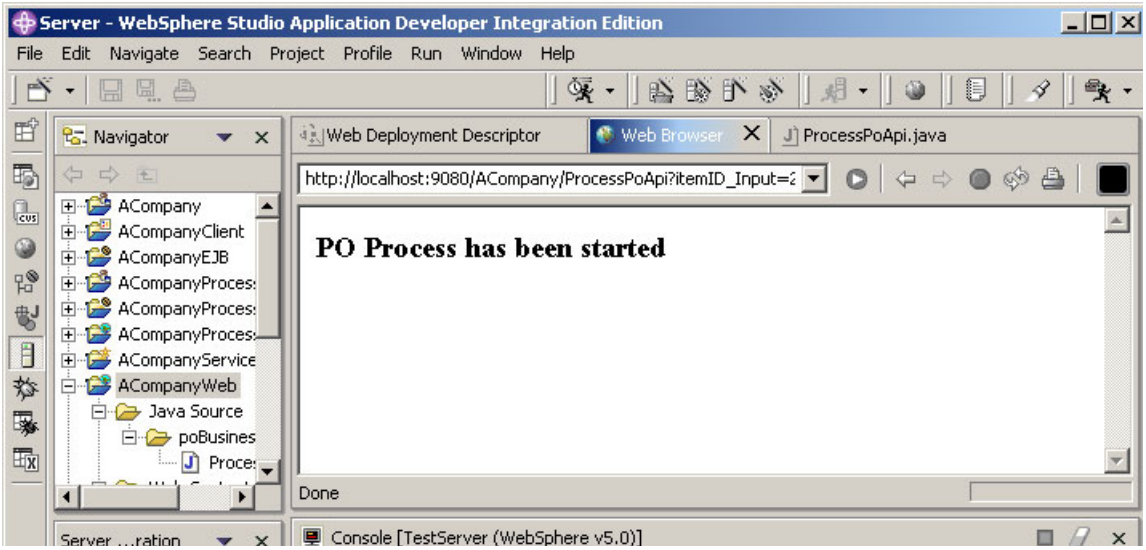


Figure 4-46 PO process started confirmation

4.4 Testing and debugging

Debugging problems in your business process logic can be difficult unless you can choose to step through the code from the business process level, to identify the business process component problem, step through the Java code to see more details on the problem, and fix it. WebSphere Studio IE provides a process debugger that can be used together with the normal Java debugger capabilities, if needed, to make problem determination of your business process easier. The three main components used to support business process debugging are:

- The process engine runs in the application server where the business process is deployed. It can be a WebSphere Application Server Enterprise V5 or the EE Test Environment in the WebSphere Studio IE V5.
- The process debugger controls the execution of the process debugger and runs in the WebSphere Studio IE V5 Process Debug perspective.
- IBM Agent Controller is responsible for the communication between the process debugger and the process engine. It must be installed in the same system as the process engine, and even if the process debugger and the process engine are in the same system, such as when you use EE Test Environment in the WebSphere Studio IE.

4.4.1 Unit test environment

First you need to create a EE Test Environment server, configure it, and deploy the sample. If you do not know how to do it, see “Development environment” on page 667.

Running only the process debugger

If you do not need to debug through the Java code, just the business process, follow this fast procedure. The process debugger allows you to set breakpoints in the control links, so you can stop in any control link and step through the business process. You can set a breakpoint by opening the .process file. Select the **Process** tab in a Service perspective, then double-click the control link or right-click the control link and select **Add Breakpoint**. A small ball appears in the control link representing the breakpoint. Figure 4-47 on page 150 shows the Catalog update process with two breakpoints:

- In the control link between the TransformRequest and CatalogItemUpdate.
- In the control link between the TransformResponse and Updated.

Now let’s run the catalog update process in the process debug mode to stop at these breakpoints.

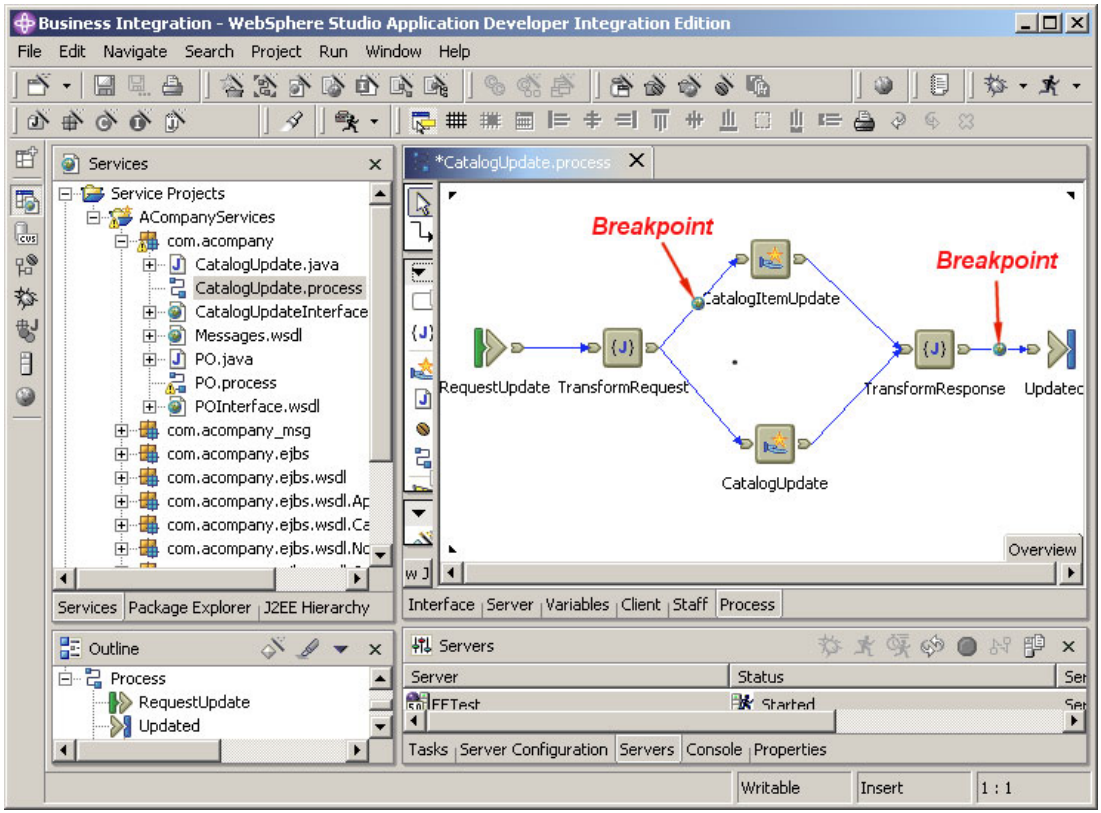


Figure 4-47 Catalog Update process with two breakpoints

1. Start the EE Test Environment, right-click the server under the Servers tab of a Server perspective, and select **Start**.
2. Wait until the following message appears in the Console tab of the Server perspective:

   ```
   WSVR0001I: Server server1 open for e-business
   ```

3. Open a Process Debug perspective and select **Window -> Open Perspective -> Process Debug** from the menu bar. As you can see in Figure 4-48 on page 151, in the Process Debug perspective you have buttons to attach to the process. Step through the process flow and at the end detach from the process.

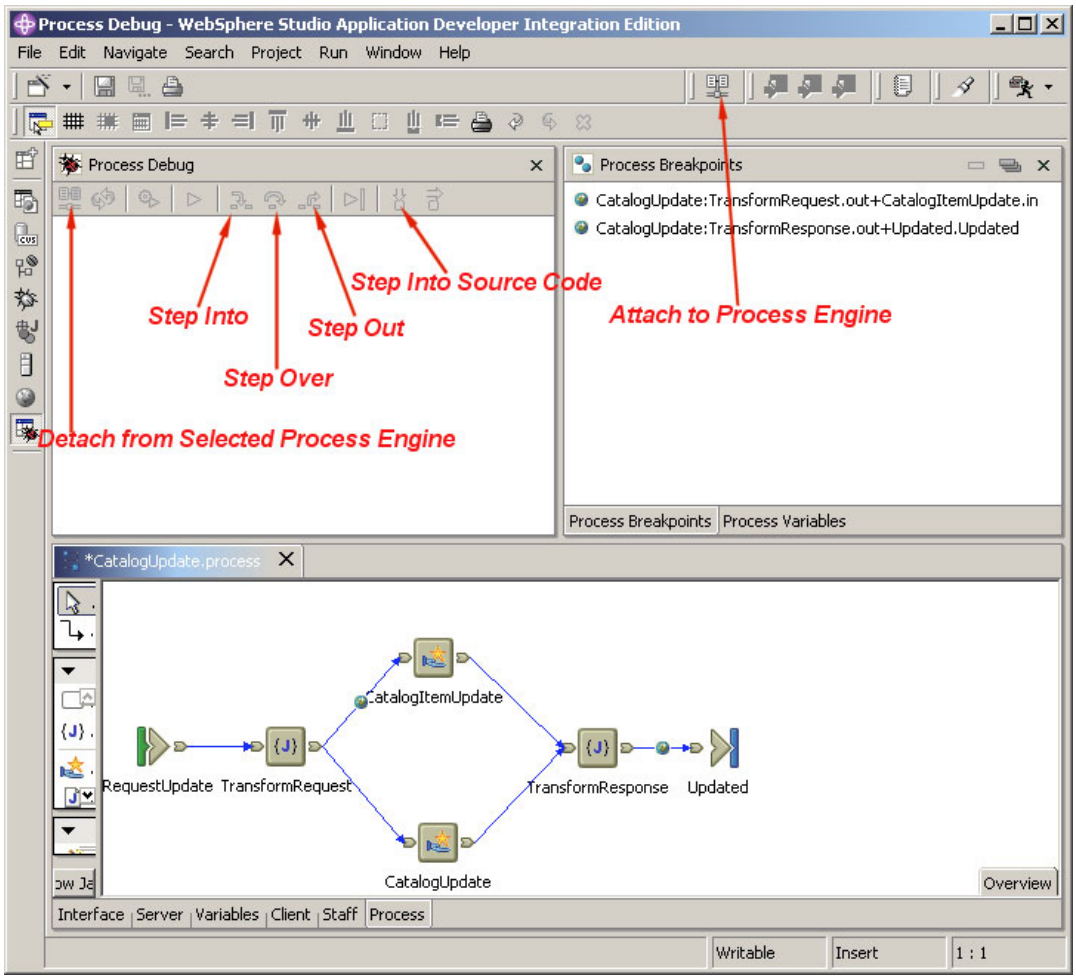


Figure 4-48 Process Debug perspective

4. To attach to the process you want to debug, click the **Attach to Process Engine** icon in the menu bar. On the Attach to Process Engine window, select **localhost** and click **Next**. Select **Server1[xxxx]** and click **Finish**. The default Web client will start.
5. In the Web client browser, select the **CatalogUpdate** Template and click **Start**. The Process Input Message window appears. Enter `1` for the part number and `1.2` for the new price and click the **Start** process button.
6. The Catalog Update process starts and stops at the first breakpoint, as shown in Figure 4-49 on page 152. In this figure, you can see and change the

variables' values. The breakpoint ball in the control link between the TransformRequest and the CatalogItemUpdate now has two yellow arrows:

– The vertical means that you have a Java code in this control line. Now you cannot step through the Java code because you started without Java debug support.

– The horizontal means that you stopped in this point.

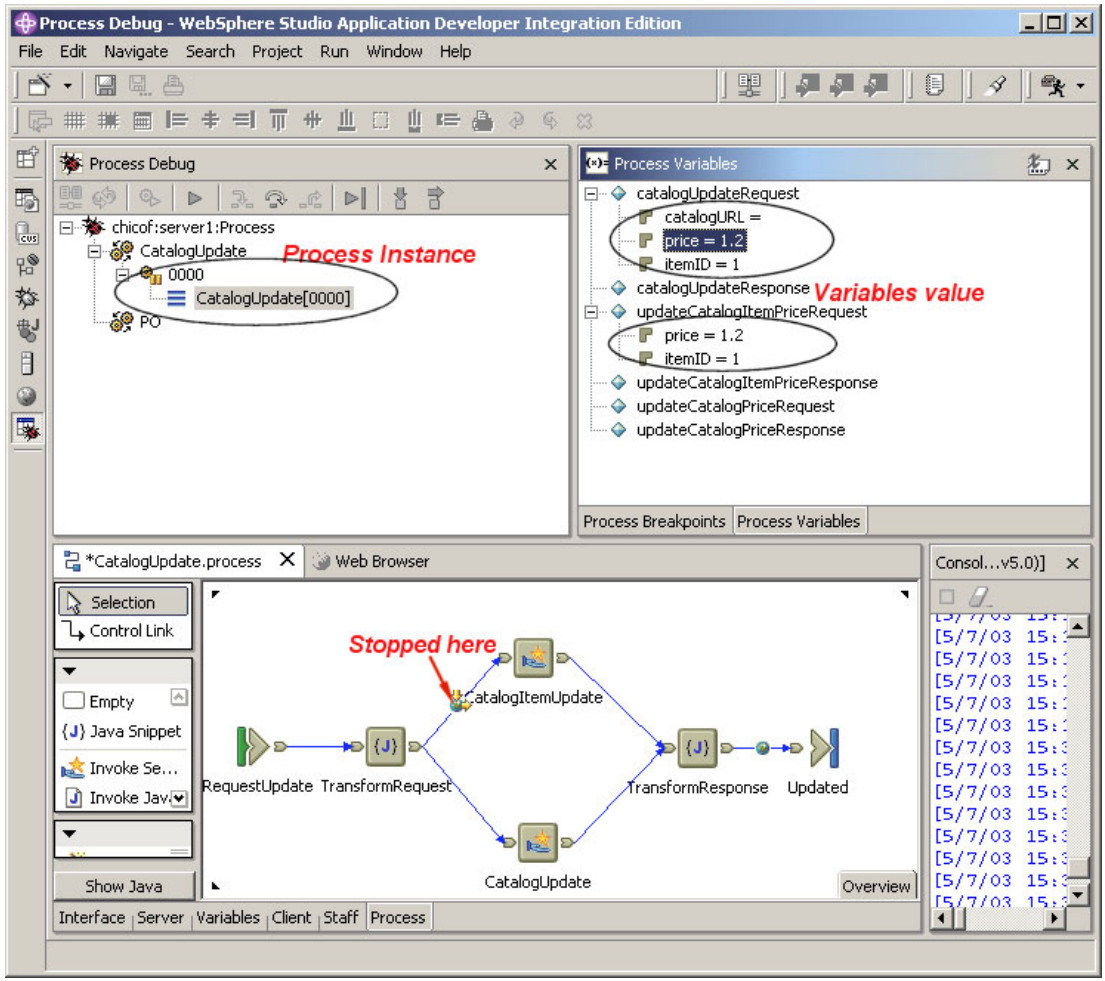


Figure 4-49 Catalog Update process stopped in the breakpoint.

If you click the **Step Over** icon (see Figure 4-48 on page 151), the code of the next block (CatalogItemUpdate) will be executed, and will stop in the next control line, as shown in Figure 4-50 on page 153. You also can see that now we have the value returned from the CatalogItemUpdate output message.

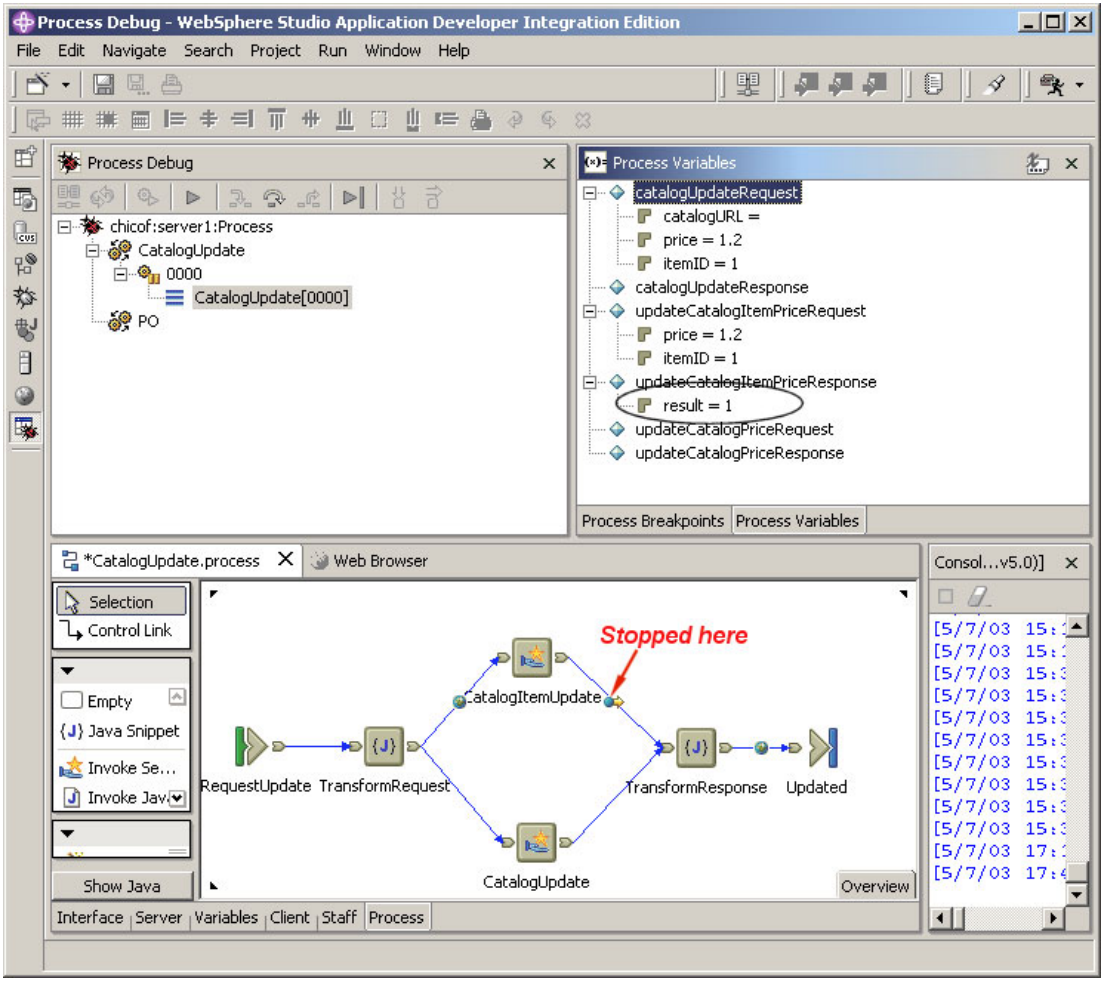


Figure 4-50 Catalog Update process stopped after the Step Over

If you want only to stop in the breakpoints, click the **Resume Process Execution** icon instead of the Step Over.

7. After you finish the debug end the process debug, click the **Detach from Selected Process Engine** icon. Stop the EE Test server in a server perspective.

Running the process debugger with the Java Debugger

1. First let's add a Java code breakpoint in the catalog update process. Open the **CatalogUpdate.process** in a Business Integration perspective, select the **Process** tab, select the **TransformRequest** Java snippet, and click **Show Java**.

2. In the code pane, double-click in the right gray bar in front of the line you want to add the breakpoint. A small blue ball appears (see Figure 4-51).

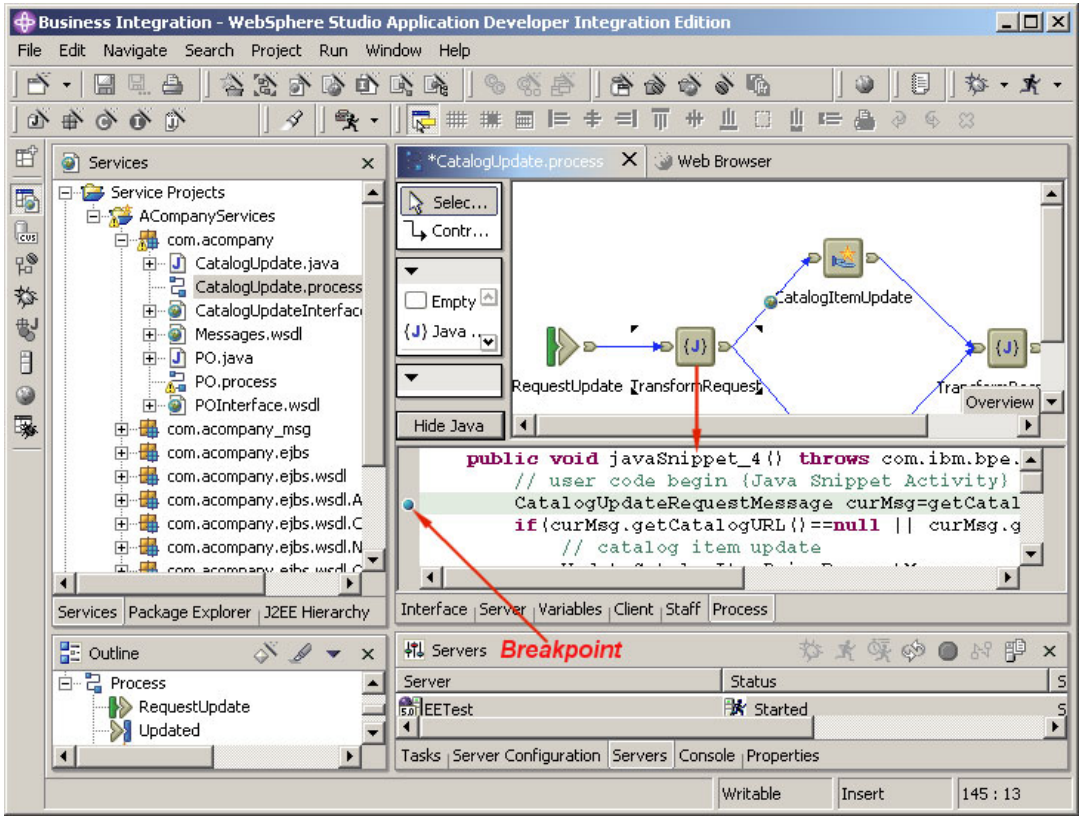


Figure 4-51 Java breakpoint in the TransformRequest snippet of Catalog Update process

3. Now let's start the EE Test server in debug mode. Select the **Servers** tab in a Server perspective, right-click the **EETest server**, and select **Debug**. If the Step by Step Debug window appears (see Figure 4-52 on page 155), check the **Disable step-by-step** mode checkbox, and click **OK**.

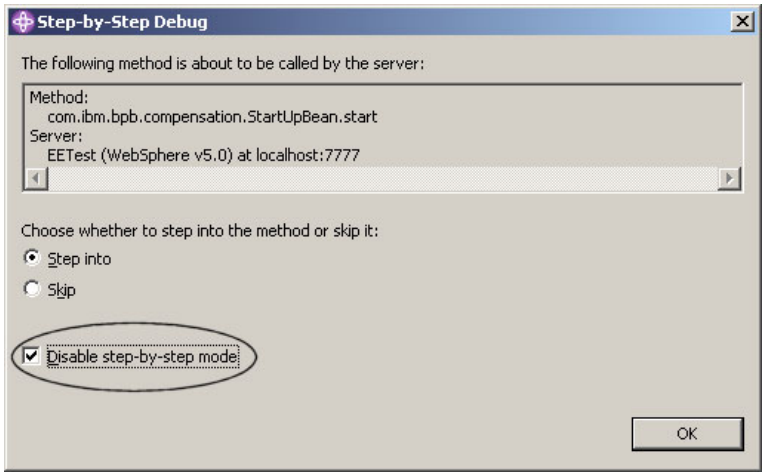


Figure 4-52 Step by step debug

4. On the Debugger Source Lookup window (see Figure 4-53), click **OK**.

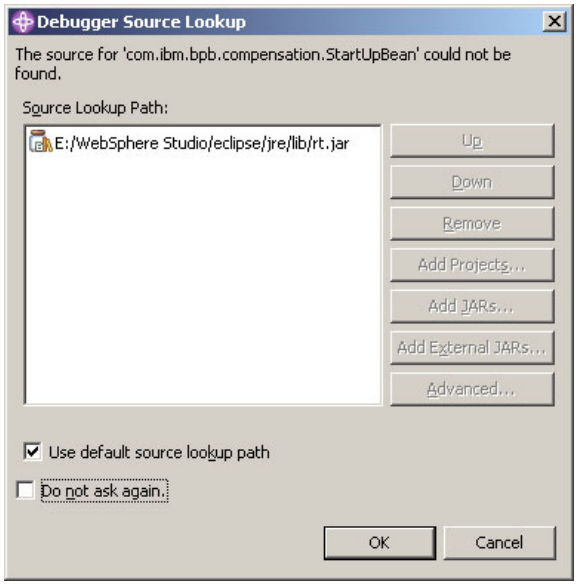


Figure 4-53 Debugger Source Lookup window

5. Now you should be in a Java Debug perspective, select the **Debug** tab in the Servers pane. Select the **Suspended** thread and click the **Resume** icon (see Figure 4-54 on page 156).

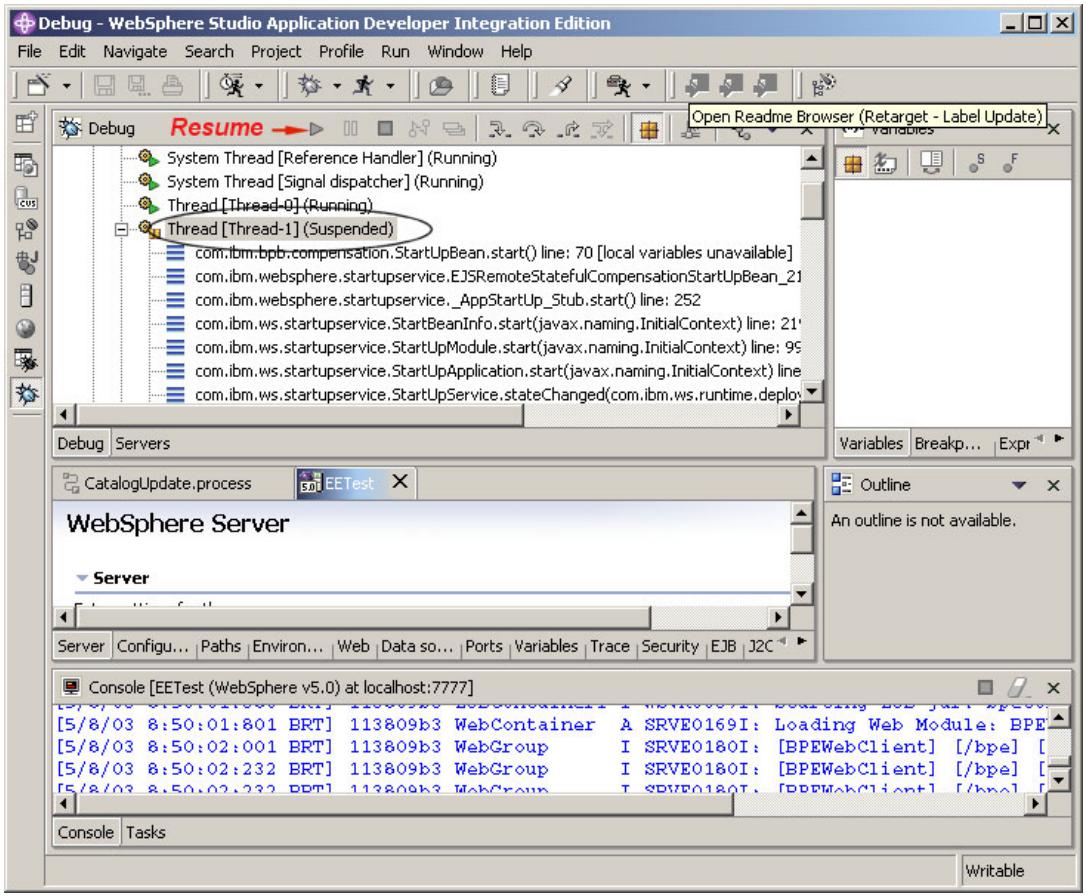


Figure 4-54 Java Debug perspective resume the suspended thread.

6. Wait until the following message appears in the Console tab of Server perspective:

   ```
   WSVR0001I: Server server1 open for e-business
   ```

   Open the **Process Debug** perspective. As you can see in Figure 4-48 on page 151, in the Process Debug perspective you have buttons to attach to the process, step through the process flow, and at the end detach from the process.

7. To attach to the process you want to debug, click the **Attach to Process Engine** icon in the menu bar. On the Attach to Process Engine window, select **localhost** and click **Next**. Select **Server1[xxxx]** and click **Finish**.

8. The default Web client starts. In the Web client browser, select the **CatalogUpdate** template and click **Start**. The Process Input Message

window appears. Enter `1` for the part number and `1.2` for the new price and click **Start process**.

9. The Catalog Update process starts, and stops at the first breakpoint, which now is a Java breakpoint (see Figure 4-55). You can step to this Java code (javaSnippet_4 method) and look at or change the Java variables. When you finish with Java debugging, you can click the **Resume** icon to proceed to the next breakpoint. The perspective will change to the Process Debug (see Figure 4-49 on page 152), and from now on you can step through the Process Debug, which was covered in “Running only the process debugger” on page 149, but now interacting with the Java Debug perspective.

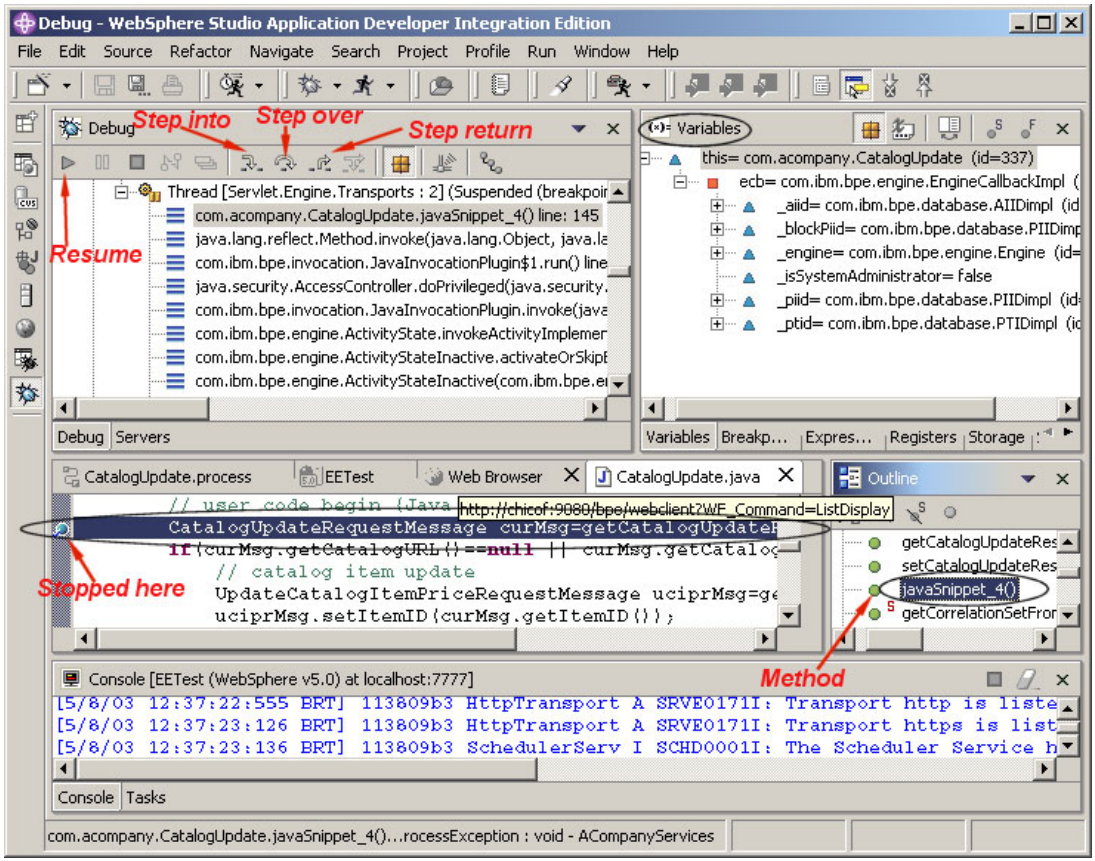


Figure 4-55 Stopped at Java breakpoint

Process debugger in the WebSphere Application Server

The procedure to run the process debugger with WebSphere Application Server Enterprise is almost the same as the EE Test Environment. The only differences are:

1. Deploy the sample application in the WebSphere Application Server Enterprise. See “Development environment” on page 667.
2. Configure the WebSphere Application Server Enterprise:
3. Open an Administrative Console in a Web browser:

   ```
   http://localhost:9090/admin
   ```

4. Select **Servers -> Application Servers**.
5. Select **server1** and under Additional Properties, select **Debugging Service**.
6. In the Configuration tab, select the **Startup** box, which will cause the service to be started in debug mode. Click **OK.**
7. Under Additional Properties, select **Transaction Service.** Change the Total transaction lifetime timeout to 0, to disable the transaction timeouts. Change the Client inactivity timeout to 0, and click **OK**.
8. Save the configuration for WebSphere.
9. Log out of the Administrative Console, and restart WebSphere Application Server Enterprise.
10. Open the Process Debug perspective, shown in Figure 4-48 on page 151.
11. Attach to the process you want to debug by clicking the **Attach to Process Engine** icon in the menu bar. On the Attach to Process Engine window, select **localhost** (if the Application Server is running in the same system) and click **Next**, select **Server1[xxxx]**, and click **Finish**. From now on you can follow the instructions provided in “Running only the process debugger” on page 149 to step through the process flow.

4.5 Staff support

In reality, most business processes cannot be fully automated and therefore require some kind of human interaction. For example:

- A Bank Manager must approve certain types of account transactions.
- A Customer must specify additional data to complete an order.

Human interaction in business processes is handled by staff activities. Any number of staff activities can be added at any point within a business process. Staff activities have various properties, including:

- *Role-based permissions*, which specify the users who can interact with the activity
- *Input data,* which is read-only information provided to users
- *Output data* that can be modified by a user
- A *duration* that indicates the amount of time before the staff activity expires.

When a running process instance encounters a staff activity, process execution is suspended, and a *work item* is created. Work items can be viewed, updated, and *claimed* by users, according to the role-based permissions assigned to the staff activity that generated the work item. Claimed work items can be *completed*, at which time the process instance execution is resumed.

The WebSphere process engine contains a component known as the *Work Item Manager*, which controls the generation of work items and provides the interfaces used to authorize, access, and claim work items.

Users can interact with the Work Item Manager either via the built-in business process Web client, or through a custom interface using the APIs provided. The business process Web client is a customizable Web application that allows users to view, edit, claim, and complete work items.

Staff activity roles and staff queries

There are three different roles associated with staff activities. Table 4-15 shows these roles and the actions permitted by members of these roles.

Table 4-15 Staff Activity Roles:

Role	Actions Permitted
Reader	May view work item
Editor	May view and update work item
Potential Owner	May view, update, claim, and complete work item

Note: Staff activity roles, which apply to a particular staff activity, should not be confused with process roles, which apply to the whole process. See “Process-level staff roles” on page 162 for a discussion of process roles and the actions permitted by process role members.

Users are assigned to staff activity roles indirectly, through the use of *staff queries*. A staff query is a generic way of specifying users, which is independent from a user registry until deploy time, and which provides much more functionality than the standard J2EE security role mapping. For example, staff queries provide the means to grant *potential owner* authority to the manager of a

user who claimed a previous staff activity. The staff query architecture also provides developers with the ability to write custom queries, should they need to do so. Additionally, the staff runtime can be configured to use a different user registry than that which is used for WebSphere itself. See 5.4, “Staff plug-in provider configuration” on page 208 for more information on configuring staff plug-in providers.

Staff queries are specified by application developers using WebSphere Studio Application Developer Integrated Edition. They are composed of a *query verb* and associated parameters. A staff query verb is an abstract query template, which is used at deploy time to define concrete staff queries against a user registry. Staff query parameters are the values assigned to these verbs.

The queries associated with each staff activity in a process are stored in the process’ FDML (flow description markup language) file, along with information about the other process activities and the connections that link them together. The format of the staff queries (query verbs and their assigned parameters) in the FDML file is known as the *parameterized verb* format. At deploy time, the parameterized verbs are translated into XML, which is specific to the staff resolution plug-in being used by the process template. Each staff resolution plug-in has its own transformation XSL file for this purpose. See 5.4, “Staff plug-in provider configuration” on page 208 for more information on staff plug-ins.

The staff query verbs available by default are shown in Table 4-16.

Table 4-16 Default staff query verb set

Verb	**Parameter Values**
Users	User name
Users by User ID	User ID of user
Group Members	Group name
Department Members	Department name
Role Members	Role name
Manager of Employee	User name of employee (not Manager)
Manager of Employee by user ID	User ID of employee (not Manager)
Person Search	Search for a person by specific attributes
Group Search	Search for a group by specific attributes
Native Search	Search with a specific search string
Everybody	All users

Verb	Parameter Values
Nobody	No Users

Important: Not all query verbs are supported by each staff plug-in. The actual set of verbs available at runtime is therefore dependent on the staff plug-in that is configured at deploy time.

In addition to specifying staff query parameters as hard-coded string values, they can also be specified using *late binding values*. These allow for very flexible and dynamic queries, based on parameters that are deferred until runtime.

Table 4-17 Late binding values

Value	Description
`%wf:process.starter%`	User who started the current instance
`%wf:process.readers%`	User(s) with the permission to read instances
`%wf:process.administrators%`	User(s) with permission to administer instances
`%wf:activity(`*`activity_name`*`).potentialOwners%`	User(s) who could claim a previous staff activity
`%wf:activity(`*`activity_name`*`).owner%`	User who claimed a previous staff activity
`%wf:activity(`*`activity_name`*`).editors%`	User(s) who could edit a previous staff activity
`%wf:activity(`*`activity_name`*`).readers%`	User(s) who could read a previous staff activity

The following example takes you through the steps required to assign a query to a staff activity. In this example, we will grant potential owner authority to the manager of the user who claimed a previous activity, called “Approval”.

1. Right-click the staff activity, and select **Properties**.
2. Click **Staff** on the left-hand side to view the staff properties.
3. Select the **Potential Owner** from the list of staff roles, and click **Change**.
4. Select **Manager of Employee by user ID** from the list of **Verbs**.
5. Enter `%wf:activity(Approval).owner%` in the **Value** box beside **EmployeeUserID.** Leave the **Domain** field blank as shown in Figure 4-56 on page 162, and click **OK**.

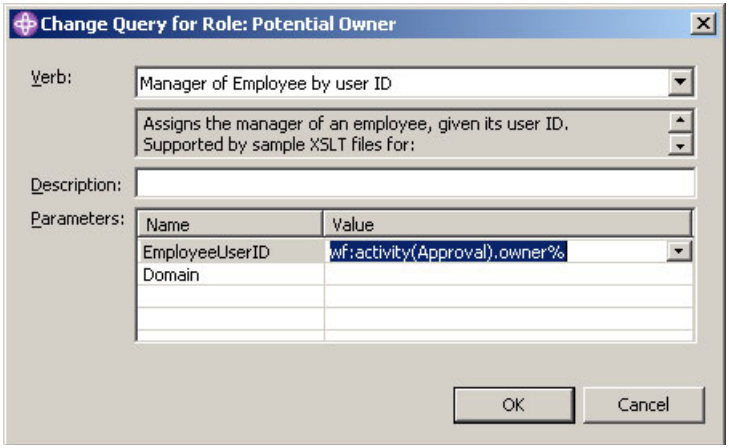


Figure 4-56 Staff activity query configuration

6. Click **OK** to close the staff activity properties window.

Process-level staff roles

In contrast to the staff activity roles, which authorize users who may interact with individual staff activities, process-level staff roles authorize users who may interact with the process as a whole. The process-level roles are shown in Table 4-18.

Table 4-18 Process-level staff roles

Role	Actions Permitted
Administrator	May view, terminate, and delete instances; May edit and complete work items
Reader	May view instances
Starter	May start process instances

Assignment of staff queries to these roles is optional. If the Administrator role is unassigned, the user ID that started the process is granted administrative authority over the process instance. This is the same behavior as one would see if the late binding value `%wf:process.starter%` were assigned to the Administrator role.

Note: In most cases, it is recommended that a group of administrators be assigned to the process Administrator role, in order to prevent administrative authority to be automatically inherited by the process starter.

If the Starter role is unassigned, then any authenticated user is allowed to start a new process instance.

Assignment of staff queries to process-level staff roles is done in WebSphere Studio Application Developer Integrated Edition as follows:

1. In the process editor, click the **Staff** tab.
2. Select one of the roles, and click **Change**.
3. Choose a query verb and assign parameters as desired, then click **OK**.

Creating custom staff query verbs

When the default set of Staff query verbs is insufficient, developers can modify existing verbs and/or add new verbs by editing the VerbSet.xml file, located at <install_dir>\runtimes\ee_v5\ProcessChoreographer\Staff\VerbSet.xml, where install_dir is the directory where WebSphere Studio IE was installed.

When a new staff query verb is created, the transformation XSL file associated with the staff plug-in used by the process must also be modified on the application server.

Note: The VerbSet.xml file can also be found on the WebSphere Application Server Enterprise, at <install_dir>\ProcessChoreographer\Staff\VerbSet.xml. However, this file is only used at development time. It is therefore not necessary to update the VerbSet.xml file on the server when editing the file in the development environment.

Note: Section 2.2 of the *WebSphere Application Server Enterprise Process Choreographer Staff Resolution Parameter Reference* provides a complete reference on the syntax of staff query verbs in the VerbSet.xml file. Appendix A.1 of that document contains the staff query verb XML schema.

The remainder of this section details the steps necessary to create a new staff query verb, called “Manager of Employee by Email Address”, and to modify the LDAP staff plug-in XSL file to support this new verb. This section will only address the aspects of the staff plug-ins necessary to complete this task. A full discussion of staff plug-ins can be found in 5.4, “Staff plug-in provider configuration” on page 208.

In the VerbSet.xml file, verbs consist of a name used to identify the verb, a description, which is displayed when the verb is selected in the Change Query For Role window shown in Figure 4-56 on page 162, and a set of mandatory and optional parameters that are specified when using the verb. Parameter

descriptions consist of a name, a variable type, and a hint, which appears in the drop-down box in the value column of the Change Query for Role window.

The new staff query verb, “Manager of Employee by Email Address”, is very similar to the existing verb “Manager of Employee by user ID”, so we can use that one as a model.

Example 4-23 shows the verb definition for the Manager of Employee by Email Address. This should be added to the VerbSet.xml used by WebSphere Studio IE.

Example 4-23 Staff query verb definition in VerbSet.xml

```
<vs:DefineVerb name="Manager of Employee by Email Address">
   <vs:Description>Assigns the manager of an employee, given its Email
address. Supported by sample XSLT files for: - LDAP</vs:Description>
   <vs:Mandatory>
      <vs:Parameter>
         <vs:Name>EmployeeEmailAddress</vs:Name>
         <vs:Type>xsd:string</vs:Type>
      </vs:Parameter>
   </vs:Mandatory>
   <vs:Optional>
      <vs:Parameter>
         <vs:Name>Domain</vs:Name>
         <vs:Type>xsd:string</vs:Type>
      </vs:Parameter>
   </vs:Optional>
</vs:DefineVerb>
```

As you can see, verb definitions are quite simple. The magic is in how the verbs are used in conjunction with the staff plug-ins. Since our new query verb relies on organizational information (that is, the manager of the employee), the LDAP plug-in is the only plug-in which will support this verb. Accordingly, the LDAPTransformation.xsl is the only plug-in XSL file which we need to modify.

The plug-in XSL files can be found on the WebSphere Application Server Enterprise at <install_dir>\ProcessChoreographer\Staff. Here are the steps necessary to add support for “Manager of Employee by Email Address” to LDAPTransformation.xsl:

1. Open <install_dir>\ProcessChoreographer\Staff\LDAPTransformation.xsl for editing.
2. The following line should be added to the Global Variables section at the top. If the “email” attribute in your LDAP schema is something other than *email*, then modify this variable accordingly.

   ```
   <xsl:variable name="DefaultUserEmailAttribute">email</xsl:variable>
   ```

3. In the section under Global Dispatching, add the following “when” clause to the <xsl:choose> element:

```
<xsl:when test=”$verb=’Manager of Employee by Email Address’”>
   <xsl:call-template name=”Manager of Employee by Email Address”/>
</xsl:when>
```

4. At the bottom of the file, add the following template element:

```
<!-- Begin template Manager of Employee by Email Address-->
   <xsl:template name="Manager of Employee by Email Address">
      <sldap:staffQueries>
         <xsl:attribute name="threshold">
            <xsl:value-of select="$Threshold"/>
         </xsl:attribute>

      <sldap:intermediateResult>
         <xsl:attribute name="name">manager</xsl:attribute>
         <sldap:search>
            <xsl:attribute name="filter">
               <xsl:value-of
select="$DefaultUserEmailAttribute"/>=<xsl:value-of
select="staff:parameter[@id='EmployeeEmailAddress']"/>
            </xsl:attribute>
            <xsl:attribute name="searchScope">objectScope</xsl:attribute>
            <xsl:attribute name="recursive">no</xsl:attribute>
            <sldap:attribute>
               <xsl:attribute name="name">
                  <xsl:value-of select="$DefaultManagerAttribute"/>
               </xsl:attribute>
               <xsl:attribute name="objectclass">
                  <xsl:value-of select="$DefaultPersonClass"/>
               </xsl:attribute>
               <xsl:attribute name="usage">simple</xsl:attribute>
            </sldap:attribute>
         </sldap:search>
      </sldap:intermediateResult>

      <sldap:user>
         <xsl:attribute name="dn">%manager%</xsl:attribute>
         <xsl:attribute name="attribute">
            <xsl:value-of select="$DefaultUserIDAttribute"/>
         </xsl:attribute>
         <xsl:attribute name="objectclass">
            <xsl:value-of select="$DefaultPersonClass"/>
         </xsl:attribute>
      </sldap:user>
   </sldap:staffQueries>
</xsl:template>
<!-- End template Manager of Employee by Email Address-->
```

5. Save and close the file.

At this point, the new query verb should be available for use within WebSphere Studio Application Developer Integrated Edition, and should be usable on any WebSphere Application Server Enterprise that has an LDAP staff plug-in that uses the updated LDAPTransformation.xsl file.

Staff activity data

Like most other business process activities, staff activities have an input and output terminals that can be assigned process variables. Variables assigned to a staff activity's input terminal are read-only, and are viewable by users who can read, edit, or claim the corresponding work item. This input data can be used to provide information to the users that may be needed to complete the business process. Variables assigned to a staff activity's output terminal can be modified by users who can edit or claim the corresponding work item.

Note: In addition to providing informational data using the staff activity's input terminal, the business process Web client can be customized to present virtually any type of data that users might need in order to complete the work item.

Staff activity duration

Whenever a business process requires human intervention, it is always a good idea to design the process so that it can recover in the event that a work item is never claimed by a user. In the case of a process that is waiting for approval from a user, it might be appropriate to cancel the originating request after some reasonable delay. Or, it might be required that a backup user be asked to approve the request. To facilitate these types of scenarios, staff activities can have a limited duration. If a work item is not claimed before the corresponding activity expires, the process will resume. Using the Business Process Choreographer API, it is possible to determine whether a staff activity was completed successfully, or whether it expired.

The format of the string used to specify a staff activity duration is dependent on the Scheduler Calendar JNDI name that is (optionally) specified when the Business Process Container is installed. This is the JNDI name of a UserCalendar session bean. For example, com/ibm/websphere/scheduler/calendar/DefaultUserCalendarHome is the JNDI name of the default UserCalendar session bean included in the com.ibm.websphere.scheduler package. If no Scheduler Calendar JNDI name is specified during the Business Process Container installation, then only the SIMPLE calendar format is available. If a UserCalendar session bean JNDI name is specified, then any calendar format supported by that UserCalendar can be used when setting staff activity durations.

The default UserCalendar (JNDI name com/ibm/websphere/scheduler/calendar/DefaultUserCalendarHome) supports two calendar formats: SIMPLE and CRON. Both of these are well documented in the WebSphere JavaDoc pages. In short, the SIMPLE calendar format describes a time duration in seconds, minutes, hours, days, etc. For example, the string "`30minutes 2hours`" specifies an activity duration of 2.5 hours. On the other hand, the CRON calendar format describes a fixed point in time using a syntax that is similar (yet distinctly different) from that used to schedule jobs on UNIX-like systems. For example, the string "`0 30 1 ? * MON-FRI`" specifies a staff activity duration that will expire at 1:30 am Monday through Friday, regardless of when the activity started.

Note: In order to use any UserCalendar session bean as the Business Process Container's scheduler calendar, an EAR containing the UserCalendar must be installed on the application server. On the WebSphere Application Server Enterprise, the SchedulerCalendars.ear (which contains the default UserCalendar described above) is installed by default. On the unit test server within the WebSphere Studio Application Developer Integrated Edition, Versions 5.0 and 5.0.1, the SchedulerCalendars.ear is not installed by default. When using these versions of WebSphere Studio Application Developer Integrated Edition, the SchedulerCalendars.ear must be imported into the user's workspace and deployed on the unit test server in order to configure a UserCalendar as the Business Process Container's scheduler calendar. Otherwise, only the SIMPLE calendar format will be available within the unit test server.

Important: Staff activities that are Claimed never expire.

To add a duration to a staff activity, do the following:

1. Right-click the staff activity and select **Properties**.
2. Click **Server** on the left-hand side to view the Server properties.
3. In the Duration field, enter a time specification using the SIMPLE Arithmetic Calendar format, as described in the com.ibm.websphere.scheduler.UserCalendar JavaDoc reference. For example, to assign a duration of five hours and 30 minutes to the staff activity, enter 5hours 30minutes as shown in Figure 4-57 on page 168.

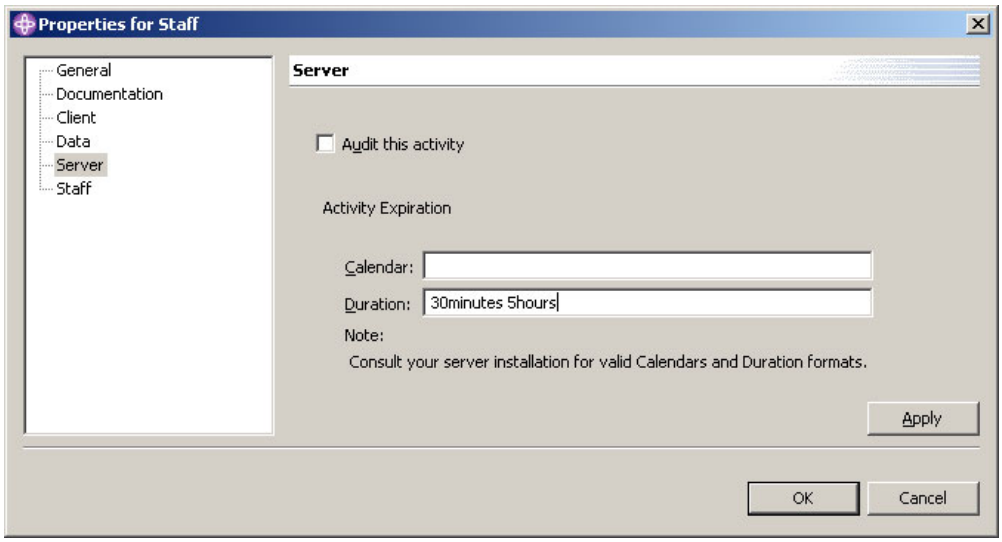


Figure 4-57 Staff activity duration configuration

4. Alternatively, if a Scheduler Calender JNDI name has been specified for your business process container as described above, then other calendar formats may be used. For example, if using the default UserCalendar, then a duration string using the CRON format may be used by specifying CRON in the Calendar field, and a CRON calendar string (for example: `0 30 1 ? * MON-FRI`) in the Duration field.

   Note: When the Calendar field of the Staff Activity Duration window is left blank, the SIMPLE calendar format is assumed. Note also that this calendar format is available even if no Scheduler Calendar is configured for the business process container, and even if the SchedulerCalendars.ear is not installed.

5. Click **OK** to close the staff activity properties window.

When a staff activity with a limited duration expires, the process will resume. In this event, subsequent activities will need a way to know that a staff activity expired, or not. This can be accomplished by using a Java snippet activity to determine the state of the staff activity, and set a boolean process variable accordingly.

The following example shows how this is done using the Business Process API. The name of the staff activity is “Approval”, and the boolean process variable is “approveSuccessful”.

Example 4-24 Using the Business Process API to determine the state of a staff activity

```
try {
   // First, acquire the BusinessProcess object via its EJB Home interface
   Context initialContext = new InitialContext();
   Object result =
      initialContext.lookup("com/ibm/bpe/api/BusinessProcessHome");
   BusinessProcessHome processHome = (BusinessProcessHome)
      javax.rmi.PortableRemoteObject.narrow
      (result,BusinessProcessHome.class);
   BusinessProcess process = processHome.create();
   // Next, acquire the ActivityInstanceData for the “Approval” staff activity
   // using the process instance ID for this instance of the process
   ActivityInstanceData staffActivity =
      process.getActivityInstance(processInstance().getID(),"Approval");
   // Now get the execution state of the staff activity, compare to the
   // constants and set the approveSuccessful process variable accordingly
   int executionState =  staffActivity.getExecutionState();
   BooleanMessage approveSuccessful = getApproveSuccessful();
   if (executionState == staffActivity.STATE_EXPIRED) {
      System.out.println("activity expired");
      approveSuccessful.setValue(false);
   } else if (executionState == staffActivity.STATE_FINISHED) {
      System.out.println("activity finished");
      approveSuccessful.setValue(true);
   }
   setApproveSuccessful(approveSuccessful);
      catch (Exception e) {
   e.printStackTrace();
}
```

4.6 Sample scenario

The following are some considerations during the development and testing of the sample application:

- Every time you change anything that changes the business process (any change in the visual process designing, Java Snippet code, and so on), you need to run the Deploy Process for this change to be reflected in the EE Test Environment.
- If you make changes in the code and try to publish, but a publish error message is received and you are not able to start the EETest server, remove the failing publish project from the EETest server. Expand the EETest server in the Server Configuration pane, right-click the failing project and select **Remove**. Add it again by right-clicking the **EETest** server in the Server Configuration pane, selecting **Add**, and selecting the removed project.

5

Process Choreographer runtime environment

The previous chapter focused on the development steps related to the Process Choreographer function. This second chapter on the Process Choreographer describes the runtime environment in detail.

The following topics are covered in this chapter:

- Process container architecture
- Runtime topologies
- Process container installation steps
- Configuring the staff plug-in provider
- Security considerations
- Managing the business process applications
- Problem determination and troubleshooting

5.1 Process container architecture

WebSphere Application Server Enterprise provides the runtime environment necessary for the programming model described in the previous chapter through the business process engine, which is also known as the business process container or process container. Its architecture is illustrated in Figure 5-1. It is implemented as a J2EE application that utilizes WebSphere Application Server runtime services and resources. The key components of the process container are:

- Process navigation
- People interaction
- Factory
- External interfaces
- Internal interfaces

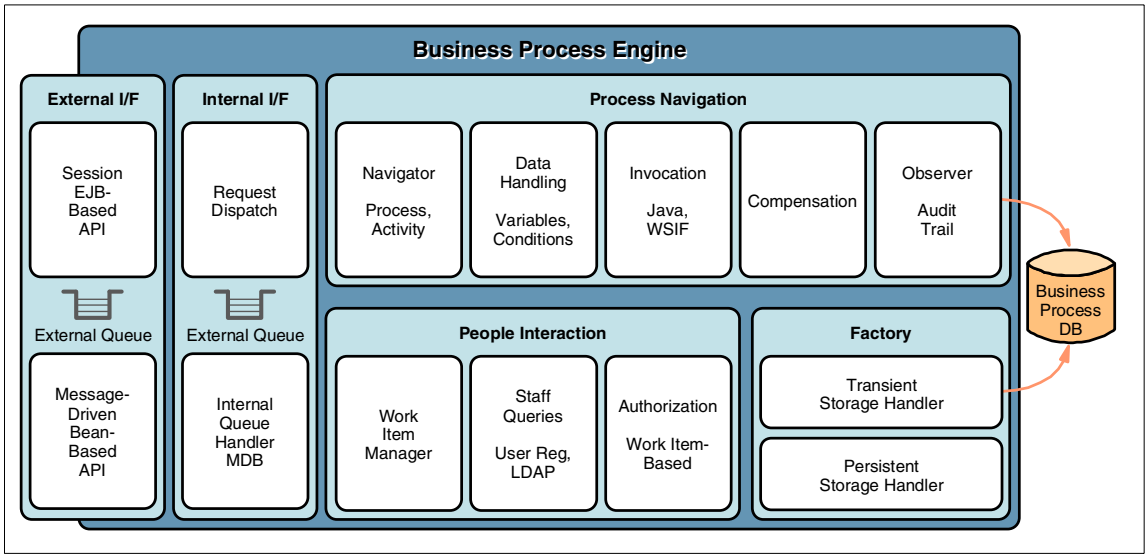


Figure 5-1 Internal architecture of the WebSphere business process engine

For more details on the process container architecture, see the article in the WebSphere Developer Domain at:

http://www7b.boulder.ibm.com/wsdd/library/techarticles/wasid/WPC_Concepts/WPC_Concepts.html

Process navigation

Process navigation is composed of the navigator and some plug-in components.

Navigator

The Navigator component is the heart of the process engine. It manages the state transitions for all process instances, and the state transitions for all activities in those process instances. These are illustrated in Figure 5-2.

The normal life of a process instance begins with a start request. This creates the process instance and puts it into its running state. When all its contained activities have reached an end state, the process instance is marked finished. The process instance ceases to exist, either implicitly or via an explicit API call.

In exceptional cases, the process instance might encounter a fault that was not processed as part of the process logic. In this case, it waits for the completion of the active activities before putting the process into its failed state. Compensation is then invoked if it was defined for the process.

A process instance can also be terminated by a process administrator. In this case, after completion of the active activities, the process instance is put into its terminated state.

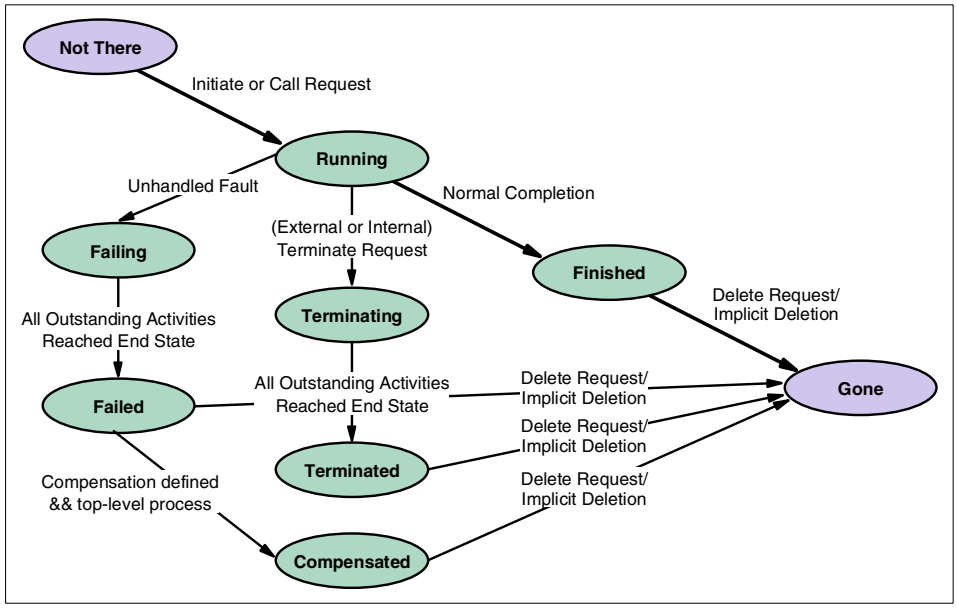


Figure 5-2 State diagram for life cycle of a process

Plug-ins for process navigation

The Navigator delegates some of the tasks it has to perform to plug-ins. These plug-ins decouple the Navigator from components it needs to use, and will allow IBM to easily extend the capabilities of the process engine in the future.

Plug-ins are provided for:

- The invocation of activity implementations.

  The process engine has two plug-ins for this: one for the invocation of (external) services via the Web Services Invocation Framework (WSIF), and one for the invocation of Java snippets.

- The handling of data in the process, such as evaluating conditions.

  The process engine has a plug-in that understands conditions written in Java against WSDL messages.

- The logging of interesting events in an audit trail.

  The process engine has a plug-in that writes data to the audit trail table of the process engine's database.

Factory

The Factory component is responsible for the management of the physical state information the process engine deals with. It allows data to be stored in one of the following forms:

- Transiently in memory, as required by non-interruptible processes for efficient execution
- Persistently in a database, as required by interruptible processes for durability

The supported databases include DB2®, Oracle, Sybase, and Cloudscape™.

Internal interfaces

The process engine uses its internal queue to send JMS messages to itself to process stratified transactions that are needed for interruptible processes. This queue is a JMS queue that allows the persistent queuing of messages.

External interfaces

A façade-session EJB and a façade MDB provide the synchronous and asynchronous renderings of the external interfaces.

People interaction

Note that Process Choreographer supports business processes with people interaction only when the WebSphere Application Server security is enabled. Otherwise, no authentication occurs in WebSphere Application Server, and since the user is unknown, his work items cannot be determined.

The main components involved in people interaction are:

- Web client
- Work item manager

- Staff support service
- Staff resolution plug-in
- Staff repositories

The architecture for these components is shown in Figure 5-3.

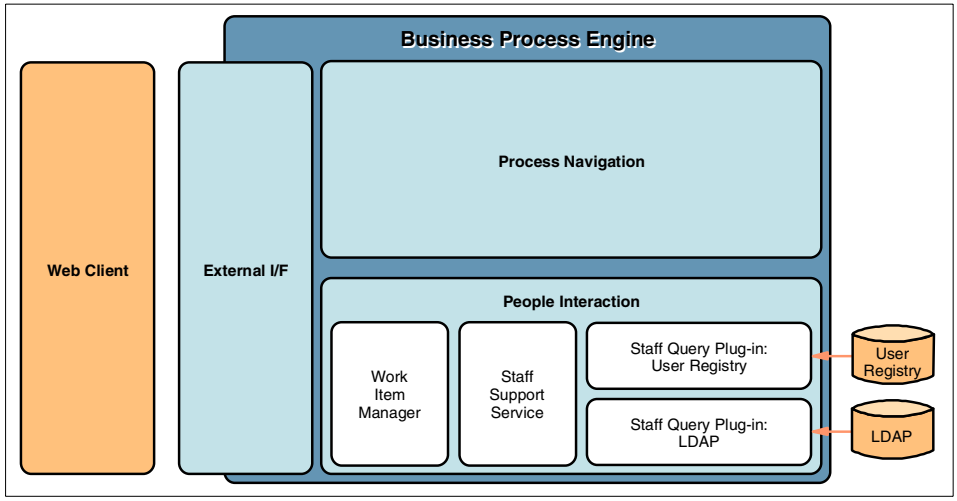


Figure 5-3 Staff Resolution Architecture

Web client

The Web client offers users a Web browser-based graphical user interface, based on JSP (JavaServer Pages) technology. It offers the user a set of worklists containing work items with which he can perform queries, view further details, or perform certain actions. It also allows building JSP-based custom forms that support staff activities with complex data or interaction scenarios. The Web client interacts with the process engine via the API. It does not interact directly with the other components supporting human interaction. The Web client is described in more detail in the document “Understanding the WebSphere Process Choreographer Web client” found at:

`http://www7b.boulder.ibm.com/wsdd/zones/was/wpc.html`

Work item manager

The work item manager (WIM) is a component of the process engine specially dedicated for the management of work items. It is responsible for:

- The creation and deletion of work items in the database.
- The resolution of work item queries from process participants against this database.

- The coordination of the staff query resolution, which is performed by the staff support service and the staff resolution plug-ins.
- The instance-related authorization based on the work items created for users. Users are authorized to perform actions against an activity or process only if they have received a corresponding work item. For example, process participants can claim an activity only if they have a work item of the potential owner type for that activity.

The work item manager uses an internal cache for resolved staff queries, with a default query result expiration time of one hour. Thus, when a staff query result from an identical query template already exists in the cache, the second resolution will not occur and the cached object will be reused. This allows for better performance. Also, when the cached query result expires and the corresponding work item is queried, the staff query gets executed again. This allows for updated query results, when changes occur in the staff repository. In consequence, the WIM does not invoke the staff resolution for all work item scheduling, and may also invoke the staff resolution when currently no work item scheduling occurs.

Staff support service

The staff support service (SSS) acts as a staff resolution plug-in from the WIM point of view. However, it is a master plug-in that manages the life cycle of the specialized staff resolution plug-ins, and delegates the actual deployment and staff resolution requests to these specialized plug-ins.

Staff resolution plug-ins

The specialized staff resolution plug-ins are bound to a specific staff repository such as the User Registry or an LDAP (Lightweight Directory Access Protocol) directory. They are responsible for the deployment and the actual staff query execution, and perform these queries by invoking a set of API calls against their repository. For more details on the staff resolution plug-ins, refer to the document “WebSphere Application Server Enterprise Process Choreographer - Staff Resolution Architecture” found at:

`http://www7b.boulder.ibm.com/wsdd/zones/was/wpc.html`

Staff repositories

Operating system (OS) registries, user registries, or LDAP registries can be used as staff repositories.

5.2 Process container runtime topologies

In order to run business process applications to a WebSphere Application Server, it must have a process container installed. This involves installing the

process container EAR to an application server and configuring all the resources it requires.

Before installation of business process containers and application servers, you should determine a suitable installation topology and required hardware and software components. The topology choice should take into account scalability, load balancing, availability, and failover requirements for your business process application with an understanding of the process container architecture. Other factors to consider are security, ease of administration, and monitoring of business processes.

Scalability and availability

The process container is implemented as a stateless process engine, many instances of which can run in parallel on a single node or distributed in a cluster. It can exploit both the clustering capabilities of WebSphere Application Servers for IIOP requests and the clustering capabilities of WebSphere MQ for JMS-based requests.

For microflows, there is an affinity between a particular process instance and the thread that executes it. Many instances of a particular process template can run in parallel on parallel threads.

For long-running processes, no such thread affinity exists. A given process instance is simply reflected by a set of tuples persisted in the process database. Threads work on segments of process instances. Multiple threads, even on multiple nodes, can work concurrently on parallel branches of a process.

The entire approach relies on the business process database as the central state repository that needs to be always available (and thus provides a single point of failure). If critical, this availability has to be guaranteed by putting the database on a reliable node managed by high availability software.

Sample installation topologies

Some of the common topologies that we may use in the installation of process containers are described here. Instead of presenting an exhaustive list of all topologies, we concentrate on how a process container and its related resources should be configured for some typical topologies. We do not touch on other topics relating to topologies, such as separation of Web tier and EJB tier, using a load balancer (Edge components) to distribute client HTTP requests to each HTTP server, introduction of firewalls, or DMZ. For more details on these topics, refer to *WebSphere Application Server V5 Handbook* and *WebSphere Application Server Network Deployment V5, Getting Started*.

In the sample topologies, the notations listed in Table 5-1 on page 178 are used to indicate some required software components.

Table 5-1 Notations for software components in the topology samples

Notation	Software component
WMQ	WebSphere MQ Version 5.3.01
WAS	WebSphere Application Server (version 5.0 is implied)
Plug-in	Web Server plug-in for WebSphere Application Server Version 5.0
BPE	Business Process Engine or process container
database	database system (DB2, Oracle or Sybase)

The following sample topologies will be discussed:

1. Stand-alone application server on a single machine (later referred to as SA topology).
2. Multiple application servers in Network Deployment without clustering (later referred to as ND-noC topology).
3. Application server clusters in Network Deployment (later referred to as ND-C topology).
   a. Vertical scaling (later referred to as ND-VS topology).
   b. Horizontal scaling (later referred to as ND-HS topology).
4. Application server cluster with WebSphere MQ Clustering (later referred to as MQ-C topology).
5. Application server cluster with high-availability configuration of database and central queue manager machines (later referred to as DB-C&MQ-C topology).

5.2.1 Stand-alone application server on a single machine (SA)

In this topology, we have WebSphere Application Server Enterprise together with the HTTP server and Web server plug-in installed on a single machine.

A database system of choice (DB2, Oracle, Sybase, or Cloudscape) is installed on the same machine. We have one process database and a corresponding JDBC data source to connect to this database.

JMS provider can be either embedded WebSphere JMS (Embedded Messaging) or WebSphere MQ. If WebSphere MQ is the JMS provider, it is installed locally. A single WebSphere MQ queue manager and four required queues are created locally. The two JMS connection factories are mapped to this queue manager.

- Simple installation that requires only one machine.
- No load balancing or failover.
- Suitable for development and test environments.

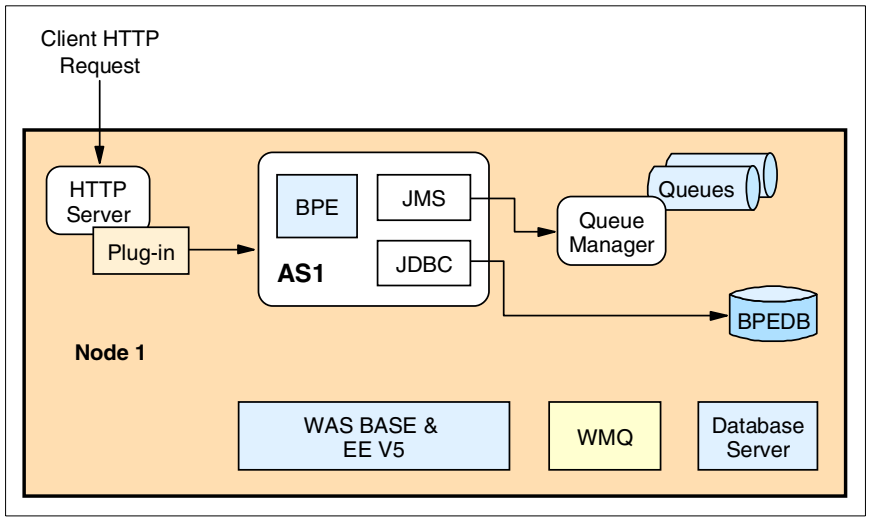


Figure 5-4 Topology 1: single application server instance, single machine

5.2.2 Application servers network deployed with no clustering (ND-noC)

In this topology, WebSphere Application Server for Network Deployment, Administrative Console extensions for WebSphere Application Server Enterprise are installed on one machine. An HTTP server and plug-in are also located on this machine. A database client software for the database system of choice is installed on this machine.

WebSphere Application Server Enterprise is installed on Node1 and Node2.

Application server AS1 on Node1 uses a remote process database. Hence a database system of choice (DB2, Oracle, Sybase, or Cloudscape) is installed on a dedicated node. A corresponding database client is installed on Node1.

Application server AS2 on Node2 uses a local process database. Hence a database system of choice with one local process database is installed on Node2.

Each application server node has a WebSphere JMS (Embedded Messaging) server that can be used as the JMS provider.

If WebSphere MQ is selected as the JMS provider, it is installed on the same machine.

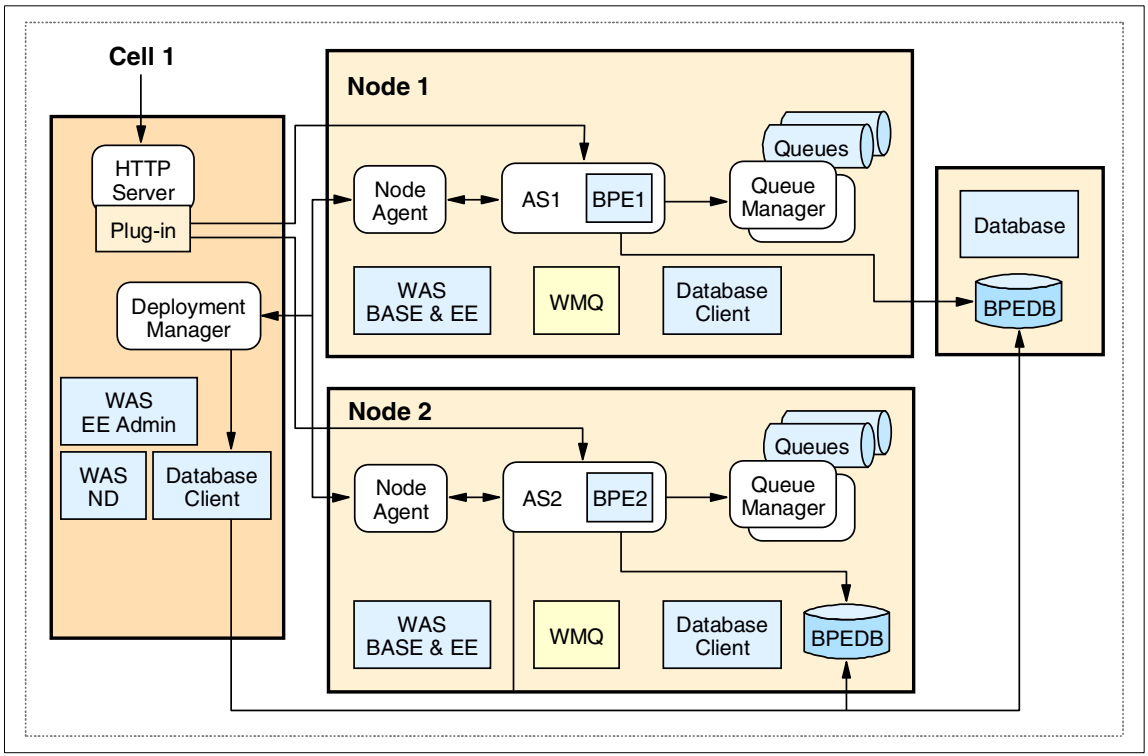


Figure 5-5 Topology 2: Network Deployment of multiple server instances on a node without clustering

In this topology, the deployment manager provides a single point of administration for all nodes and server instances in the cell.

- When configuring a JDBC data source for an application server in the cell, each data source must use its own unique database, that is BPEDB1 and BPEDB2.
- Each application server instance has JMS resources configured to connect to its own local queue manager and queues.
- No load balancing or failover.
- Suitable for administration of development and test environments for different applications hosted on different application servers.

5.2.3 Application server clusters in Network Deployment (ND-C)

In Network Deployment, you can configure identical application server instances into a cluster to share the workload. Clusters enable workload management

between member servers. As your workload increases, you can add member server instances to a cluster.

Process containers in a WebSphere Application Server cluster must share the same process database and hence the same data source. Process containers in a cluster must also share the same JMS resources. If your business process has human activities, the containers in a cluster must also share the same staff plug-in provider configurations. Global security must be enabled for the cell containing the cluster.

There are two ways to form a server cluster: vertical scaling and horizontal scaling.

Vertical scaling variant (ND-VS)

You can add multiple application server instances on the same machine or node.

- It allows increased utilization if CPU and memory on each application server node.
- It provides failover in case of application server process failure. If an application server process fails, client requests can be redirected to remaining cluster members on the node.
- The failure of the host machine or node itself presents a single point of failure.

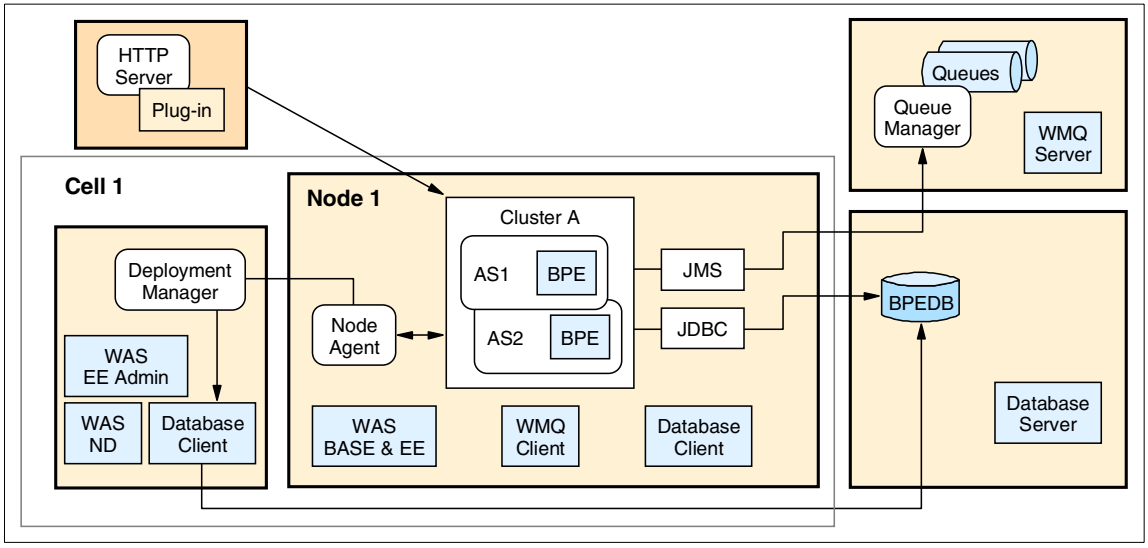


Figure 5-6 Topology 4, WebSphere clustering in Network Deployment, vertical scaling

Horizontal scaling variant (ND-HS)

You can add machines or nodes, each with an identical application server instance to a cluster.

- It provides increased throughput by utilizing CPU and memory of additional machines.
- Hosting cluster members on multiple nodes isolates hardware failures of a node and provides failover support. Client requests can be redirected to the application server members on other nodes if a node goes down.
- Hosting cluster members on multiple nodes also isolates application software failures and provides failover support. If an application server process stops running, client requests can be redirected to cluster members on other nodes.

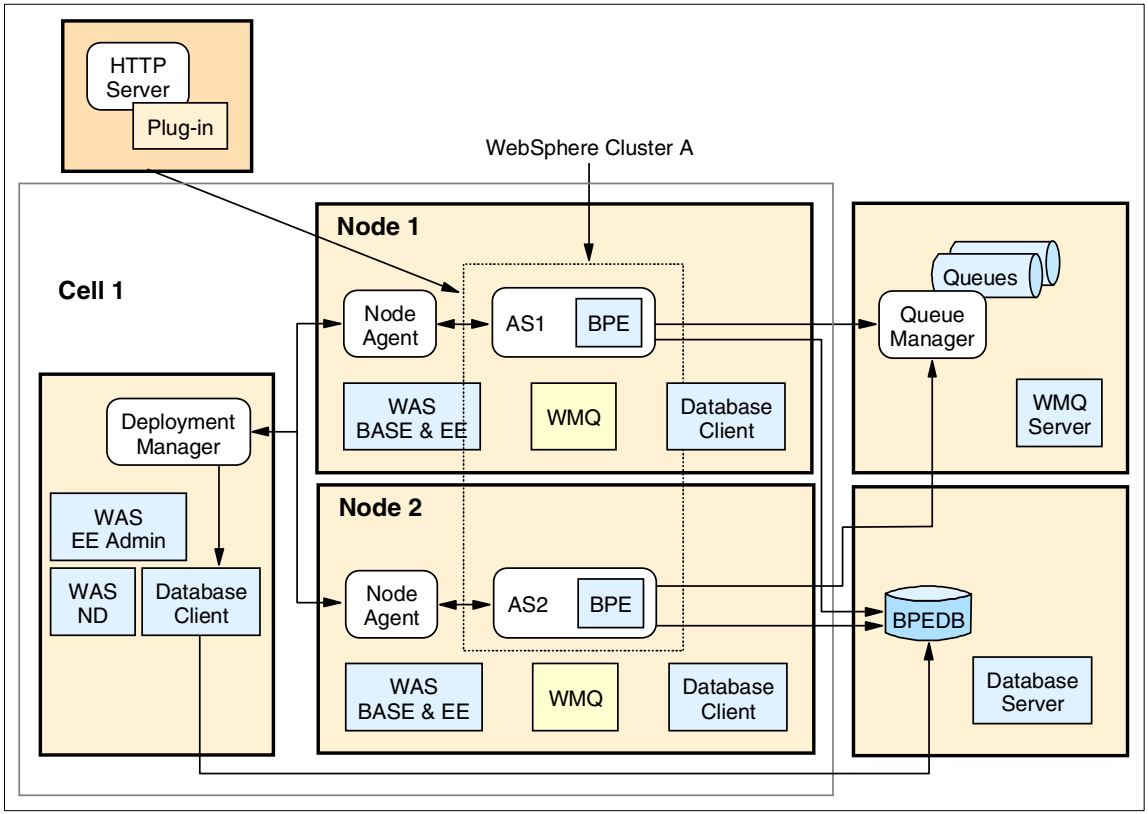


Figure 5-7 Topology 5, WebSphere clustering in Network Deployment horizontal scaling

In both topologies with vertical or horizontal scaling:

- Database system of choice is installed on a dedicated server machine.

- WebSphere MQ is installed on a dedicated server machine. The required queue manager and queues are located on this machine.
- Either WebSphere MQ client or Embedded Messaging JMS client is installed on application server nodes to access the queue manager on the WebSphere MQ server node.
- For application servers in Network Deployment, there is one internal JMS server per application server node. You may select WebSphere JMS (Embedded Messaging) as the JMS provider for the cluster. In this case, one internal JMS server should be configured on a dedicated node with the required queue manager and queues. This queue manager can be accessed remotely by all application servers in the cluster through embedded JMS client.
- The nodes for the database system, WebSphere MQ, and deployment manager present single points of failure.

Note: You may install your database system of choice and WebSphere MQ on separate disk volumes and file systems on the same dedicated machine.

5.2.4 Application server cluster with WebSphere MQ Clustering (MQ-C)

In addition to WebSphere Application Server clustering, intra-process load balancing can be achieved by using clustered queue managers. This requires full installation of WebSphere MQ instead of the embedded WebSphere JMS provider. WebSphere MQ and WebSphere MQ Event Broker 5.3 are included in the WebSphere Application Server Enterprise V5 package. The license is limited to the use of WebSphere MQ on the same machine (node) as the WebSphere Application Server.

To use WebSphere MQ queue manager clusters, you need two local queue managers for each application server node: one to put messages that map to connection factory jms/BPECFC, and one to get messages for mapping jms/BPECF. The get queue manager owns the four queues that map to four JMS destinations. We need both connections to two queue managers at the same time, because put operations and get operation happen in the same transaction.

Restriction: On UNIX® systems, WebSphere MQ does not allow two connections with BIND transport to two different queue managers at the same time. Hence, one of the queue managers on a node needs to be configured with CLIENT transport.

These two local queue managers on each node are federated into a WebSphere MQ queue manager cluster. This provide extra intra-process load balancing between queue managers. Note that you need three queue manager nodes for load balancing. For more details on setting up a WebSphere MQ queue manager cluster for a process container, refer to the article in WebSphere Developer Domain “WebSphere Application Server Enterprise Process Choreographer using Process Choreographer in a distributed environment”, found at:

http://www7b.boulder.ibm.com/wsdd/library/techarticles/wasid/WPC_UsingDist/WPC_UsingDist.html

With WebSphere MQ queue manager clustering, if a request message reaches its destination queue manager, or it is en route to the destination queue on the channel, and the destination queue manager fails, the system will not be able to recover the message until the destination queue manager itself recovers. When a message reaches a queue, it remains in this queue, even if it is a cluster queue. If the node with this queue crashes, the message is not distributed to other queues in the cluster. The message cannot be processed before the system comes up again. Hence, WebSphere MQ queue manager clusters provide intra-process load balancing but do not provide failover.

Figure 5-8 on page 185 illustrates one possible topology.

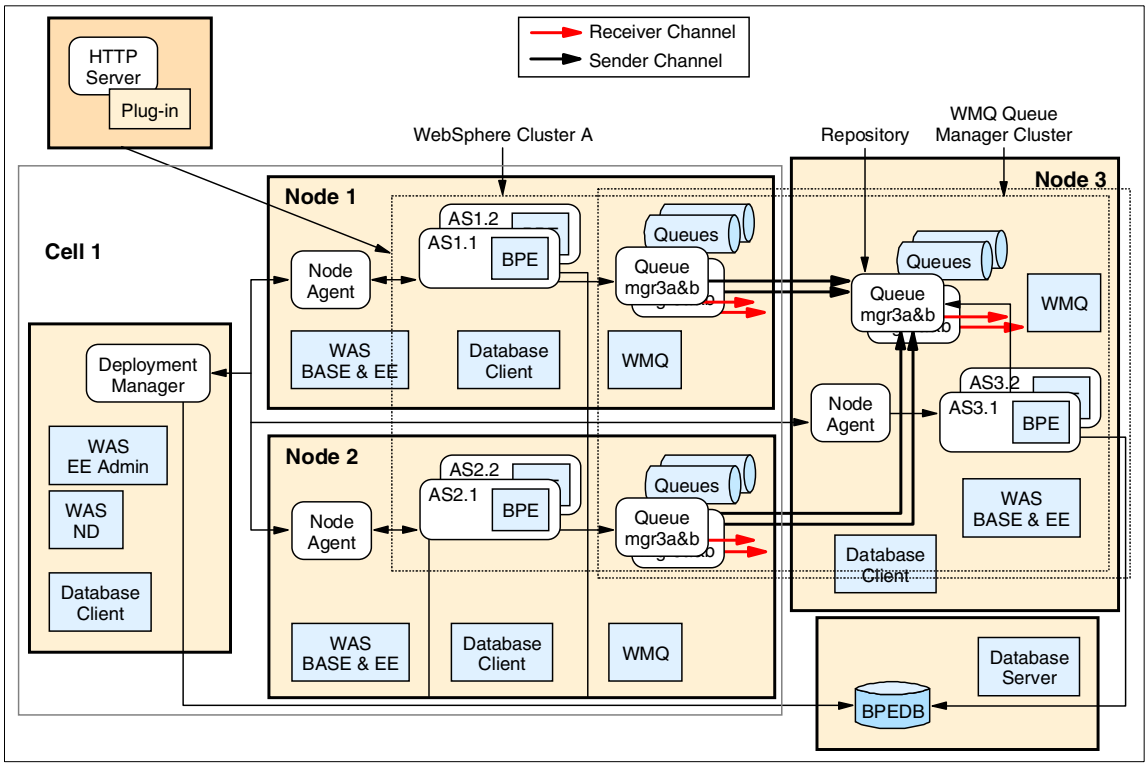


Figure 5-8 Process container with WebSphere Application Server clustering and WebSphere MQ queue manager clustering

5.2.5 Application server cluster and high-availability configuration (DB-C&MQ-C)

You can eliminate single points of failure in topologies 3.a and 3.b by configuring the process database and central WebSphere MQ queue manager on some form of high-availability (HA) hardware/software cluster. If you use an LDAP server for user registry, it also needs to be configured on an HA cluster. For example, you can install the database server and WebSphere MQ server on clustered AIX machines using HACMP (High Availability Cluster Multi-processing). For details on HACMP on AIX, refer to *High Availability Cluster Multi-processing for AIX, Concepts and Facilities Guide V4.5*, found at:

http://publibfp.boulder.ibm.com/epubs/pdf/c2342764.pdf

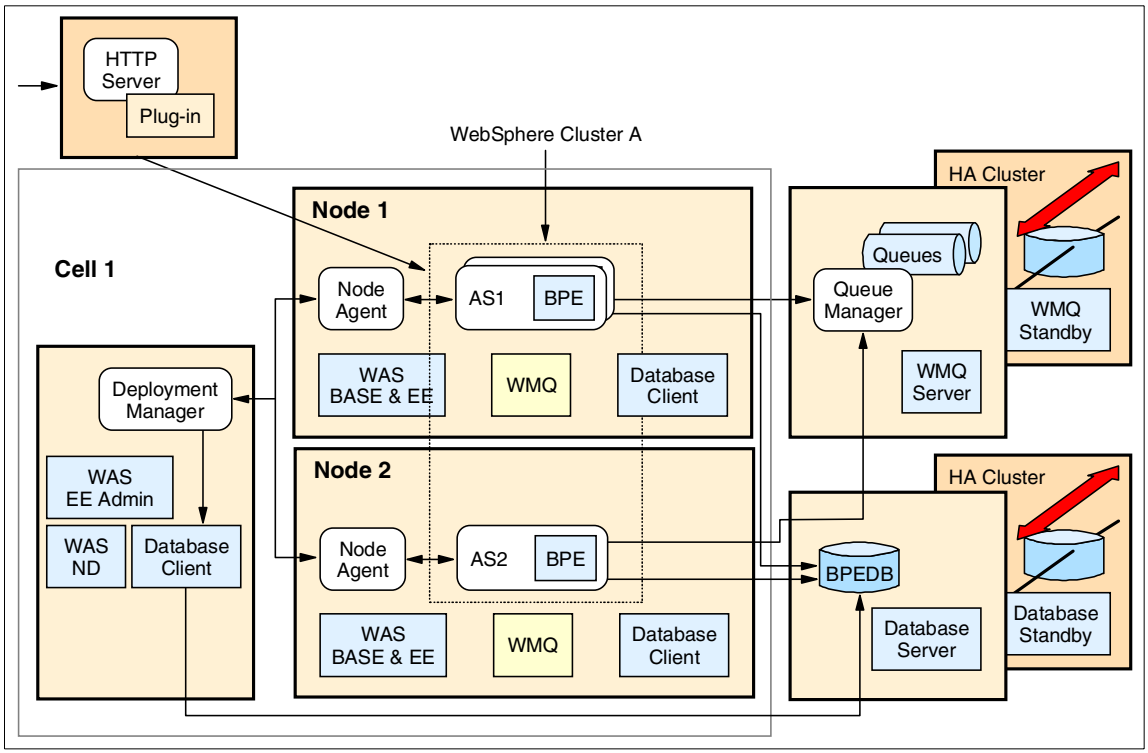


Figure 5-9 High-availability (HA) topology with HA configuration of WebSphere MQ servers and database servers

5.3 Installing the process container

This section describes the installation process for the process container.

5.3.1 Resources required by a process container

In selecting a topology and installing a process container, we need to install the process container a enterprise application and configure resources it requires.

The default names and JNDI names of these resources used by the Process Container Installation wizard of the Administrative Console are as follows:

- JDBC provider and a data source to access its process database. The default JNDI name of the data source is jdbc/BPEDB. The default name of the process database is BPEDB, which may be changed as necessary.

You need to choose a database system for this. Supported database systems are:

- Cloudscape (included with WebSphere Application Server Enterprise)
- DB2 Universal Database™ Enterprise V7.2 fp7 or V8.1
- Oracle8i database Release 8.1.7 or Oracle9i database releases
- Sybase Adaptive Server Enterprise (ASE) V12.0 or higher

- Scheduler to access business process database.

  BPEScheduler is used as both the default name and default JNDI names.

- Work manager to manage threads for the scheduler.

  The default name is BPESchedulerWorkManager (wm/BPEScheduler).

  Scheduler and work manager are services offered by WebSphere Application Server Enterprise.

  For more details on scheduler and work manager, refer to Chapter 13, “Scheduler service” on page 509.

- JMS Provider and the connection factories and JMS destinations.

  Each container requires a pair of JMS connection factories. The default names of these factories are:

  - BPECFC (jms/BPECFC): Factory for put messages
  - BPECF (jms/BPECF): Factory for get messages from

  The connection factory for getting messages (jms/BPECF) has four queue destinations. The default names (JNDI names) of these queues are:

  - BPEIntQueue (jms/BPEIntQueue): Queue for process container internal messages
  - BPEApiQueue (jms/BPEApiQueue): Queue for BPE external API messages
  - BPEHldQueue (jms/BPEHldQueue): Queue for held messages that could not be processed
  - BPERetQueue (jms/BPEIntQueue): Queue for messages that temporarily could not be processed.

- You can choose WebSphere Embedded Messaging Server (WebSphere JMS) or WebSphere MQ (formerly known as MQSeries®) as the JMS provider. If you want to use WebSphere Embedded Messaging as the JMS provider, make sure that both server and client Embedded Messaging features are selected when you install the WebSphere Application Server.

- Three listener ports configured for message listening service of each application server. Their default names are:

  - BPEApiListenerPort: To listen to the BPEApiQueue

  - BPEInternalListenerPort: To listen to BPEIntQueue
  - BPEHoldListenerPort: To listen to BPEHldQueue

- Staff plug-in provider and configuration

  If your business process has human activities, global security must be enabled for staff queries.

5.3.2 Installing required software components

Based on the installation topology, you need to install different required software components on different machines or nodes.

Installing WebSphere Application Servers

Process Choreographer installation instructions in the following sections assume that a WebSphere Application Server (Base, Network Deployment or Enterprise) is already installed in a suitable topology. If you plan to use WebSphere MQ as the JMS provider, make sure you installed WebSphere Application Server without the Embedded Messaging Server option. We recommend that you install and use the WebSphere Application Server Embedded Messaging Client.

For details on WebSphere Application Server Version 5, Base and Network Deployment installation, refer to *IBM WebSphere Application Server V5.0 System Management and Configuration, WebSphere Handbook Series* and the WebSphere Application Server V5.0 InfoCenter.

Installing WebSphere MQ

Before you install WebSphere MQ on UNIX systems, create and mount a journalized file system called /var/mqm for your messaging working data. Use a partition strategy with a separate volume for the WebSphere MQ data. This means that other system activity is not affected if a large amount of messaging work builds up in /var/mqm. You can also create separate file systems for your log data (var/mqm/log) and error files (var/mqm/errors). You should store log files on a different physical volume from the messaging queues (var/mqm). This ensures data integrity in the case of a hardware failure. If you are creating separate file systems, allow a minimum of 30 MB of storage for /var/mqm, 20 MB of storage for /var/mqm/log, and 4 MB of storage for /var/mqm/errors. For installing WebSphere MQ client only, the storage requirement is typically less (for example, 15 MB for /var/mqm).

The /var file system is used to store all the security logging information for the system, and is used to store the temporary files for e-mail and printing. Therefore, it is critical that you maintain free space in /var for these operations. If you do not create a separate file system for messaging data, and /var fills up, all security logging will be stopped on the system until some free space is available

in /var. Also, e-mail and printing will no longer be possible until some free space is available in /var.

For more information on installing WebSphere MQ on various platforms, refer to the appropriate WebSphere MQ Quick Beginnings book:

- *WebSphere MQ for Windows, V5.3 Quick Beginnings*, GC34-6073
- *WebSphere MQ for AIX, V5.3 Quick Beginnings*, GC34-6076
- *WebSphere MQ for Solaris, V5.3 Quick Beginnings*, GC34-6075
- *WebSphere MQ for HP-UX, V5.3 Quick Beginnings*, GC34-6077
- *WebSphere MQ for Linux for Intel® and Linux for zSeries®, V5.3 Quick Beginnings*, GC34-6078

You can get these books from the WebSphere MQ messaging platform-specific books Web page at:

http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html

After installing WebSphere MQ on multiple processor UNIX machines, ensure that all processors can be used by issuing the following command:

```
setmqcap no_of_CPUs
```

where *no_of_CPUs* is the number of processors in the system.

Updating WebSphere environment variables for WebSphere MQ

IBM WebSphere Application Server uses the WebSphere environment variable MQJMS_LIB_ROOT to locate the WebSphere MQ libraries for the WebSphere MQ JMS Provider. This variable in turn is set relative to the environment variable MQ_INSTALL_ROOT, which is the location of the WebSphere MQ (client or server) installation.

Hence, you need to update WebSphere variables for WebSphere MQ, as follows:

1. In the Administrative Console, select **Environment -> Manage WebSphere Variables**.
2. In WebSphere Variables, make sure that the correct scope (node where WebSphere MQ is installed) is selected. Click the link **MQ_INSTALL_ROOT**.
3. In the text area for Value, enter the install root WebSphere MQ. For example:

   On AIX:

   ```
   /usr/mqm
   ```

   On Solaris, HP-UX and Linux:

   ```
   opt/mqm
   ```

On Windows:

```
c:\Program Files\IBM\WebSphere MQ
```

Installing the database system

You can use different database systems for different Process Choreographer business process containers. Your database can run on DB2, Oracle, or Sybase.

Cloudscape database software is included with WebSphere Application Server Enterprise. The sample Process Choreographer configuration uses this database. A Cloudscape database does not support remote database access; hence you cannot use it as a database system for Process Choreographer in a Network Deployment environment. Since Process Choreographer uses distributed transactions, you will need to purchase and install the DTM feature for Sybase ASE.

For UNIX systems, you should install a database system on a dedicated file system.

Installing DB2

Use the installation wizard to install DB2 Universal Database Enterprise Edition Version 7.2 FP7 or higher on the machine where the database is to be located. If the database machine is remote to your application server machine, use the wizard to install the DB2 runtime client on all the application server machines.If you are installing your application servers in Network Deployment, install the DB2 runtime client on the deployment manager machine. The installation wizard will also create a DB2 instance and instance owner *db2_instance_owner* (default is db2inst1) on these machines.

Detailed steps to install DB2 Universal Database systems are described in the following manuals:

- *DB2 Universal Database for UNIX, Quick Beginnings V 7*, GC09-2970
- *DB2 Universal Database for Windows, Quick Beginnings V 7*, GC09-2971
- *DB2 Universal Database, Quick Beginnings for DB2 Servers V 8*, GC09-4836
- *DB2 Universal Database, Quick Beginnings for DB2 Clients V 8*, GC09-4832

On UNIX systems, perform the following steps after installation:

1. Update the .profile file for the user ID used by WebSphere Application Server instance to access DB2. This is the administrative user ID that starts the WebSphere Application Server. The default is root.

   Add the following line:

   ```
   . /home/db2_instance_owner/sqllib/db2profile
   ```

You can also add the line to the WebSphere Application Server, V5 setupCmdLine.sh script.

2. On machines with multiple processors, update the number of processors that can be used with the following command:

   ```
   db2_install_dir/adm/db2licm -l
   ```

   where *db2_install_dir* is installation directory for DB2.

Configuring DB2 client-server communication

If your application server is to access the database system remotely, you need to configure the DB2 runtime client on the application server machine:

1. Log in as instance owner:

   ```
   su - db2_instance_owner
   ```

2. Start the DB2 instance if it is not already started:

   ```
   db2 db2start
   ```

3. Catalog the remote instance. Refer to the DB2 documentation for the parameters that can be used with this command.

   ```
   db2 catalog tcpip node node_name remote IP_address server port_no
   remote_instance db2_instance
   ```

   where *node_name* is the node name to be used by the client to refer to the remote server instance, and *IP-address* is the IP address or host name of the database server.

4. Test communication to the remote instance:

   ```
   db2 attach to node_name user db2_instance_owner using password
   ```

5. To detach from the remote instance, use the command:

   ```
   db2 detach
   ```

Update WebSphere environment variables for DB2 JDBC driver

On the DB2 JDBC Provider definition window of the WebSphere Application Server Administrative Console, the JDBC provider classpath is set by default to ${DB2_JDBC_DRIVER_PATH}/db2java.zip. You can update the classpath by updating the variable DB2_DRIVER_PATH.

Select **Environment -> Manage WebSphere Variables** and set the DB2_JDBC_DRIVER_PATH variable to the value /home/*db2_instance_owner*/sqllib/java12.

Note: Windows NT® or Windows 2000 platforms support only one DB2 installation. The DB2 environment variables are populated in the system environment automatically. It is not necessary that WebSphere Application Server set these environment variables.

5.3.3 Process container on a stand-alone topology (SA)

In this section, we describe how to install a process container to a stand-alone application server. We assume that the following prerequisite steps have been completed:

- WebSphere Application Server Enterprise is already installed on the machine.
- The database system of choice is already installed and configured.
  - For a local database, a database system is installed and configured on the application server machine
  - For a remote database, a database system is installed on a dedicated database server machine. A database client is installed on the application server machine. Communication between the database client and server is configured and tested.
- WebSphere MQ is the JMS provider of choice. It is already installed and configured on the application server machine.

You may install a process container to the application server and configure resources required by the container by:

1. Using the Process Container Installation wizard of the Administrative Console. You must create database tables and WebSphere MQ queue manager and queues manually after running the wizard.
2. Manually configuring necessary resources and using the install user interface in the Administrative Console. To manually configure the process container, the following steps need to be performed:
   a. Configuring the JDBC provider and data source.
   b. Configuring the JMS queue resources.
   c. Creating and configuring the scheduler.
   d. Installing the business process container.

   For details, refer to the WebSphere InfoCenter for Enterprise.

The following steps follow the first configuration approach, using the installation wizard:

1. Make sure that your IBM WebSphere Application Server is running.
2. Launch the Administrative Console for WebSphere.
3. In the Administrative Console, select **Servers -> Application Servers -> *servername***.
4. In the Additional Properties section, click **Business Process Container**.
5. Near the bottom of the Business Process Container window, under Additional Properties, click the link for the **Business Process Container Install Wizard**, and follow the step-by-step instructions provided.

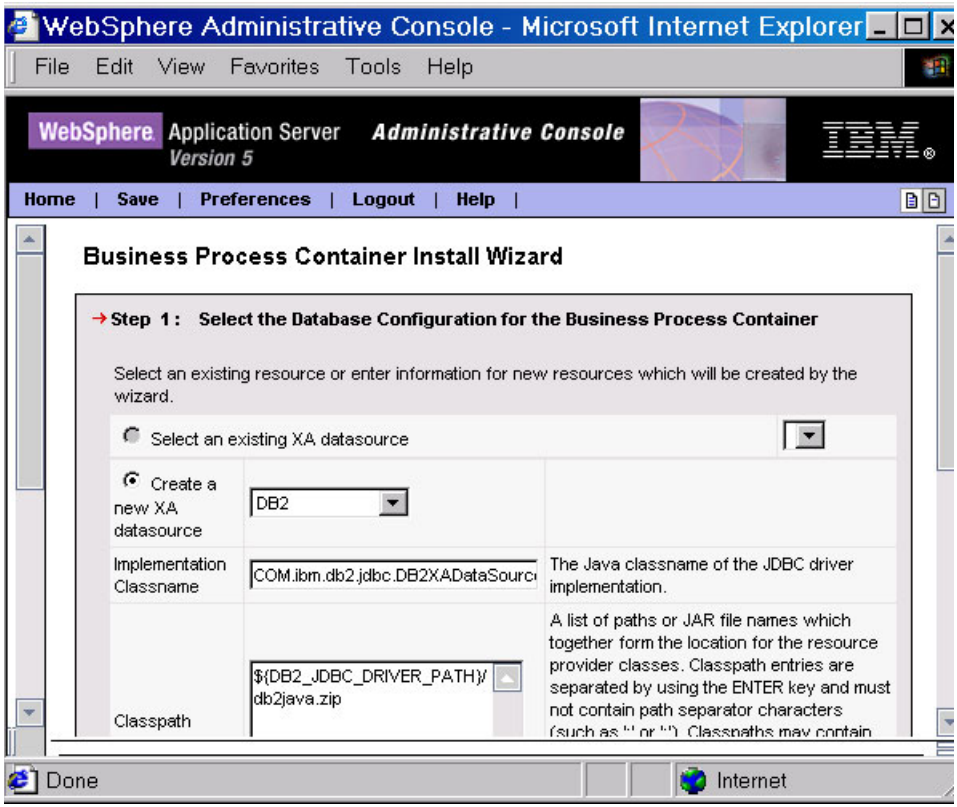


Figure 5-10 Business Process Container Install wizard

6. On the Step 1 page, shown in Figure 5-10, enter the properties of the data source to be used by the process container. Select **Create a new XA data source**, and select **DB2** from the menu list. The GUI will select an appropriate

implementation class name for this driver, for example COM.ibm.db2.jdbc.DB2XADataSource.

Enter the following properties for the data source:

– In the Classpath field, you must enter a list of paths or JAR file names that form the location for the JDBC provider classes. Classpath entries are separated by using the Enter key and must not contain path separator characters (such as ';' or ':'). A classpath may contain variable (symbolic) names, which can be substituted using a variable map. Check your driver's installation notes for specific JAR file names that are required. For a DB2 Universal Database, enter `${DB2_JDBC_DRIVER_PATH}/db2java.zip`, where `DB2_JDBC_DRIVER_PATH` is the WebSphere environment variable for the JDBC driver location defined earlier in "Update WebSphere environment variables for DB2 JDBC driver" on page 191.

– The Datasource user name is the name used to access the data source. This user should have the authority to create and administer the database. For DB2 on UNIX systems, enter the owner of db2 instance db2_instance_owner. For DB2 on Windows, enter the DB2 user ID defined during DB2 installation.

– The Datasource password is the password of the above data source user.

– The Custom properties are extra properties for your database system, such as a database name (default is BPEDB) and a port number. They are specified in multiple lines of property=value pairs.

  The following are the custom properties you may modify for DB2:

  • databaseName: Defines which database to access, for example BPEDB.

  • description: An optional description of the data source. Only used for information purposes.

  • portNumber: An optional integer that specifies the TCP/IP port number where the JDBC provider resides.

  • connectionAttributes: Optional connection attributes specific to DB2. Refer to the DB2 documentation for a complete list of features.

  • loginTimeout: Optional integer. If set to zero, there will be no timeout. Non-zero values specify the maximum number of seconds allowed to establish a connection to the database.

  • enableMultithreadedAccessDetection: Optional boolean. If set to true, it will automatically detect multi-threaded access to a connection and its corresponding Statements, ResultSets, and MetaDatas.

  For custom properties for Oracle, Sybase and Cloudscape databases, refer to the InfoCenter for WebSphere Application Server Enterprise.

Click **Next**.

7. On the Step 2 page, shown in Figure 5-11 on page 197, enter the properties of the JMS provider to be used by the process container:
   - A JMS provider is used for asynchronous messaging based on the Java Message Service (JMS). The default is WebSphere JMS Provider, the Embedded Messaging provider that can be installed with WebSphere. To use external WebSphere MQ as a provider, select **WebSphere MQ JMS provider**.
   - In the Queue manager field is the name of the external JMS provider's queue manager to use for the process container. Keep the default name for the queue manager: `WAS_`*`nodename`*`_server1`.
   - In the Classpath field, you must enter a list of paths or JAR file names that form the location for the JMS provider classes. Classpath entries are separated by using the Enter key and must not contain path separator characters (such as ';' or ':'). The classpath may contain variable (symbolic) names, which can be substituted using a variable map.

     For AIX:

     ```
     /usr/mqm/java/lib
     ```

     For Solaris, HP-UX and Linux:

     ```
     /opt/mqm/java/lib
     ```

     For Windows, enter the WebSphere MQ installation path, for example:

     ```
     C:\Program Files\IBM\WebSphere MQ
     ```

     Tip: WebSphere will use the variable ${MQJMS_LIB_ROOT} as the classpath that is set to ${MQ_INSTALL_ROOT}/java/lib automatically. If you have already set the WebSphere variable MQ_INSTALL_ROOT, you do not need to edit this classpath.

   - The queue connection factory uses a JMS user ID to establish a connection to the queue, by defining a J2C authentication alias to this user ID. This user ID is not integrated with WebSphere Application Server security. For an external WebSphere MQ provider, this user ID is in a user registry (OS or LDAP) that WebSphere MQ security is using. For UNIX systems, the user ID must belong to the mqm group.
   - The JMS password is the password for the JMS user ID.

Note: If WebSphere MQ is installed on the same machine as WebSphere Application Server, the transport type is set to Bind (default). Then the JMS user ID must be the same as the user ID with which WebSphere Application Server is started.

8. In the Business Process Container Security Configuration pane shown in Figure 5-11 on page 197, set the following values:
   - Security Role mapping: The user or group from the user registry that is associated with the Business Process Administrator role. The user or group is defined in the user registry configured for the WebSphere Application Server security. This user ID will be mapped to the role defined in the process container application's Deployment Descriptor, for example BPESystemAdministrator.
   - JMS API User ID: The user ID that the Business Process container MDB will use when processing asynchronous API calls. This user ID will be mapped to the role defined in the process container application's Deployment Descriptor, for example JMSAPIUser.
   - JMS API password: The password for the user ID entered above.

   Set these values and click **Next**.

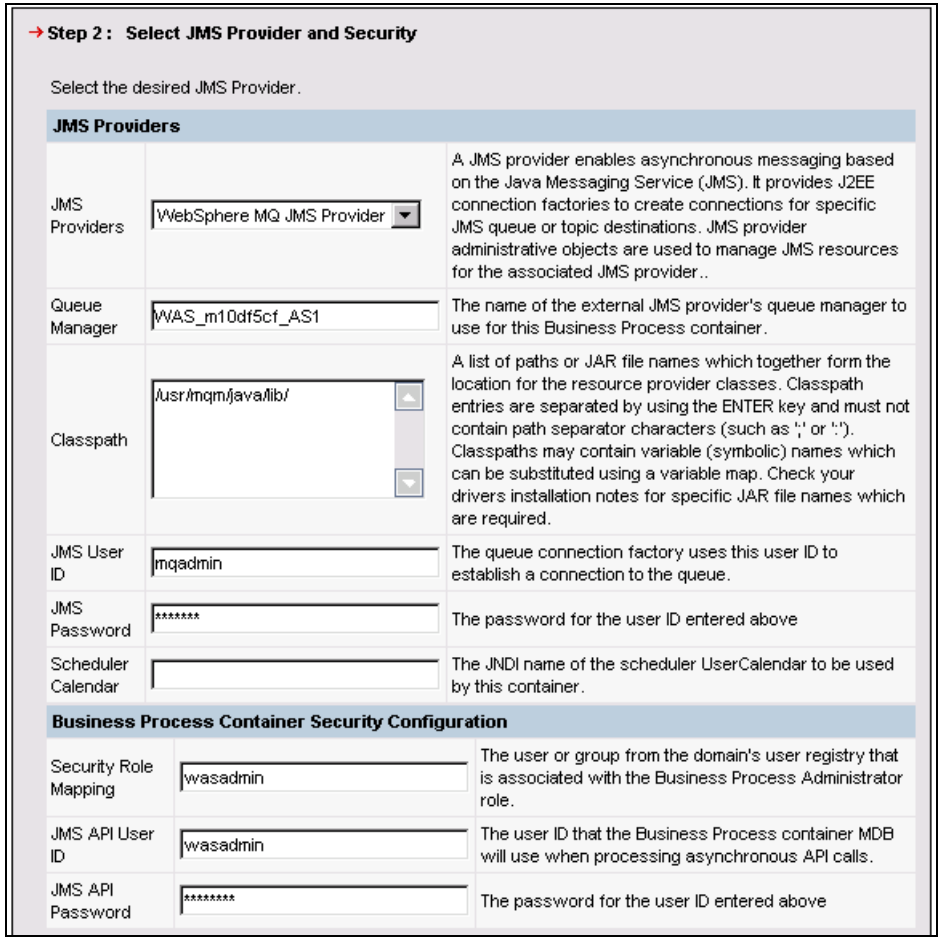


Figure 5-11 Step 2 of process container install wizard

9. On the step 3 page, shown in Figure 5-12 on page 198, keep the default selection of the Create new MQ resources and click **Next**.

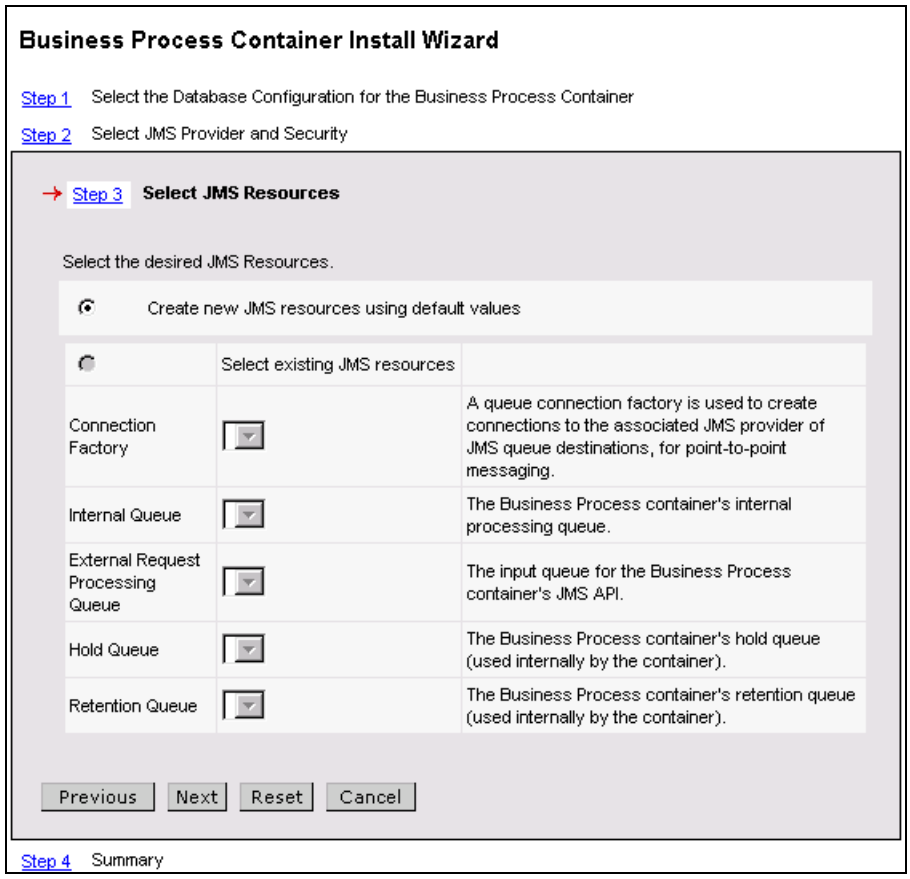


Figure 5-12 Selecting JMS resources in install wizard

10. For step 4, review the summary shown in Figure 5-13 on page 199 to ensure all entries are correct. Also review the reminder of tasks to be performed after install wizard finishes.

    Click **Finish**.

11. Save the configuration for WebSphere after getting a response window with the message:

    `BPEContainer_`*`nodename`*`_server1 installed successfully.`

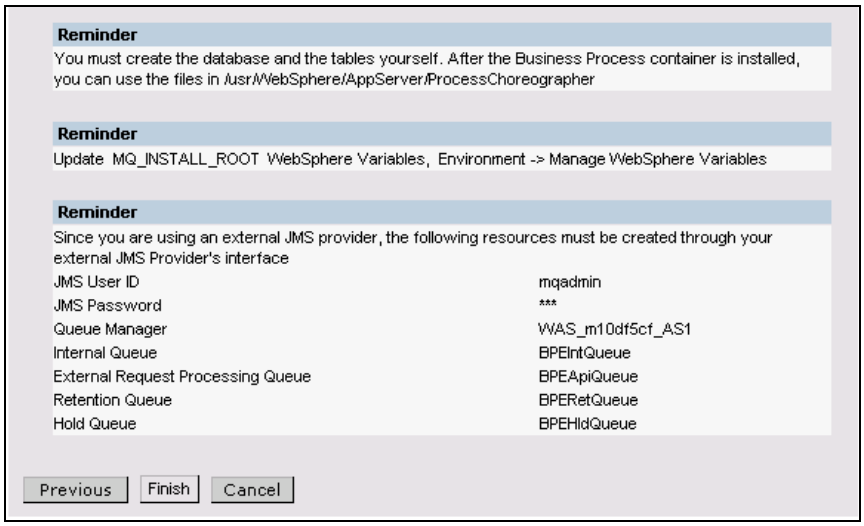


Figure 5-13 Step 4 remaining tasks reminders with WebSphere MQ as JMS provider

12. There are two major steps left to finish the process container installation.
    - You need to create the business process database and tables.
    - You need to create the queue manager and queues if you are using WebSphere MQ as the JMS provider. The queue manager and queues are created by the wizard if you selected WebSphere JMS (Embedded Messaging) as the JMS provider.

    The two steps are covered in the following sections.

Creating the process database and tables

To create the process database and tables, perform the following steps:

1. Create the database and tables using the scripts provided in the ProcessChoreographer directory:

   On Windows:

   ```
   %WAS_HOME%\ProcessChoreographer
   ```

   On UNIX:

   ```
   $WAS_HOME/ProcessChoreographer
   ```

   Where `WAS_HOME` is WebSphere Enterprise installation root, for example:

   On Windows:

   ```
   C:\WebSphere\AppServer
   ```

   On AIX:

```
/usr/WebSphere/AppServer
```

On Solaris, HP-UX, and Linux:

```
/opt/WebSphere/AppServer
```

2. Copy the file createDatabaseDb2.ddl from this directory to the database server machine if it is a remote server.

   Edit this file if necessary, for example to change the database name from BPEDB to something else.

   Note: The database name of the process database created on the database server should be aliased to the database name specified in the process container of your WebSphere Application Server (DB2 client) machine.

3. On UNIX, make sure the db2 environment variables are set by executing the profile:

   ```
   . /home/db2_instance_owner/sqllib/db2profile
   ```

4. Log in as a DB2 instance owner on the DB2 server.

   ```
   su - db2_instance_owner
   ```

5. Create the database by issuing the following commands through CLI:

   ```
   db2 -tf createDatabaseDb2.ddl
   ```

6. Bind the CLI programs against the database:

   For UNIX:

   ```
   db2 connect to BPEDB
   db2 bind $DB2DIR/bnd/@db2cli.lst blocking all grant public
   db2 connect reset
   ```

   For Windows:

   ```
   db2 connect to BPEDB
   db2 bind %DB2PATH%\bnd\@db2cli.lst blocking all grant public
   db2 connect reset
   ```

7. Catalog the process database to an alias on the client.

   On an Application Server machine where DB2 runtime client is installed, catalog the just-created database BPEDB to BPEDB_alias, which is the alias specified in installing the process container:

   ```
   su - db2_instance_owner
   db2 catalog database BPEDB as BPEDB_alias at node node_name
   ```

Creating a queue manager and queues in WebSphere MQ

To create a WebSphere MQ queue manager, perform the following steps:

1. Locate the script in the ProcessChoreographer directory:

   On Windows:

   ```
   %WAS_HOME%\ProcesChoreographer\createQueues.bat
   ```

   On UNIX:

   ```
   $WAS_HOME/ProcessChoreographer/createQueues.sh
   ```

   where `WAS_HOME` is WebSphere Enterprise installation root. Typical locations for `WAS_HOME` are:

   On AIX:

   ```
   /usr/WebSphere/AppServer
   ```

   On Solaris, HP-UX, and Linux:

   ```
   /opt/WebSphere/AppServer
   ```

   On Windows:

   ```
   C:\WebSphere\AppServer
   ```

2. Go to the ProcessChoreographer directory and run the script to create the queue manager named WAS_qmgr_name, specified in the install wizard and the four queues required by the process container.

   On Windows:

   ```
   createQueues.bat WAS_qmgr_name
   ```

   On UNIX:

   ```
   ./createQueues.sh WAS_qmgr_name
   ```

3. After a successful execution of the scripts, enter the following command to check the status of the queue manager:

   ```
   dspmq
   ```

   To check the queues created, enter the following command:

   ```
   runmqsc
   ```

   At the mqsc prompt, type the command to display a list of local queues:

   ```
   dis (pl*)
   ```

   Note: You cannot have more than one queue manager with BIND transport on UNIX systems. If you have another application that requires a second queue manager on UNIX systems, the transport of the JMS provider for the second queue manager should be specified as CLIENT with an associated port number (the default is 1414).

5.3.4 Installing process container in a cell topology (ND-noC)

In the ND-noC topology, we have a Network Deployment cell with multiple application server nodes, with one application server instance on each node. The Network Deployment manager is on a second dedicated machine. There is no WebSphere Application Server clustering.

Deployment manager

1. Install the WebSphere Application Server Network Deployment (ND).
2. Install the Administrative Console extensions for WebSphere Enterprise using the WebSphere Application Server Enterprise installation wizard.
3. Install the Web server and the Web server plug-in.
4. Install the database client for the database system of your choice to access the process database on a database server.

Database server

1. Install your database system of choice.
2. Create a process database BPEDB1 for Application Server AS1.
   a. Copy the database creation DDL file from an application server machine to the database node.
   b. Edit the database creation DDL and change the database name from BPEDB to BPEDB1. You can also change the database path.
   c. Create the database with appropriate commands using the DDL.

Node1

1. Install WebSphere Application Server Enterprise Version 5 (which includes Base Version 5). If you want to use WebSphere Embedded Messaging, make sure that you include both the Embedded Messaging Server and Client when WebSphere is installed. if you plan to use WebSphere MQ as the JMS provider, ensure that the Embedded Messaging Server feature is not installed.
2. Install the client software for your database system of choice.
3. If you plan to use WebSphere MQ as the JMS provider, install WebSphere MQ V5.3.01.
4. Create a server, for example, AS1.
5. Install the process container on AS1 using the Process Container Installation wizard. Make sure the database name is changed to BPEDB1 (from the default BPEDB) and the database is cataloged on Node1 to access the database BPEDB1 on the database server remotely.

6. Create the local queue manager and queues if WebSphere MQ is used as a JMS provider.

Node2

1. Install WebSphere Application Server Enterprise V5 (which includes Base Version 5). If you want to use WebSphere Embedded Messaging, make sure that you include both the Embedded Messaging Server and Client when WebSphere is installed. if you plan to use WebSphere MQ as the JMS provider, ensure that the Embedded Messaging Server feature is not installed.
2. Install the database system of choice and create a process database for application server AS2.
3. If you plan to use WebSphere MQ as the JMS provider, install WebSphere MQ V5.3.01.
4. Create a server, for example AS2.
5. Install the process container on AS2 using the Process Container Installation wizard. Make sure the database name is changed to BPEDB2 (from the default BPEDB) and the database is cataloged on Node2 to access the database BPEDB2 locally.
6. Create the local queue manager and queues if WebSphere MQ is used as a JMS provider.

Adding nodes to the cell

You can add the nodes configured to the deployment cell either by using the Administrative Console or by using the wsadmin command-line interface.

1. Add Node1 to the deployment manager cell:
   a. Make sure Node1 is up and running.
   b. Bring up the Administrative Console for the cell.
   c. Select **System Administration -> Nodes**.
   d. Click **Add Node**.
   e. In the Add Node window, enter the network host name of Node1.
   f. If security is enabled, enter the user ID and password of the WebSphere administrator for the deployment manager.
   g. Select **Include applications**.
   h. Click **OK**.

   Alternatively, from the command line:

   a. Change directory to the bin directory of the stand-alone application server installation.

b. Enter the **addNode** command:

```
addnode dmgr_hostname SOAPport -includeapps
```

where *dmgr_hostname* is the host name of the deployment manager and *SOAPport* is the port number of the SOAP connector (default is 8879).

2. Add Node2 to the deployment manager cell:

Repeat the above steps, but enter the network host name of Node2 instead of Node1.

3. In adding an existing stand-alone node to a cell, specify a bootstrap port number (default is 9810) that is different from those being used by existing nodes.

a. Select **Servers -> Application Servers**.

b. Click the name of the server whose port is to be updated, for example **server1**.

c. Under Additional Properties, click **End Points**.

d. Click **BOOTSTRAP_ADDRESS**.

e. Under Configuration properties, change the entry for Port to an available port number.

- Update the Web server plug-in.
  - Select **Environment -> Update Web Server Plugin**.
  - Click **OK** to generate the plug-in.
  - Check that the generated configuration file plugin-cfg.xml is correct and that it is located at the right directory.

    A plug-in configuration file named plugin-cfg.xml is generated in the directory <*DeploymentManager_root*>/config/cells. The entry for the plug-in configuration file in <Web_Server_root>/conf/httpd.conf file points to <*WebSphere_root*>/config/cells/plugin-cfg.xml by default.

    You can either edit this entry in httpd.conf tile to point to the correct location (and restart the HTTP server) or copy the plugin-cfg.xml file to the right location.

5.3.5 Installing process container in topology (ND-C, ND-VS, ND-HS)

First, install and configure the necessary software components.

Web server

Install the Web server and Web server plug-in.

Deployment manager

1. Install WebSphere Application Server Network Deployment.
2. Install WebSphere Application Server Enterprise.
3. Install database client software for the database system of your choice.
4. Configure the database to access the process database remotely from the client.

Database server

1. Install database system of your choice.
2. Copy the database creation DDL from an application server machine to the database node.
3. Edit the database creation DDL and change the database name if necessary. You can also change the database path.
4. Create the database with the appropriate commands using the DDL file.
5. Configure the database to allow remote client access to the process database.

WebSphere MQ server

This topology requires a dedicated WebSphere MQSeries server with a central queue manager. This queue manager is responsible for both the put and the get messages.

1. Install WebSphere MQ 5.3.01 or higher.
2. Create a central queue manager:
   a. Copy the script $WAS_HOME/ProcessChoreographer/createQueues.sh on UNIX (createQueues.bat on Windows platforms) from a WebSphere Application Server Enterprise node to the WebSphere MQSeries server.
   b. Run the **createQueues** command, specifying the name of the queue manager (*Queue_Manager_Name* in the following command):

      ```
      ./createQueues.sh Queue_Manager_Name
      ```
3. Add a listener service:
   a. On UNIX, add a dedicated port for the new queue manager to the /etc/services file:

      ```
      Service_Name port_number /tcp
      ```

      where *Service_Name* is the name of the queue manager service and *port_number* is the port number for the listener (default is 1414).
   b. Add a dedicated port for the new queue manager to the /etc/inetd.conf file:

```
Service_Name stream tcp nowait mqm /usr/mqm/bin/amqcrsta amqrsta -m
Queue_Manager_Name
```

where *Service_Name* is the name of the queue manager service (the same name defined in the /etc/services file) and *Queue_Manager_Name* is the name of the queue manager.

c. On Windows platforms, use WebSphere MQ Explorer to add the listener.

Application server nodes

1. Install WebSphere Application Server Enterprise V5 (which includes Base Version 5).
2. Install the client software for the database system of your choice.
3. Install WebSphere MQ Client Version 5.3.01. You may choose to use the WebSphere Embedded Messaging Client instead.

For topology ND-VS, add Node1 to the cell. For topology ND-HS, add Node1 and Node2 to the cell. Once the nodes are added to the cell, process containers can be installed. There are two ways to install business process containers to application servers in a WebSphere Application Server cluster.

1. Create a cluster with two or more servers without the process container. Then install the process container to the cluster using the install wizard.
2. Create a server and install a process container to the server using the wizard. Then create a cluster using this server as a template.

Creating cluster first and installing process container

1. Create an application server AS1 on Node1 using the default server as a template.
2. Create a cluster including the server AS1 as a template.
3. Add a second server clone AS2 to the cluster. For topology ND-VS, AS2 is located on the same node as AS1. For topology ND-HS, AS2 is located on a second node, Node2.
4. Configure the WebSphere MQ JMS provider with two connection factories and four queue destinations manually at the cell scope. Specify Client binding and provide the host name and port number for the WebSphere MQ server.
5. Install the process container using the install wizard. In the step 3 page, select **Use existing JMS resources**. Then select the queue connection factories and queue destinations set up in the previous step from the drop-down menu.

Creating a server with process container first and creating a cluster

1. Create the application server AS1 using the default server as a template.
2. Install the process container using the Process Container Install wizard.

Note that the install wizard uses the following default values to configure new JMS resources JMS connection factories: jms/BPECF, jms/BPECFC, and BIND transport. Hence after the install wizard is finished, you must edit the two connection factory configurations to change it to Client transport and provide the host name and port number for the WebSphere MQ server machine.

a. Select **Resources -> WebSphere MQ JMS Provider -> WebSphere MQ Queue Connection Factories**.

b. Click the link for the connection factory **BPECF**.

c. Enter the network host name of the WebSphere MQ server.

d. Enter the port number of the WebSphere MQ service (1414).

e. Click **OK**.

f. Repeat the same steps (c to e) for the connection factory BPECFC.

g. Save the configuration for WebSphere.

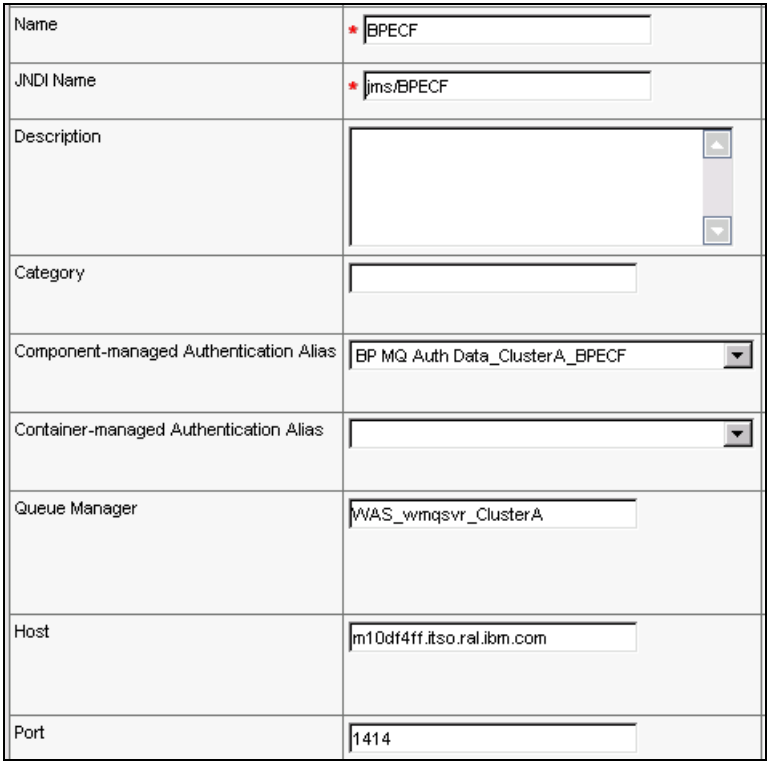


Figure 5-14 Changing the connection factory configuration to Client transport

3. Create a cluster including the server AS1 as a template.
4. Create a clone server AS2.

   This will also clone the process container and all the resources configured in AS1 for the process container on the server AS2.

5.4 Staff plug-in provider configuration

Important: WebSphere global security must be enabled when using business process staff activities.

As mentioned in 4.5, “Staff support” on page 158, staff resolution plug-ins resolve the abstract staff queries (that is, verbs and parameters specified when the process is developed, also known as *parameterized verbs*) into real searches performed against the appropriate user registry at runtime. The registry used to resolve staff queries (also called the *staff registry*) can be the same registry as that used by the WebSphere Application Server itself, or a separate LDAP registry.

It is important to remember that staff plug-ins and the staff registry are only used to authorize a previously authenticated user ID to interact with the business process or its staff activities. User authentication is handled by WebSphere itself, using the registry that is configured as part of the global security configuration.

The layers of indirection between the abstract staff queries and the real registry queries allow developers to implement human interaction into business processes in a manner that is both highly customizable and completely independent of the actual staff registry which is configured at deploy time.

Each staff resolution plug-in (also called staff plug-in) is associated with a JNDI name that is used by process templates at deploy time to determine which plug-in is used at runtime. This information is included in the process’ ibm-flow.xmi file, which is part of the flow archive (FAR) file that represents the process template. In WebSphere Application Server Enterprise Version 5.0, there is no way to edit this value in either the Administrative Console or in the Application Assembly Tool. Therefore it must be set using WebSphere Studio IE, as follows:

1. Right-click the service project containing the business process.
2. Click the **Staff** button in the left-hand panel to see the staff properties.
3. Enter the JNDI name of the desired staff plug-in.
4. Click **OK** when done.

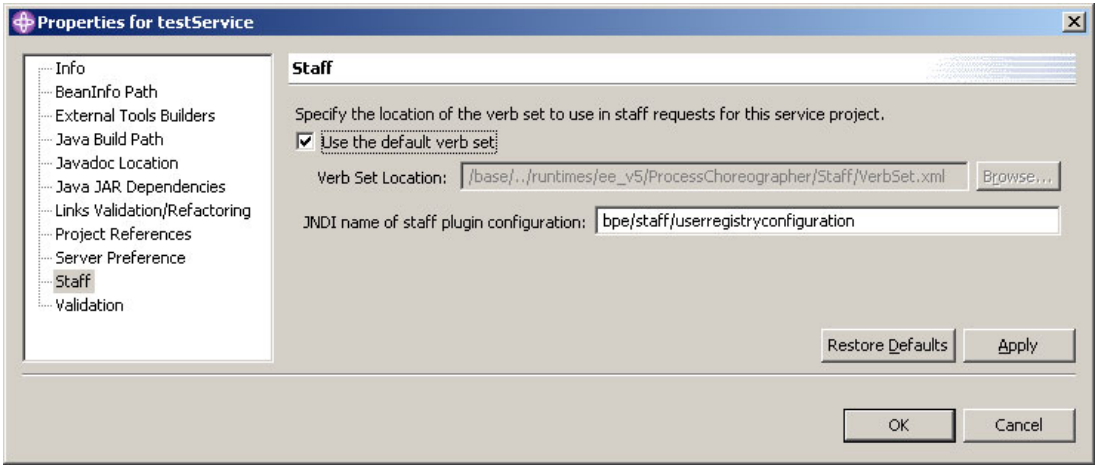


Figure 5-15 Configuring which Staff plug-in a process should utilize

Each staff resolution plug-in is associated with an XSLT (Extensible Stylesheet Language Transformations) file, which transforms the staff query verbs into the language supported by the plug-in. The plug-in may also be associated with required properties. For example, the LDAP staff plug-in requires various LDAP server settings (for example, host name, port, baseDN, etc.) to be set as properties during the plug-in configuration.

In the WebSphere Application Server Enterprise configuration, staff resolution plug-ins are collected into *staff plug-in providers*. A staff plug-in provider is a JAR file that contains the necessary Java class files for zero, one, or many staff resolution plug-ins. For example, the User Registry staff plug-in consists of the UserRegistryTransformation.xsl, which is mapped to the JNDI name bpe/staff/userregistryconfiguration, and is contained within the User Registry staff plug-in provider that is mapped to the JAR file $(WAS_INSTALL_ROOT)/lib/bpestaffuserregistry.jar.

While there is no published API for creating new staff plug-in providers, the staff plug-in XSL files and the parameterized verb XML schema that are the input to the staff plug-in transformation are documented in the *WebSphere Application Server Enterprise Process Choreographer Staff Resolution Parameter Reference.* Using this, it is possible to create new staff plug-ins based on one of the existing plug-in providers (System, User Registry, or LDAP). An example of this can be found in “Creating custom staff query verbs” on page 163.

The WebSphere Application Server Enterprise is installed with three staff plug-in providers, each of which is described in the following sections.

System staff plug-in provider

After installation, the System staff plug-in provider contains two staff plug-ins, both of which are intended for developer use only. Neither plug-in uses a back-end staff registry, and therefore neither is suitable for production environments. The plug-ins included within the System staff plug-in provider are:

- The Everybody staff plug-in, which transforms all staff queries to return *Everybody*. This plug-in can be very useful during testing and debugging, since it allows any authenticated user to claim any staff activity, regardless of the query verb and parameter associated with that activity.
- The System staff plug-in, which supports the simple staff query types `everybody`, `nobody`, and `userID`. This plug-in is also quite useful during testing and debugging, but it cannot handle the more advanced query verbs supported by the LDAP and User Registry plug-ins.

User Registry staff plug-in provider

The User Registry staff plug-in provider contains the User Registry staff plug-in. This plug-in uses the registry that the WebSphere server is itself configured to use for global security. Using this plug-in will force the business process engine to use the same registry for authentication and authorization. This is the most straightforward way to configure staff support for a production environment. It is installed ready to use and requires no additional configuration parameters.

The user registry staff plug-in will work in conjunction with any of the three supported WebSphere user registries: local OS, LDAP, and custom user registry.

Note: When using a custom user registry for WebSphere security in conjunction with the User Registry staff plug-in, the deprecated getUsersForGroup() interface must be implemented for the custom registry.

The User Registry staff plug-in supports the following query verbs:

- users
- users by userID
- group members
- person search
- group search
- native query
- everybody
- nobody

LDAP staff plug-in provider

The LDAP staff plug-in provider contains the LDAP staff plug-ins, which are intended for processes that require staff authorizations that are dependent on organizational information, such as management chains, or various other user

attributes that are commonly found in LDAP directories, such as serial number, contact information, location, etc. The LDAP staff plug-ins can provide these capabilities by supporting a rich set of query verbs and parameters.

The LDAP staff plug-in provider supports all of the query verbs supported by the User Registry staff plug-in, in addition to the following:

- Department members
- Role members
- Manager of employee
- Manager of employee by user ID

In addition to supporting addition query verbs, the LDAP staff plug-in provider also supports additional query parameters. For example, when using the User Registry plug-in, a person search query is limited to searching using the UserID parameter, while the LDAP registry plug-in provider allows searches using any attribute that is part of the LDAP schema.

Because staff plug-ins are only used for authorization, authentication is handled by WebSphere itself. Therefore, the user registry configured for global security must be capable of authenticating every user who might interact with the staff activity. The simplest configuration using the LDAP staff plug-in is one in which WebSphere global security is configured to use the same LDAP directory used by the LDAP staff plug-in. (Note that the user registry staff plug-in would also work and indeed be preferable in this case, if the more complex query verbs supported by the LDAP staff plug-in are not required.) An alternative would be to configure WebSphere to use another user registry containing the subset of the user IDs in the LDAP directory that might interact with the process' staff activities.

After installation, a sample LDAP staff plug-in is configured, with the JNDI name bpe/staff/sampleldapconfiguration. This sample is useful as a guide for creating an LDAP staff plug-in that is suitable for use in your environment.

The LDAP transformation file used by the sample LDAP staff plug-in, $(WAS_INSTALL_ROOT)/ProcessChoreographer/Staff/LDAPTransformation.xsl, might need to be modified to reflect your LDAP schema. Mapping between query verb parameters and LDAP attributes can be found at the top of this file, in the global variables section, as shown in Example 5-1. The variables with the prefix "GS" are used by the group search queries, and the variables with the prefix "PS" are used by the person search queries.

Example 5-1 Mapping between query verb parameters and LDAP attributes

```
<!-- Begin global variables -->
  <xsl:variable name="Threshold">20</xsl:variable>
  <xsl:variable name="DefaultPersonClass">inetOrgPerson</xsl:variable>
  <xsl:variable name="DefaultUserIDAttribute">uid</xsl:variable>
```

```
  <xsl:variable name="DefaultManagerAttribute">manager</xsl:variable>
  <xsl:variable name="DefaultRecursivity">yes</xsl:variable>
  <xsl:variable name="DepartmentFilter">(OrgType=Department)</xsl:variable>
  <xsl:variable name="RoleFilter">(OrgType=Role)</xsl:variable>

  <xsl:variable name="GS_GroupID">unkown</xsl:variable>
  <xsl:variable name="GS_Type">unkown</xsl:variable>
  <xsl:variable name="GS_IndustryType">unknown</xsl:variable>
  <xsl:variable name="GS_BusinessType">unkown</xsl:variable>
  <xsl:variable name="GS_GeographicLocation">unkown</xsl:variable>
  <xsl:variable name="GS_Affiliates">unkown</xsl:variable>
  <xsl:variable name="GS_DisplayName">unkown</xsl:variable>
  <xsl:variable name="GS_Secretary">unkown</xsl:variable>
  <xsl:variable name="GS_Assistant">unkown</xsl:variable>
  <xsl:variable name="GS_Manager">unkown</xsl:variable>
  <xsl:variable name="GS_BusinessCategory">unkown</xsl:variable>
  <xsl:variable name="GS_ParentCompany">unkown</xsl:variable>

  <xsl:variable name="PS_UserID">uid</xsl:variable>
  <xsl:variable name="PS_Profile">unkown</xsl:variable>
  <xsl:variable name="PS_LastName">sn</xsl:variable>
  <xsl:variable name="PS_FirstName">unkown</xsl:variable>
  <xsl:variable name="PS_MiddleName">unkown</xsl:variable>
  <xsl:variable name="PS_Email">unkown</xsl:variable>
  <xsl:variable name="PS_Company">unkown</xsl:variable>
  <xsl:variable name="PS_DisplayName">unkown</xsl:variable>
  <xsl:variable name="PS_Assistant">unkown</xsl:variable>
  <xsl:variable name="PS_Secretary">unkown</xsl:variable>
  <xsl:variable name="PS_Manager">manager</xsl:variable>
  <xsl:variable name="PS_Department">unkown</xsl:variable>
  <xsl:variable name="PS_EmployeeID">unkown</xsl:variable>
  <xsl:variable name="PS_TaxPayerID">unknown</xsl:variable>
  <xsl:variable name="PS_Phone">unkown</xsl:variable>
  <xsl:variable name="PS_Fax">unkown</xsl:variable>
  <xsl:variable name="PS_Gender">unkown</xsl:variable>
  <xsl:variable name="PS_Timezone">unkown</xsl:variable>
  <xsl:variable name="PS_PreferredLanguage">unkown</xsl:variable>
<!-- End global variables -->
```

So, for example, if you need to do a person search using the “Department” query parameter against an LDAP directory that uses the attribute name “dept”, then edit the `PS_Department` variable definition to read:

```
<xsl:variable name="PS_Department">dept</xsl:variable>.
```

Before using an LDAP staff plug-in, it must be configured via the WebSphere Administrative Console. The properties required by each LDAP staff plug-in are specified by the LDAP staff plug-in provider JAR file. Remember that each LDAP

staff plug-in maintains its own transformation file and its own set of LDAP staff provider properties. The following steps are required for the configuration of an LDAP staff plug-in:

1. On the WebSphere Administrative Console, select **Resources ->Staff Plugin Provider**.
2. On the Staff Plugin Provider window, click **LDAP Staff Plugin Provider**.
3. On the LDAP Staff Plugin Provider window, click **Staff Plugin Configuration**.
4. On the Staff Plugin Configuration window, click **LDAP Staff Plugin Configuration sample**.
5. Click **Custom Properties**.
6. On the Custom Properties window, specify the LDAP server URL (for example, `ldap://localhost:389`) in the ProviderURL field, the Base Distinguished Name (for example, `o=itso,c=us`) in the BaseDN field, and the search scope (for example, `subtreeScope`) in the SearchScope field. If the LDAP server requires authenticated access, specify the name of the appropriate J2C authentication alias in the AuthenticationAlias field.

Query result caching

By default, when the business process container performs a staff query to generate work items, the results of this query are cached for one hour. These cached results will be re-used when creating work items within the current process instance, as well as within other process instances.

The default query cache duration can be changed by modifying the StaffQueryResultValidTimeSeconds property in the business process engine properties file <WAS_ROOT>\properties\bpe.properties.

5.5 Security considerations

First we look at issues related to securing the Process Choreographer's resources. Then we look at the security context of a running process.

5.5.1 Securing the Process Choreographer resources

When a business process is used to provide a vital business function, all of the resources used by the process engine should be secured. These are:

- The business process data source.
- The JMS queue destinations used internally to manage processes: BPEApiQueue, BPEHldQueue, BPEIntQueue, and BPERetQueue.

Securing the data source resource used by the business process is similar to securing any other data source. Specify the appropriate J2C authentication alias for the data source's connection factory (for example, jdbc/BPEDB), and ensure that no untrusted user IDs may access the back-end database.

Similarly, securing the queue resources used by the business process is a matter of specifying the appropriate J2C authentication alias for the queue connection factory used by the process engine (for example, jms/BPECF), and ensure that no untrusted users may access the JMS provider's queue destinations.

Important: The Process Choreographer configuration JACL scripts will create data source and JMS resources in WebSphere with the appropriate security applied. If using Embedded Messaging, the queues themselves will also be created with security applied. However, if using the external MQ JMS provider, the queue access controls must be applied manually in order to secure these queues. For more information, see the WebSphere MQ Security product documentation.

5.5.2 Process security context

A business process will either run with the security context of the user who started the process or with no security context, depending on the mechanism used to start the process. There are several ways a process may be started:

- If the process is started using the business process Web client, the process will inherit the security context of the user running the Web client.
- If the process is started via a Session EJB binding, the process will inherit the security context of the session bean that "wraps" the process. Unless the session bean uses delegation, this will also be the security context of the caller of the EJB.
- Similarly, if the process started via a SOAP binding, it will inherit the security context of the Web Project that "wraps" the process. If there is a security constraint on the service's URL requiring the Web service client to authenticate, the process will run with the security context of this authenticated user. If there is no security constraint on the Web service's URL, the process will run with no security context.
- If the process is started via a JMS binding, it will run with no security context, unless delegation is used on the message-driven bean that "wraps" the process. Using delegation, the process can be made to run as a particular user, or with the WebSphere server's user identity.

Once started, a process maintains the same security identity throughout its lifetime. While various users may interact with the process through staff activities, the process' security context will remain unchanged.

An exception to this can occur in the case of an interruptible business process that is started by a user ID that becomes not valid before the process is complete. For example, a process that is started by an employee via the business process Web client will run using the security context of that employee. If this employee leaves the company before the process runs to completion, the process will be unable to complete. If this occurs, a process administrator will eventually need to terminate and delete the process instance.

5.5.3 Process Choreographer J2EE security roles

There are three J2EE security roles within the business process container application: the Process Administrator role, the WebClientUser role, and the JMSApiUser role. Each are discussed in the following sections.

Note: For more information on J2EE security roles and how they are administered, see the *WebSphere Version 5 Security Handbook*.

Process Administrator role

As described in "Process-level staff roles" on page 162, process-level administrative control may be assigned using the appropriate staff query configuration, or, if the process-level administrator role is left unassigned, it is inherited by the user ID that starts the process instance. In addition to these methods of acquiring process-level administrative control, it may also be assigned via the business process container's Process Administrator security role configuration. This attribute of the business process container is an optional configuration when the process container is installed, and it may also be modified subsequently by editing the security role bindings for the business process container application.

In contrast to the process-level administrator staff role, which is assigned using staff queries, the Process Administrator security role is a J2EE security role that is bound directly to user IDs and groups in the user registry that WebSphere is currently using for global security. This mechanism provides a way for WebSphere administrators to assign process-level administrative control that applies to all process templates, and which is independent of all aspects of the process staff configuration.

WebClientUser role

The WebClientUser role is, by default, bound to the All Authenticated Users special subject. This role controls who has access to the business process Web client, and the default setting allows any member of the user registry to access this Web application. Note, however, that the process staff roles will dictate whether or not users may be able to do anything after accessing the business process Web client.

In some cases, it might be useful or necessary to restrict access to the business process Web client to a subset of users. This can be easily done via the WebClientUser security role binding.

JMSAPIUser role

The JMSAPIUser security role is used to apply a Run-as role (also known as *delegation*) to the receiver bean that responds to the business process JMS API interface. Since message-driven beans have no security context when responding to received messages, Run-as roles are often used to insert a particular user's identity, so that these beans' methods run with a known security context.

The user ID and password used for the JMSAPIUser Run-as role can be configured when the business process container is installed on the application server, or subsequently by editing the Run-as mapping for the business process container application.

If using the business process JMS API interface, it is necessary to map a user who is a member of the Process Administrator security role to the JMSAPIUser Run-as role. Otherwise, if not using the JMS API interface, this role can be left unconfigured.

5.6 Managing business process applications

Business process applications are applications that contain the business processes. They can be administered in a similar manner to any other J2EE applications with a few additional considerations.

5.6.1 Artifacts of a business process application

Business process applications are packaged into Enterprise Application Archive (EAR) files used to package standard J2EE applications. Typically, a J2EE EAR file contains Web application archive (WAR) files, EJB modules in EJB Java archive (EJB JAR) files, Java archive (JAR) files that contain supporting Java classes, and other files contain properties and configuration information. EJB

modules are deployed into EJB containers, and Web application modules are deployed into Web containers.

A business process application project called as a service project in WebSphere Studio IE can include one or more process models. Each business process model is described in Flow Definition Markup Language (FDML). When a deployable service is generated in WebSphere Studio IE, all necessary artifacts of the service project are packaged into an EAR file. The EAR file includes:

1. One flow archive (FAR) file named *project_name*.far, where *project_name* is the name of the service project. The FAR file includes:
   - ibm-flow.xmi that specifies process templates included in the FAR file, including their validFrom date and Web context.
   - One FDML file for each process model. Each FDML file is named *model_name*.fdml, where *model_name* is the name of the business process model or template.
2. One Java archive (.jar) file whose default name is *project_name*.jar. This file includes the following artifacts:
   - One *model_name*.process file for each process, which defines each process model graphically in WebSphere Studio IE.
   - One class file *model_name*.class file for each process, that includes Java snippets for data transformations, conditions for process transitions and loops, and accessor methods for process variables. This class is extended from BPE ProcessBackingBase.
   - Java classes for all global variables defined in each process model.
   - WSDL files for the interface, binding, and services defined for each process model.
   - Java classes used in implementation of activities in each process model.
   - All supporting files such as XML files, properties files used in implementation of activities in each process model.
   - Services file that specify staff plug-in JNDI names and the verb sets to be used with staff activities for all process models in the project.
3. One EJB JAR file (default name *project_nameEJB*.jar) that includes deployed classes session EJBs, if EJB façade and JMS façade are generated for all the process models in the project.

   We may have other EJB JAR files that may be used in implementing activities in each process model.
4. One WAR file (default name *project_nameWeb*.war) that includes Web services (SOAP) interfaces to all process models in the project.

We may have other WAR files that contain Web applications that may be used in implementing activities or custom user interfaces to each process model.

5. Resource archive, RAR files: Zero or more .rar files for connecting to J2EE resources such as JCA connectors to CICS.
6. Other J2EE EAR artifacts: Normal J2EE artifacts for an EAR produced for WebSphere Application Server Enterprise, such as:

- application.xml
- ibm-application-ext.xmi
- ibm-application-bnd.xmi

At a minimum, a business process application EAR should consist of the following files:

- *project_name*.far
- *project_name*.jar
- Either a *project_nameEJB*.jar or a *project_nameWeb*.war

5.6.2 Installing business process applications

You install these applications in the same way as other J2EE applications, using either the administration scripts or the application install windows on the Administrative Console. You also need to configure any resources required by dependent J2EE components that implement services, such as data sources, JMS resources, and CICS connectors. However, because FAR files are not J2EE modules, they do not appear on the application install windows.

When you install a business process application, a FAR file is deployed to a business process container by deployment tools under the covers. The rest of the J2EE artifacts are deployed to the usual places. For example, EJB JARs are deployed into an EJB container, WAR files are deployed into Web containers, and JAR files and RAR files are deployed into directories for installed classes. Configuration data for your application is added to the WebSphere configuration repository, while process metadata is added to the Process Choreographer databases.

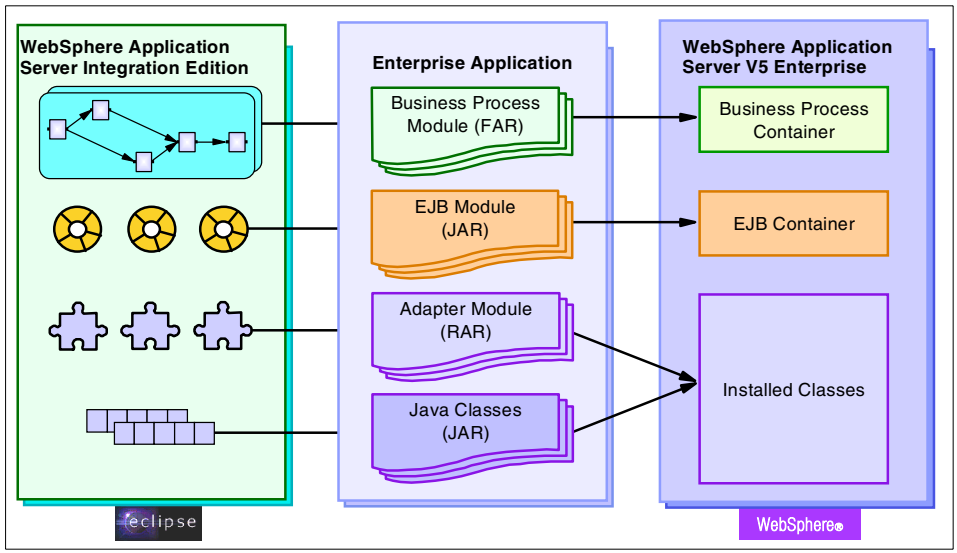


Figure 5-16 Deployment of a business process application

Currently, all the FAR files included in your business process application EAR file are distributed to the application servers and WebSphere clusters where you install any of the EJB JAR or WAR files included in this EAR file. It is recommended that you install all the modules (EJB, WAR and FAR) together on the same application servers and WebSphere clusters. As a consequence, you must configure business process containers on all these application servers and WebSphere clusters.

If you want to separate WAR files from the FAR files of your application, put them in a different EAR file and install this EAR file separately. You cannot separate EJB jars from FAR files.

When you install a process application, all the stand-alone servers and at least one application server of each WebSphere cluster where you want to install the application modules must be running. The corresponding database servers must also be running.

If only some of the required servers are available at installation time, install the application on the servers and WebSphere clusters that are running. You can map the application to further servers and WebSphere clusters later.

5.6.3 Versioning process models

You can specify a validFrom date and time for a process model. This value is specified in the Coordinated Universal Time (UTC) format. The validFrom date

and time can be in the past. This time, together with the name of the model, is used as the unique key to version a process template stored in the process database.

You can change the validFrom time in WebSphere Studio IE by selecting the Server view of the process in the process editor, as shown in Figure 5-17.

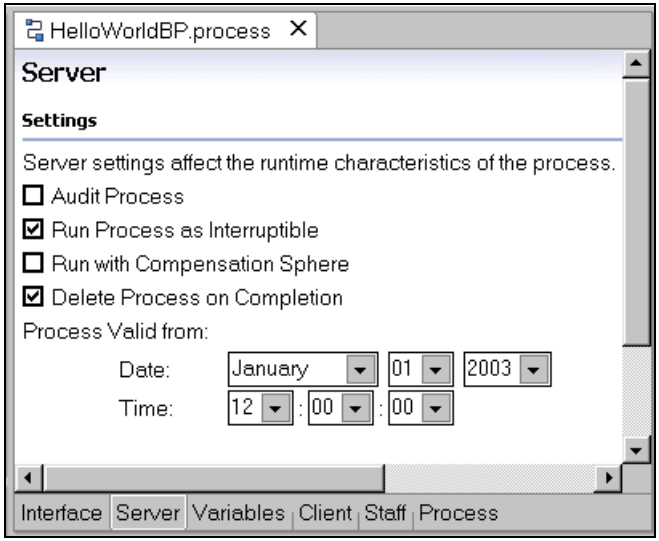


Figure 5-17 Setting validFrom date and time in WebSphere Studio IE process editor

You can overwrite an already installed process model as long as its validFrom time is still in the future, that is, it has no instances yet. Both process model name and validFrom time have to match that of the already installed process model to do that.

You can provide a new version for an already installed process model, which may or may not have instances. The new version is identified by the same name and a newer validFrom time. Existing instances continue to run with the previous version of the process model, while instances created after the validFrom time of the new model use the new version.

If an overwritten process model is already referenced by an activity of another process model, only upward compatible changes of its interface signature are allowed. This is necessary because of late binding, since the new version might be picked up by an already existing process, and thus its interface signature must be guaranteed to be compatible with the signature of all existing activities using it. The same applies to implementations.

For a given version of the process template, you can change the behavior of external implementation resources while an instance of that version is running by upgrading EJBs or services that are used by the process model, if these resources are in different EAR modules than that of the live process and the interfaces for these implementations remain the same.

To update a business process model while keeping the live instances of the original process to run to completion, do the following:

1. In WebSphere Studio IE, change the validFrom date and time of the model to something more current than that of the existing process model.
2. Generate new deployable code making sure to specify JNDI names used for the EJB façade or JMS façade different from those used with the live instance of the old process version.
   a. If your process model has activities invoking EJBs or services and you are updating their implementations with the new version, you must specify different JNDI names for these if you do not want live instances of the old version to start using the new activity implementations.
   b. If another process model is referencing your process model through its EJB façade or JMS façade, you also need to update the referencing process model accordingly. That is, change its validFrom time to be the same as that of your process model (in case it has live instances) and install it, mapping its EJB reference to the new JNDI name of your new process model. The referencing process model should use EJB references to allow updating of JNDI names at deployment.
3. Export the EAR file.
4. Install EAR specifying an application name that is different from the existing process application.
5. Save the configuration.
6. Start the new application.

All new instances of this process model created after the new validFrom date use this template of the process instead of the old template. The old template remains in place so that existing process instances can continue to completion.

5.6.4 Starting and stopping process templates

When a business process module is installed successfully, the process templates that are contained in the process module (FAR file) are started automatically. If you need to stop these process templates, perform the following steps:

1. Be sure that you are using a Console User ID that has either the operator or administrator role.

2. In the navigation pane of the Administrative Console, select **Applications -> Enterprise Applications**.
3. Click the business process application you want to manage.
4. In the Related Items section of the configuration properties window, click **Business Process Modules**.
5. Select the process module you want to manage.
6. In the Additional Properties section, click **Templates**.
7. The Process Templates window is displayed, listing the process templates that are contained in the selected process module.

You can stop a running process template, and start it again at a later point in time, depending on the validFrom time that is specified. A process template will not be started until the validFrom time is reached.

To stop a process template, check the box next to the process template and click **Stop**.

To start a process template, check the box next to the process template and click **Start.**

5.6.5 Uninstalling a business process application

When you uninstall a business process application, you remove it from all the servers and WebSphere clusters on which it is installed. If you want to remove the business process application from selected servers or WebSphere clusters only, use application editing.

To uninstall a business process application in a distributed environment:

1. Ensure that all the stand-alone servers, at least one application server per cluster, and the corresponding database servers are running.
2. Stop all process templates belonging to the application. This stops the process template on all the application servers and WebSphere clusters on which the application is installed.
3. Remove all instances of the process templates.
4. Uninstall the business process application.

5.6.6 Editing a business process application

When you edit a business process application, you change the mapping of the application modules to the application servers and WebSphere clusters in the

cell. You can use the Administrative Console or administration scripts to change this mapping.

Only the EJB modules and Web modules appear in the Administrative Console. If you change the mapping of your EJB or Web modules, the mapping of the process modules (FAR files) is changed accordingly.

During the editing of a business process application, the following restrictions apply:

- If you remove the mapping of a module to an application server or a WebSphere cluster, then the process templates of that application must be stopped and all instances of the templates must be removed from the database belonging to the corresponding application server or WebSphere cluster.
- All application servers and at least one member of each WebSphere cluster that you want to change (remove or add) must be running. The associated database servers must also be running.

5.6.7 Managing process instances

Each process container installation includes a default Web client to manage processes. It provide a graphical Web interface to view process templates installed on the application server or cluster. The URL for invoking the default Web client is:

`http://host_name:port_no/bpe/webclient`

where *host_name* is the network host name of your application server and *port_no* is the HTTP transport port number. You can start, stop, terminate and delete process instances using the Web client. For interruptible processes with staff activities, it allows users with potential owner, reader or editor roles to work on the staff activities assigned to them.

5.7 Problem determination and troubleshooting

Process Choreographer uses the WebSphere framework JRas for traces, messages, and audit logs. Typical steps applied in troubleshooting J2EE applications on WebSphere Application Server can be applied when troubleshooting process container and business process applications, keeping in mind that process container uses JMS resources and a JDBC data source for the process database.

5.7.1 Error messages

All messages within Process Choreographer are NLS enabled. At runtime the message catalog is responsible for handling the localized messages (messages depend on the locale set in your system). All messages for one target language are stored in a properties file with the naming convention you specify.

All errors, warnings and informations will be written to SystemOut.log in WAS_install_dir/logs/server_name. You can identify messages from Process Choreographer by looking at the message key:

```
HelloWorldBP operation failed javax.management.MBeanException: BPEA0008E:
Process template 'HelloWorldBP.Wed 2003-01-01 12:00:00.000' is not found.
```

The message key in this example is `BPEA0008E`, where:

- BPE is the identifier for Process Choreographer
- A is the component within Process Choreographer:

  D Deployment
  A API
  E Engine
  C Configuration
  P Plug-ins
  U Client

- 0008 is the message code within the component.
- E is the type of message:

  I Information
  W Warning
  E Error

After the message key, you can find the localized message for this message key. This message should provide the first information, such as:

```
Process template 'HelloWorldBP.Wed 2003-01-01 12:00:00.000' is not found.
```

If you need further information, you can use the Log Analyzer, which provides an explanation and possible user actions for solving your problem.

5.7.2 Tracing process container

Process Choreographer uses the WebSphere framework JRas for traces. To turn on tracing for Process Choreographer through the Administrative Console, do the following:

1. Select **Troubleshooting ->Logs and trace**.

2. In the table, select a server from the list of servers.
3. Click the link **Diagnostic Trace** to bring up the Diagnostic Trace Service window.
4. On the Configuration tab in the General Properties column, check **Enable Trace**.
5. To specify components to trace, perform one of the following steps:
   – In the Trace specification box, enter:

     ```
     com.ibm.bpe.*=all=enabled
     ```

   – Click **Modify** to bring up a pop-up window. In the pop-up window, click the **Components** tab.
6. Click the package **com.ibm.bpe** and select **All enable**.

   You may select to choose specific levels of tracing or specific subpackages of com.ibm.bpe:

   ```
   com.ibm.bpe.client.*
   com.ibm.bpe.engine.*
   com.ibm.bpe.staff*
   com.ibm.bpe.util.*
   ```

   You may change the trace output setting. Trace output is sent to a file at <*WebSphere_root*>/logs/trace.log.
7. Click **OK**.
8. Save the configuration for WebSphere.

5.7.3 Process audit trail

To use the audit trail for Process Choreographer, it must have been enabled during the modeling of a process. It can be specified on the Server tab of the process model in WebSphere Studio IE. It can also be specified at the activity level in WebSphere Studio IE. This setting is entered into the FDML (Flow Definition Modeling Language) file for the process model.

When a process instance is executed and audit trailing is enabled, Process Choreographer writes information about each significant event into an audit log, which is located in the table AUDIT_LOG_T in the process database. Process Choreographer provides a plug-in for the audit trail database to be used. In addition, you can clean up the audit log table according to your needs by using the cleanup utility called BPEAuditLogDelete. To read the content of the audit trail, use SQL or any other administration tool that allows you to read database tables.

Enabling the audit log has an adverse effect on process container performance. Auditing has a relatively small impact on performance when using interruptible business processes.

6

Extended Messaging

Asynchronous messaging patterns are an important part of the J2EE application's programming model because of the flexibility that they bring to a distributed application architecture. Asynchronous messaging systems allow for "loose coupling" between different applications, and between the different programs that constitute a single, distributed application. Importantly, these systems can span an extremely wide number of platforms. Moreover, when sending and receiving messages, the senders and recipients of these messages do not need to know about each other directly. This lets them evolve independently of each other. These patterns are useful for process flows, parallel processing, time-independent processing, and event-driven processing.

This chapter introduces Extended Messaging (EM), discusses the need for EM, introduces the design patterns handled by EM, and works through a sample application showing both sending and event style (MDB) receiving of messages using EM. The second part of the chapter explores the runtime implications of EM, including configuration, deployment, transactional concerns, and problem determination.

6.1 Planning

This chapter describes the limitations of current Java messaging APIs, then introduces a new set of technologies called Extended Messaging (EM) that IBM introduced in WebSphere Enterprise V5. EM combines a number of development and runtime enhancements to make the delivery of messaging programs faster and more efficient in a WebSphere environment.

6.1.1 Java Message Service

The Java Message Service (JMS) is a set of Java APIs that provides a framework for Java programs to make portable calls into asynchronous messaging systems. The JMS specification was developed by several vendors and is described in detail at:

http://java.sun.com/products/jms/index.html

However, JMS is a standard Java API, and does not take full advantage of the J2EE container. In addition, JMS is a relatively low-level API and requires the developer to write code for:

- Object lookups: JNDI lookups for connection factories and destinations.
- Message formats: The developer is tasked with constructing the parts of a message (such as XML), and then parsing these message parts upon retrieval.
- Caching of administration objects: For better performance, all thread-safe JMS objects should be cached.

The EJB 2.0 specification introduced message-driven beans (MDB), which provide container support for event-driven message retrieval. But the MDB specification still does not handle a number of messaging patterns, such as:

- Sending messages with or without a response.
- Application-callable receiver beans. Messages need to be retrieved during a program's application flow, as opposed to the MDB style, where messages are processed as soon as they arrive at a destination.
- Late responses in send and receive. This allows the sender to register an MDB type listener to handle responses that cannot be processed within a programmatically set timeout.
- Mapping and reformatting data in JMS messages. This lets the programmer set the details of the JMSMessage as parameters of a message, instead of having to build and parse the message data.
- Sending messages to multiple destinations (topics or queues).

6.1.2 Extended Messaging

The next generation of messaging tools and runtime support is here. It is called Extended Messaging (EM) and is delivered with WebSphere V5. EM tools are included in WebSphere Studio IE, while runtime and configuration support are part of WebSphere Enterprise.

6.1.3 Why use Extended Messaging?

EM enhances standard J2EE Messaging by providing support for all types of messaging patterns, container support for these patterns, and code simplification. The following messaging patterns are supported by EM:

- Sending
  - Send - fire and forget
  - Send with a synchronous response (optional timeout)
  - Send with a deferred response
- Receiving
  - Event style (MDB) with no response
  - Event style (MDB) with a response
  - Application callable (optional timeout) with no response
  - Application callable (optional timeout) with a response

Container support

Because EM EJB beans run within the WebSphere Enterprise V5 runtime, they take full advantage of J2EE caching, clustering, transactional, and security constructs.

Code simplification

As discussed above, one of the problems with JMS coding is the amount of code and J2EE constructs you have to write just to send a message. A traditional JMS application looks like Example 6-1.

Example 6-1 JMS Code to send a message

```
// Initialize the JNDI context
InitialContext ctx = new InitialContext();

// Look up the QueueConnectionFactory in JNDI
Object o = ctx.lookup("jms/myQCF");
qcf =
(QueueConnectionFactory)PortableRemoteObject.narrow(o,QueueConnectionFactory.cl
ass);

// Create a QueueConnection
```

```
QueueConnection conn = qcf.createQueueConnection();

//Create a QueueSession
QueueSession session  =
conn.createQueueSession(false,Session.AUTO_ACKNOWLEDGE);

// Look up the Queue in JNDI
Object o = ctx.lookup("jms/myQ");
q = (Queue)  PortableRemoteObject.narrow(o,Queue.class);

// Create a QueueSender
QueueSender sender = session.createSender(q);

// Create a message
Message message = session.createTextMessage();
message.setJMSType("LogMessage");
message.setText("Hello World");

// Send the message
sender.send(message);
```

As mentioned earlier, there is no container support for caching of connection factories, queues, senders, and so on. All that has to be done by hand.

Extended Messaging promotes code simplification by:

- Providing advanced tooling support. All messaging patterns discussed can be supported by running wizards in WebSphere Studio IE. This is discussed in detail in 6.3, “Development” on page 234.
- Data mapping of message parts.
- Sending messages to multiple destinations via Deployment Descriptor settings on input and output ports.
- Tooling support provides response queue handling.
- Runtime container caching of administration objects.

6.2 Design

Extended Messaging is provided as a set of application interfaces that provide an abstraction for JMS.

6.2.1 Messaging patterns

Let’s take a closer look at the sending and receiving patterns.

Sending pattern

The sending pattern, illustrated in Figure 6-1, follows this scenario:

1. Client application looks up a sender bean and calls one of its sender methods.
2. The sender method formats the message parts into a single message.
3. The sender bean sends the message to an output port.
4. The WebSphere Enterprise runtimemaps the output port to one or more JMS destinations.
5. The WebSphere Enterprise runtime sends the message to the destination defined for the output port.

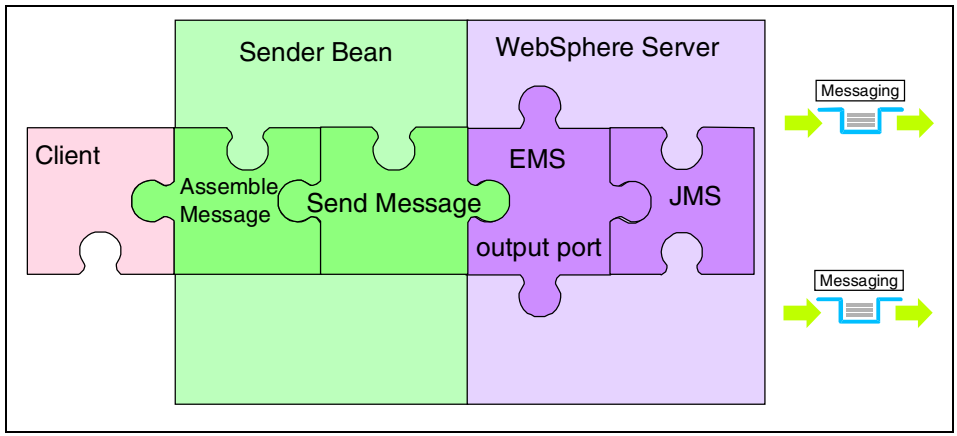


Figure 6-1 Sending process

An Extended Messaging port provides an abstraction to JMS resources. There are three types of ports, as follows:

- EM output port:
  - Specifies the output queue and connection factory (this is required).
  - Specifies the reply-to queue (this is optional).
  - If a sender expects a reply and the reply-to queue is left blank, a temporary queue is opened to accept the response.
  - Reply-to queue is required for a send with deferred response, because a temporary destination cannot be left open.
  - Reply-to queue is required for late-message handling.
- EM input port:
  - Specifies the input queue and connection factory.

- Only used by Application Callable Receiver beans.
- The reply-to queue specifies where to send replies.

- EM extended MDB listener port:
  - Used when an MDB performs the role of a late response message handler.

The EM output port is illustrated in Figure 6-2.

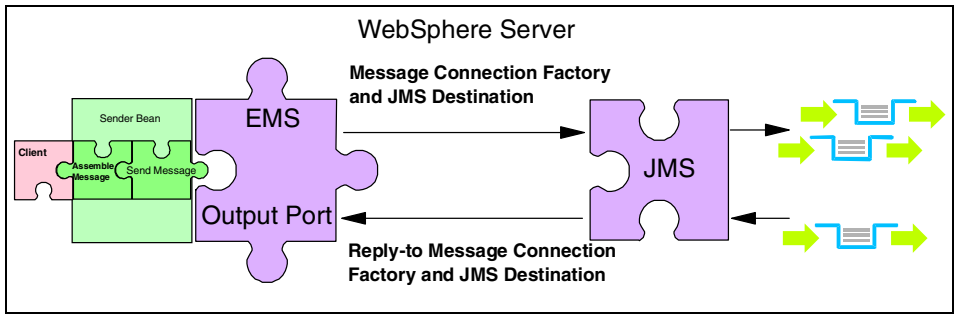


Figure 6-2 Output port

Receiving pattern

The receiving pattern has two forms:

- MDB style
  - The MDB listens to a JMS destination, receives the message, and then calls a worker session bean.
  - There is no input port for the MDB style.
- Application callable receiver bean
  - Receives the message by listening to the destination specified in the input port.
  - Parses the message.
  - Returns the message contents to the calling application.

Both receiving patterns are illustrated in Figure 6-3 on page 233.

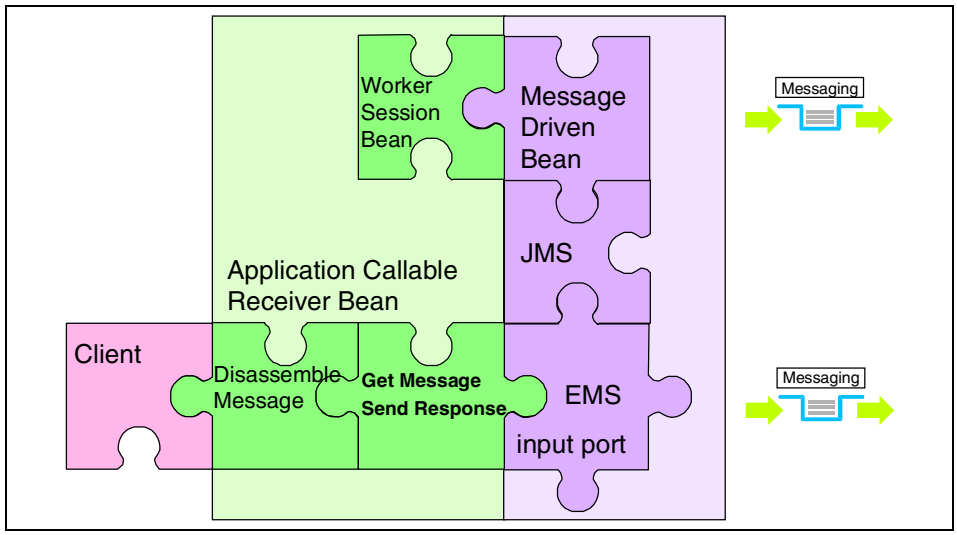


Figure 6-3 Receiving process

6.2.2 Programming considerations

Extended Messaging programs are generally built using wizards in WebSphere Studio IE, although the runtime interfaces are provided by the WebSphere Enterprise runtime and can be programmed by hand. Design considerations are similar to standard JMS and MDB applications except that EM will greatly simplify the coding of these applications. The output of the WebSphere Studio IE wizards will be either a session bean for sending messages and application callable receiving of messages, or an MDB for event style receiving of messages. These EJBs can then take full advantage of the J2EE container runtime. In the scenario example included with this redbook, an EM sender bean will be built and included in the Process Choreographer.

Important design considerations are:

- Event style receiving of messages: Create a receiver bean that implements javax.ejb.MessageDrivenBean.
- Programmatic sending/receiving of messages: Create a session bean with methods for send or receive.
- Data mapping: Use data mapping if the message has multiple parts to facilitate the parsing of the message parts.
- Reply: For both sending and receiving of messages, replies can be specified so that information can be sent to a secondary (reply) destination.

- MDB style receiver: If a MDB type receiver is being used, it is best to write the session bean that the onMessage() method will call. Then during the creation of the receive bean, the tooling will generate the calls to the appropriate session bean. This style drives a clean separation of messaging logic and business logic.
- Deferred response: Used when the sender bean does not want to block for a reply. The application calls receiveResponse() to programmatically retrieve the response method.

Note: For a tutorial that uses a deferred response, see part II of:

http://www7b.software.ibm.com/wsdd/library/tutorials/0303_cox/cox_reg.html

- Late response: Used when there is a possibility of a missed response in send/receive scenarios. When receiving a response and a timeout or other error occurs, a MDB can be used as a late response handler to retrieve the response and correlate with the sender, and process the response in the onMessage() of the late response handler MDB.

6.3 Development

The development process for EM programs is detailed in this section. The primary classes used during development are:

- CMMFactory: Static class used to build senders, receivers, parsers, and formatters.
- CMMSender: Used to send a message and optionally receive a response.
- CMMReceiver: Used as an application callable receiver.
- CMMParser: Used during data mapping to parse the message parts via name/value pairs.
- CMMFormatter: Used to build complex data mapping for messages.

6.3.1 Sample scenario

To illustrate the development process, a sender and receiver scenario will be built that can later be incorporated into this redbook's sample scenario. The order process flow will call an EM sender during the ProcessOrder activity. The ProcessOrder activity will pass the itemID, description, price, and quantity to the sender to have the order placed on a queue. Then an MDB will read the order off the queue and make the data available to other back-end processes for order completion. For the sake of the sample, the order will be written to the system

log, but a real process could make a call into a back-end process such as CICS. The sample application flow is illustrated in Figure 6-4.

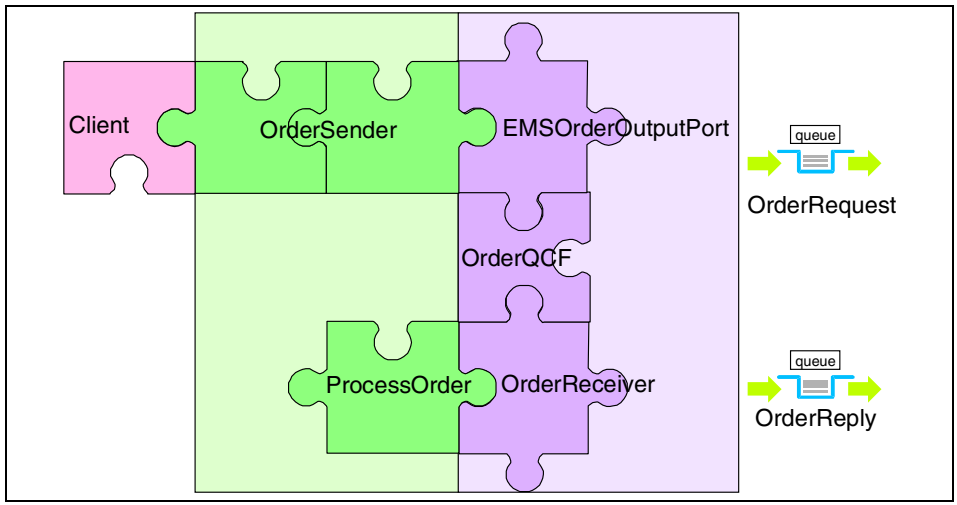


Figure 6-4 Sample application with generic names

The application components with their corresponding JNDI names are:

- Sender bean: OrderSender
- Output port: EMSOrderOutputPort, ems/itso/OrderOutputPort
- QueueConnectionFactory: OrderQCF, jms/itso/QCF
- Request queue: OrderRequestQueue, jms/itso/requestQ
- Reply queue: OrderReplyQueue, jms/itso/replyQ
- Receiver MDB: OrderReceiver
- Worker Session bean: ProcessOrder

6.3.2 Creating the sample

The sample application will be created in WebSphere Studio IE, as follows:

1. Launch WebSphere Studio IE with your workspace location.
2. Open the J2EE perspective.
3. Create a new J2EE 1.3 Enterprise Application project, with the following details:

   Enterprise application project name: `ACompanyEMS`

   Unselect application client module, because this isn't required.

Create the OrderSender bean

The following steps will describe how to create the OrderSender bean for the sample:

1. Select the **EJBModules** and **ACompanyEMSEJB**.
2. Right-click and select **Extended Messaging -> Create Sender bean**, as shown in Figure 6-5.

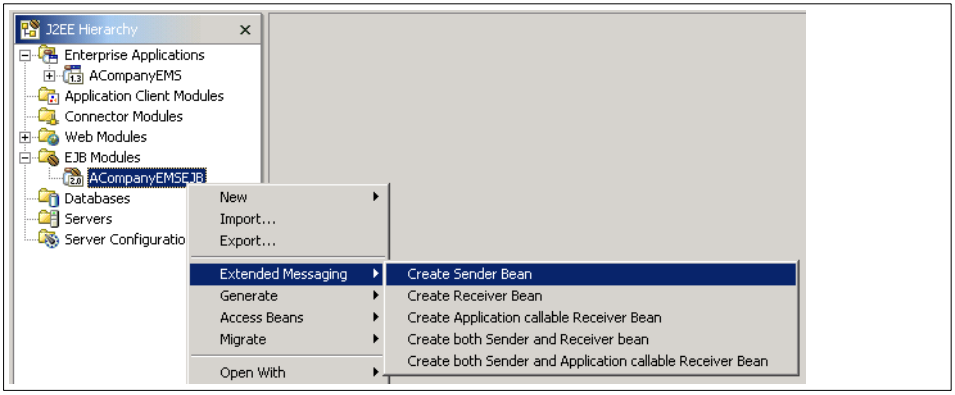


Figure 6-5 Create Sender

3. On the Create Sender bean window, click the **Create sender** button, then provide the following details:
   - Bean name: `OrderSender`
   - Default package: `com.acompany.ems.beans`

   Click **Next**.
4. On the Enterprise Beans details window, select **Local client view** as shown in Figure 6-6 on page 237. This will create local interfaces for the OrderSender session EJB so that this bean can be called efficiently from the order process flow.

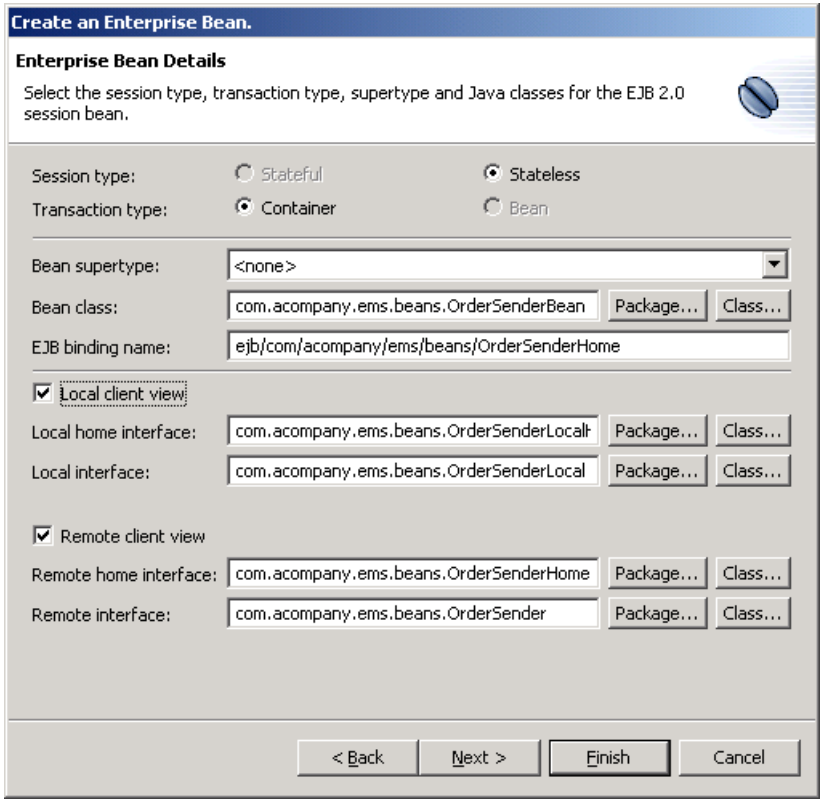


Figure 6-6 Select OrderSender details

5. Click **Finish**.
6. Fill in the port, response, and data mapping information per Figure 6-7 on page 238, then click **Next**.

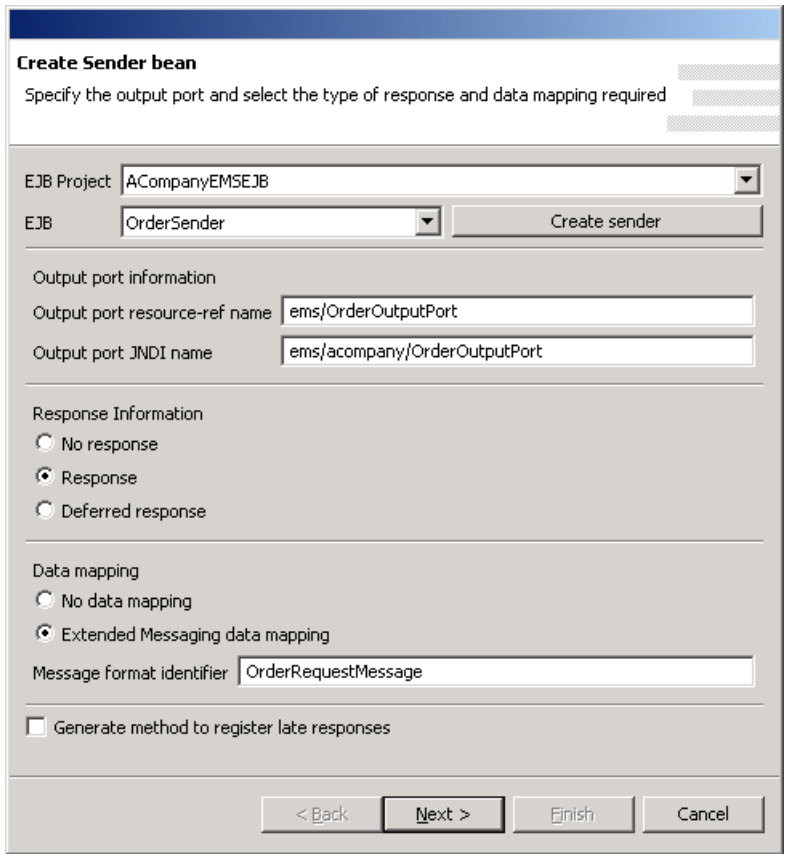


Figure 6-7 Create OrderSender bean

The output port resource-ref-name is the resource reference that will be specified in the EJB Deployment Descriptor and bound to the output port with the specified output port JNDI name.

The response selection means that when this sender bean sends a message, it will expect to immediately receive a response message back.

The message format identifier specifies an arbitrary name for the type of message that this sender bean will send. When a message is sent, the JMS message header attribute JMSType will be set to this name.

7. On the Send with response window in Figure 6-8 on page 239, select **No timeout** and **Add to local interface**.

   This adds the sendWithResponse() method to the OrderSender beans local interface so that this method is visible to the calling application.

   Click **Next**.

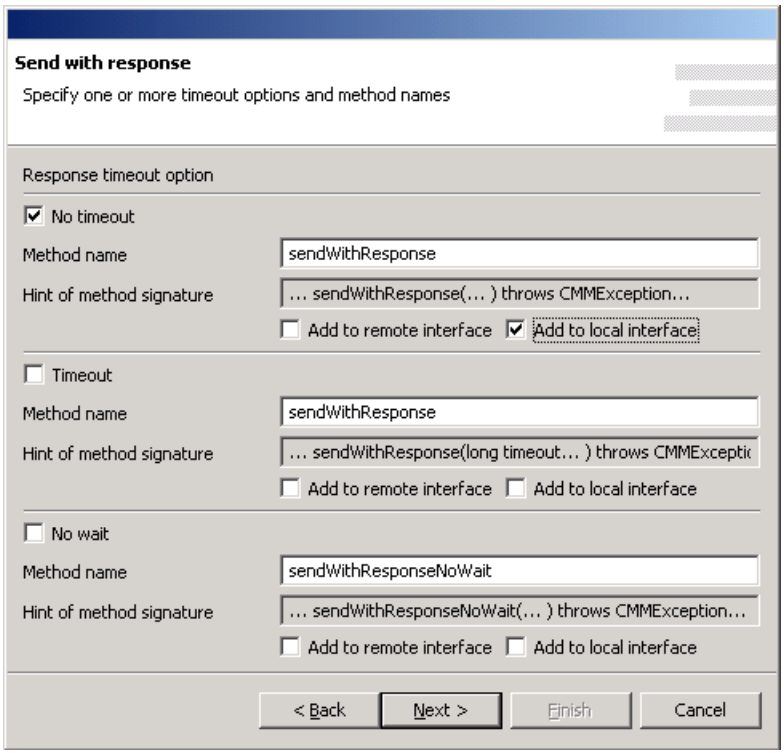


Figure 6-8 Specify the wait type for the response

8. On the data mapping window, select **Define and validate method signature**. This will define the parameters for the sendWithResponse method. The CMMFormatter class will then take these input parameters and build the JMS message to be sent. Click **Next**.
9. On the define and validate the sending method signature window, enter a return type of `java.lang.String`.

> **Note:** The Browse button only displays Java interfaces so it cannot be used for our sample.

10. The four parameters in Table 6-1 on page 240 will constitute the four parts to our message. Fill in the parameter name and parameter type input fields, and then click the **Add** button.

Table 6-1 Parameters

Parameter	Type
orderID	java.lang.String
description	java.lang.String
qty	int
price	java.lang.Double

When complete the window, should look like Figure 6-9 on page 241.

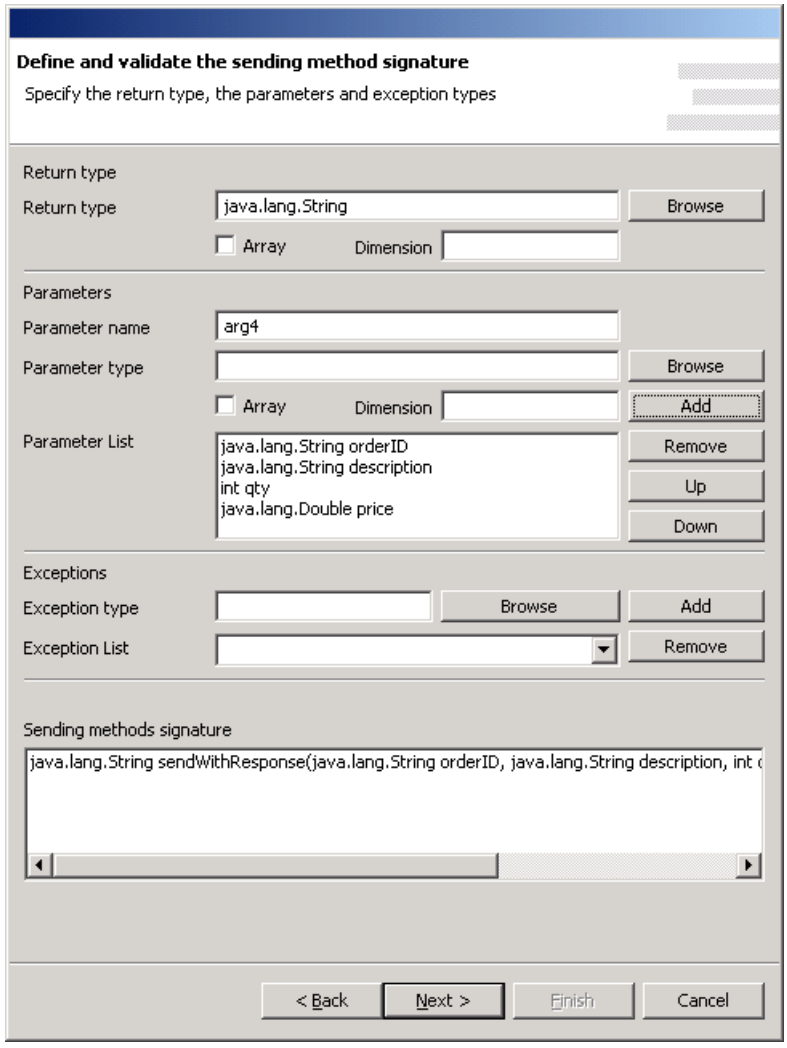


Figure 6-9 Define SendOrder method signature

11. Click **Next**.

12. The summary window will list the bean created, the method signature of any methods created, and the resource reference for the output port and message selector. Click **Finish**.

Examine OrderSender bean

We have now completed the steps for creating our OrderSender bean. You now have a stateless session bean with a sendWithResponse() method that will take

the four input parameters that were defined during wizard generation. Execution of the sendWithResponse() method will assemble the message and place the message on the queue defined by the EMSOrderOutputPort. To examine the code, do the following:

1. Expand **EJB Modules -> ACompanyEMSEJB -> OrderSender**. There are both remote and local interfaces for the session EJB because both were specified during EJB creation.
2. Double-click **OrderSenderBean**. Scroll down to the sendWithResponse() method.

   Consistent with the sending process pattern in Figure 6-1 on page 231, you will see:

   – The sender bean created based on the JNDI name of the output port.
   – A CMMFormatter created and then updated with the input parameters.
   – A message retrieved from the formatter.
   – A message type set so that the MDB can listen for messages of this type only.
   – The message sent to the queue that will be specified when the output port is configured.
   – The response is returned based on the reply queue that will be specified in the output port.

   The code listing is shown in Example 6-2.

Example 6-2 The sendWithResponse method

```
public java.lang.String sendWithResponse(java.lang.String orderID,
java.lang.String description, int qty, java.lang.Double price) throws
CMMException
   {
      // Create sender
      CMMSender sender = CMMFactory.createSender("ems/OrderOutputPort");
      try {
         // Create message factory
         MessageFactory factory = sender.getMessageFactory();
         // Create formatter
         CMMFormatter  formatter = CMMFactory.createCMMFormatter(factory);
         // Add parameters to the message
         formatter.addStringParameter(orderID);
         formatter.addStringParameter(description);
         formatter.addIntParameter(qty);
         formatter.addObjectParameter(price);
         // Get the message
         Object request = formatter.getMessage();
         // Set message type
```

```
            sender.setRequestMessageType("OrderRequestMessage");
            // Send request receive response
            Object response = sender.sendRequestReceiveResponse(request);
            // Create parser
            CMMParser  parser = CMMFactory.createCMMParser(response);
            // Process exception
            if (parser.containsException()){
               try{
                   throw parser.getException();
               }
               catch(CMMException exc){ throw exc;}
               catch(Exception exc){ throw  new CMMException("Unexpected
Exception", exc);}
            }
            // Extract response and return
            return parser.getStringParameter();
         }
         finally
         {
            sender.close();
         }
      }
```

3. Select **EJB Modules -> ACompanyEMSEJB -> OrderSender** and double-click **ResourceRef**. It will open the Deployment Descriptor for the EJB module at the resource reference. You can see the output port name of `ems/OrderOutputPort` with a JNDI name of `ems/acompany/OrderOutputPort`. We will use these later to configure the output port during runtime administration. Close the Deployment Descriptor file because it will be updated in the next step.

Create the OrderReceiver session bean

Now that we have created the sender bean, it is time to create an MDB that will receive the message, pass the message to a ProcessOrder stateless session bean, and then post a reply back to the OrderSender bean.

Create the ProcessOrder session bean

First, let's create the ProcessOrder stateless session bean. This bean can then be used in the wizards of WebSphere Studio IE during the receiver bean creation.

1. Select **File -> New -> Enterprise Bean**.
2. In the Enterprise Bean Creation window, select **ACompanyEMSEJB** for the project, then click **Next**.

3. In the Create an Enterprise Bean window, select **Session Bean**, then provide the following details:
   - Bean name: `ProcessOrder`
   - Default package: `com.acompany.ems.beans`

   Click **Next**.
4. On the Enterprise Details window, select **Local client view**, then click **Finish**.
5. The next step is to update the ProcessOrder bean by copying the method into the bean. Select **EJBModules -> ACompanyEMSEJB -> ProcessOrder** and double-click the **ProcessOrderBean** to open for editing.
6. Paste the processOrder() method from Example 6-3 into the end of the ProcessOrderBean ahead of the last '}'.

Example 6-3 processOrder method

```
public String processOrder(java.lang.String orderID, java.lang.String
description, int qty, java.lang.Double price) {
      // write out the order information to the system log. In real life
      // we would process the order through a back end system
      System.out.println("OrderID: " + orderID);
      System.out.println("Description: "  + description);
      System.out.println("Quantity: " + new Integer(qty).toString());
      System.out.println("Price: " + price.toString());
      return "Order successfully retrieved and processed by
ProcessOrderBean:processOrder";
   }
```

7. In the Outline view at the bottom left, right-click the **processOrder** method, then select **Enterprise Bean -> Promote to Local Interface**.
8. Save the file, then close.

Create the OrderReceiver bean

To create the OrderReceiver bean, follow thes steps:

1. Right-click the **EJBModules -> ACompanyEMSEJB**, then select **Enterprise Messaging -> Create Receiver Bean**.
2. Similar to what was done when creating the OrderSender bean, click the **Create Receiver bean** button and provide the following details:
   - Bean name: `OrderReceiver`
   - Default package name: `com.acompany.ems.beans`

   Click **Next**.
3. On the Enterprise Beans details window, specify the attributes as per Figure 6-10 on page 245.

- Destination Type: `Queue`
- Message selector: `JMSType='OrderRequestMessage'`
- Listener port name: `EMSOrderListenerPort`

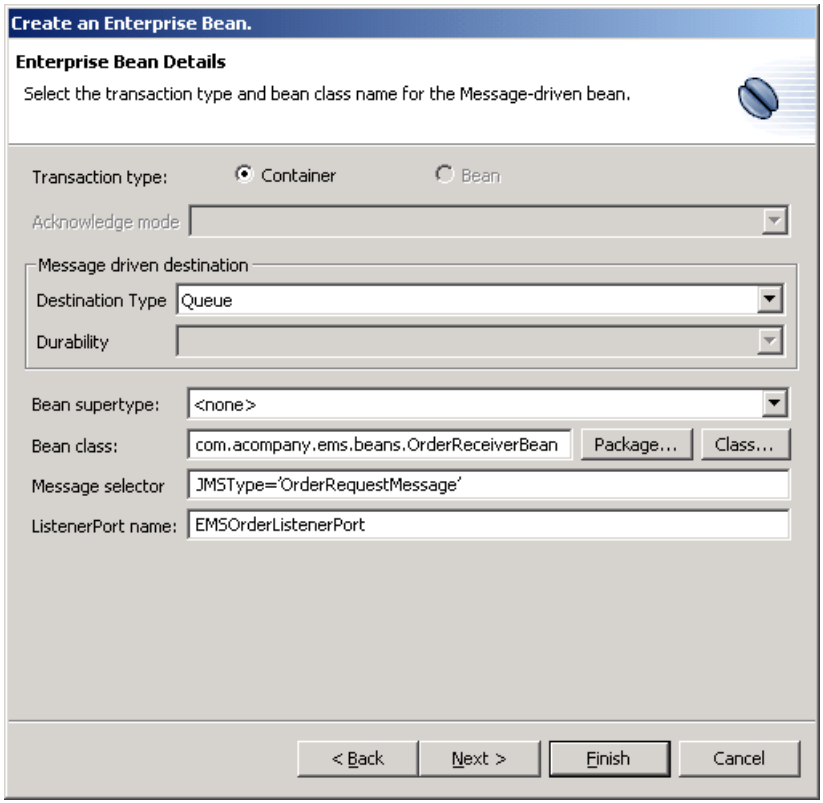


Figure 6-10 ReceiveOrder details

The Message selector specifies a filter for the messages that this message-driven bean is interested in receiving. A message selector is like an SQL “where” clause for a database query, but specifies conditions based on JMS message header attributes. In this case, we want the OrderReceiver bean to receive only messages of type OrderRequestMessage.

The listener port is a special port that you will configure later in the internal JMS server. A listener port specifies the JMS connection factory and destination to listen on for incoming messages.

4. Click **Finish**.
5. Back on the Create Receiver bean window, specify the window attributes as shown in Figure 6-11 on page 246.
   - Reply information: `Send reply`

– Extended Messaging data mapping: `OrderRequestMessage`

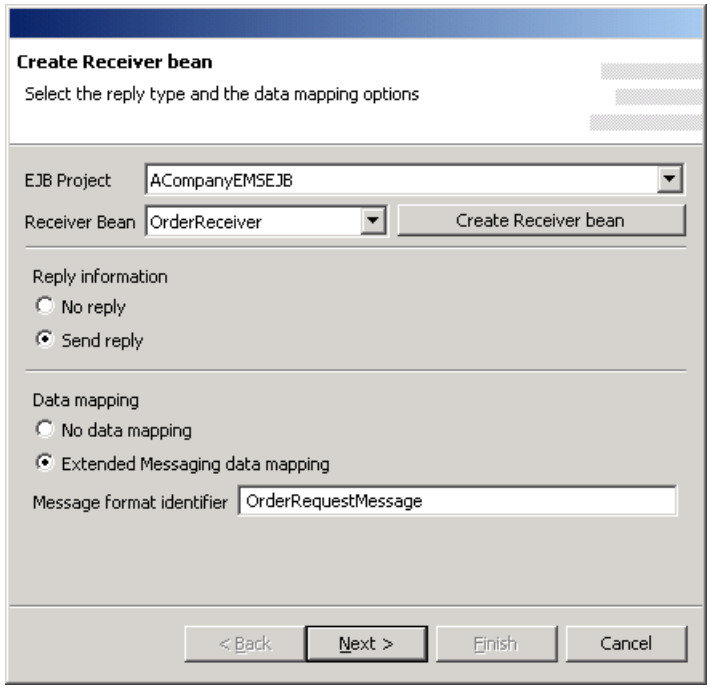


Figure 6-11 Create OrderReceiver bean

The OrderRequestMessage used for the data mapping matches the data mapping used in the OrderSender bean. Send reply is used to send a reply message back to the OrderSender bean, since it is blocking on the OrderReceiver bean reply.

6. Click **Next**.
7. On the Application EJB window, we can specify a session bean that we want to call from the onMethod method of the OrderReceiver. In this case we will call the ProcessOrder's processOrder() method. Fill in the parameters according to Figure 6-12 on page 247.
   – Select **Call method on the remote or local interface of target ejb** to call target EJB.
   – Application EJB Project: `ACompanyEMSEJB`
   – Application EJB: `ProcessOrder`
   – Select **Local interface**
   – Application method: `processOrder()`

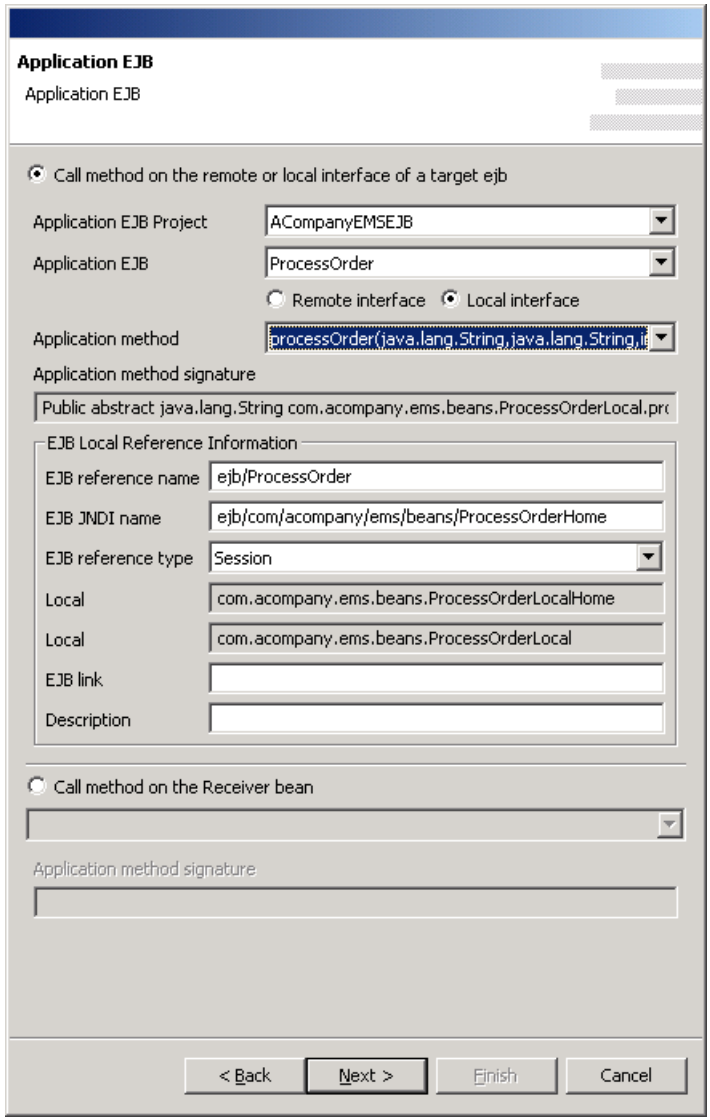


Figure 6-12 Specify EJB for onMessage() to call

8. Click **Next**.
9. At the review window, click **Finish**.

We now have the ProcessOrder, Order Sender, and OrderReceiver beans created. If you select the '**+**' next to the OrderReceiver bean, you will notice that there are no remote or local interfaces. This is because the OrderReceiver bean

is an MDB. Double-click the **OrderReceiverBean** and scroll down to the onMessage() method as seen in Example 6-4.

Example 6-4 onMethod() of OrderReceiver

```
public void onMessage(javax.jms.Message msg) {
   try {
      // Select based on the message type
      if ("OrderRequestMessage".equals(msg.getJMSType())){
         try {
            // Create reply sender
            CMMReplySender replySender = CMMFactory.createReplySender(msg);
            MessageFactory factory = replySender.getMessageFactory();
            // Create formatter
            CMMFormatter  formatter = CMMFactory.createCMMFormatter(factory);
            try {
               // Create parser to extract parameters from the message
               CMMParser  parser = CMMFactory.createCMMParser(msg);
               // Extract parameters
               java.lang.String param0 = parser.getStringParameter();
               java.lang.String param1 = parser.getStringParameter();
               int param2 = parser.getIntParameter();
               java.lang.Double param3 =
(java.lang.Double)parser.getObjectParameter();
               // Create EJB
               javax.naming.Context initialContext =  new
javax.naming.InitialContext();
               com.ibm.itso.ems.order.ProcessOrderLocalHome home =
(com.ibm.itso.ems.order.ProcessOrderLocalHome)initialContext.lookup("java:comp/
env/ejb/ProcessOrder");
               com.ibm.itso.ems.order.ProcessOrderLocal obj = home.create();
               // Invoke target method
               String reply = obj.processOrder(param0, param1, param2,
param3);
               // Set reply
               formatter.addStringParameter(reply);
            }
            catch(Exception exc) {
               // Set exception
               formatter.setException(exc);
            }
            Object reply = formatter.getMessage();
            // Send reply
            replySender.sendReply(reply);
            return;
         }
         catch(Exception exc) {
            // Handle exception
            CMMFactory.handleException(msg, exc);
```

```
            return;
          }
       }
    }
    catch (JMSException exc){
       // Failed to get message type
    }
    CMMFactory.handleMessage(msg);
    }
```

If you look at the onMessage() method, the first step is to verify that the message is of type OrderRequestMessage. Then a CMMReplySender is built to send the reply back to the OrderSender bean. A CMMParser is used to parse the message. The parser will parse the message according to the message format so it is important to use the same format as was specified in the OrderSender. After parsing of the message, the ProcessOrder session bean is instantiated, and called via its processOrder method. Finally, the sendReply method is called to send the reply message to the reply queue. Remember the OrderSender is waiting for this reply message.

Note: We have just built a round-trip send, receive, and reply message pattern. All the messaging plumbing was handled by the tooling in WebSphere Studio IE and the corresponding EM runtime support in WebSphere Enterprise. The programmer's responsibility is to write the business logic in the ProcessOrder bean and not focus on plumbing. This also promotes a good separation of business logic from messaging implementation.

Generate deployed code

The last step is to generate the stubs and skeletons for the EJBs:

1. Right-click the **ACompanyEMSEJB** project in the J2EE Hierarchy view, then select **Generate -> Deploy and RMIC Code**.
2. Click **Select All**, then **Finish**.

We are now ready to configure the WebSphere Test Environment and run the sender and receiver beans.

6.4 Unit test environment

The steps to run the sample are to create a new server, configure Extended Messaging within the server, and run the sample.

1. Follow the steps from "Development environment" on page 667 to create a new EE Test environment for the sample. You can skip the steps for adding

the enterprise applications, ACompany and ACompanyProcess, if you only want to test the Extended Messaging application.

2. Right-click the **ACompanyServer** entry in the Server Configuration view under the Servers folder and add the ACompanyEMS application.

WebSphere Studio IE does not support the Extended Messaging configuration using the server configuration tool.

The following section describes how to configure Extended Messaging for WebSphere Enterprise using the server configuration tool for basic JMS configuration and Administrative Console for Extended Messaging configuration.

If you want to use the Administrative Console to configure your test server all the way through without the server configuration tool, then skip the next section and follow the steps in 6.5.2, “Configuration with JMS Embedded Messaging” on page 259 after you have started the test server.

6.4.1 Configure Extended Messaging

It is now necessary to configure a QueueConnectionFactory, destinations, output port, and the internal JMS server.

1. Switch to the Server perspective and double-click the **ACompanyServer** in the Server configuration view to open the server configuration for the server.
2. Select the **JMS** tab and open the **Node Settings** part.
3. Create a new JMS Connection Factory by clicking the **Add** button next to the the WASQueueConnectionFactory entries, then specify the following settings:
   - Name: `ACompanyQCF`
   - JNDI Name: `jms/acompany/QCF`
   - Node: `localhost`
   - Server Name: `server1`
   - Component-managed authentication alias: `jmsuser_alias`
   - Container-managed authentication alias: `jmsuser_alias`
4. Create two queues for the server by clicking **Add** next to the WASQueue entries. Use the following details for the new entries:
   - First entry:
     - Name: `OrderRequestQueue`
     - JNDI Name: `jms/acompany/requestQ`
   - Second entry:
     - Name: `OrderReplyQueue`
     - JNDI Name: `jms/acompany/replyQ`

At this point the server configuration should look like Figure 6-13.

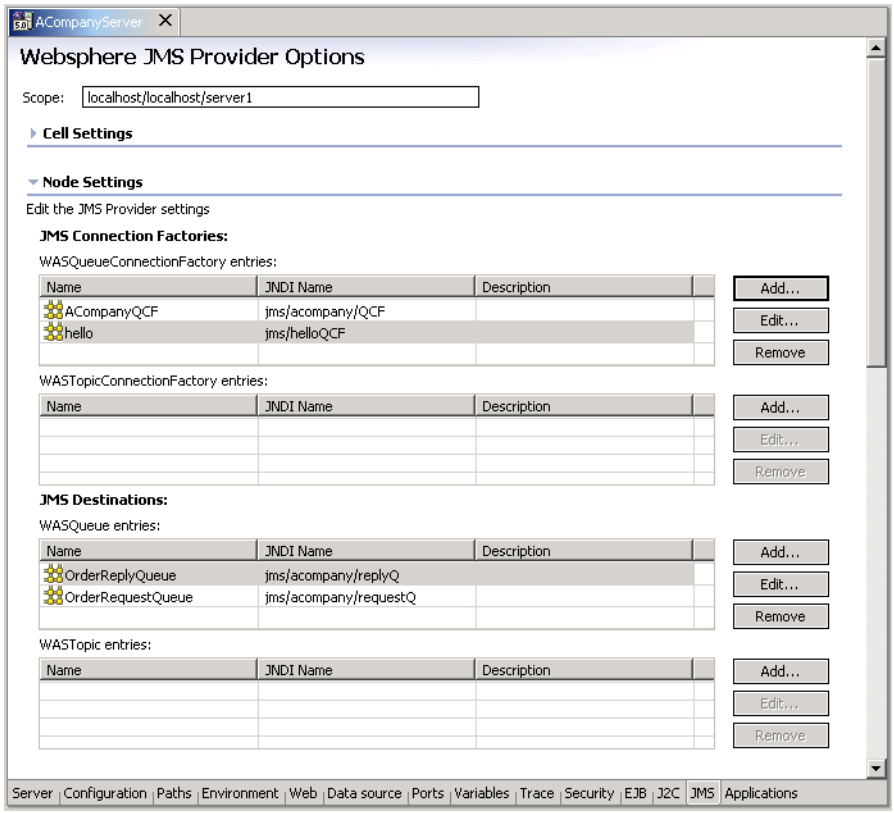


Figure 6-13 Queue Connection Factory and Queue destinations configuration

5. Scroll down to the Server Settings section on the JMS window.
6. Add two queue names to the list by clicking the **Add** button and typing the following two entries:

   `OrderRequestQueue`
   `OrderReplyQueue`
7. Make sure that the Initial State is set to START.
8. Switch to the **EJB** tab to configure the listener ports.
9. Click **Add** next to the listener ports, then specify the following settings:
   - Name: `EMSOrderListenerPort`
   - Connection factory JNDI name: `jms/acompany/QCF`
   - Destination JNDI name: `jms/acompany/requestQ`

10. Switch to the Configuration tab, and make sure that the Administrative Console is enabled for the server.
11. Save the server configuration and close the file.
12. Unfortunately, we cannot configure the output port for Extended Messaging in the server configuration tool. We have to do that using the WebSphere Administrative Console.
13. Start the test server. In the Servers view, right-click the **ACompanyServer** and select **Start**. The server is started when you see the line:

    ```
    Server server1 open for e-business in the console.
    ```

14. Launch the Administrative Console by going to the Servers tab and clicking **ACompanyServer** and selecting **Run administration console**.
15. Log into the console, using the server ID and password you set for the test server.
16. In the WebSphere Administrative Console, select **Resources -> Extended Messaging Provider -> Output Port**.
17. On the Output Port window, select **New**.
18. On the New window, specify the following:
    - Name: `EMSOrderOutputPort`
    - JNDI Name: `ems/acompany/OrderOutputPort`
    - JMS Connection Factory JNDI Name: Select **jms/acompany/QCF** from the drop-down list
    - JMS Destination JNDI Name: Select **jms/acompany/requestQ** and click **Add** to select this destination queue.
    - Response JMS Connection Factory JNDI Name: Select **jms/acompany/QCF** from the drop-down list.
    - JMS Destination JNDI Name: Select **jms/acompany/replyQ** from the drop-down list.

    Click **OK**.
19. Save the configuration for WebSphere.

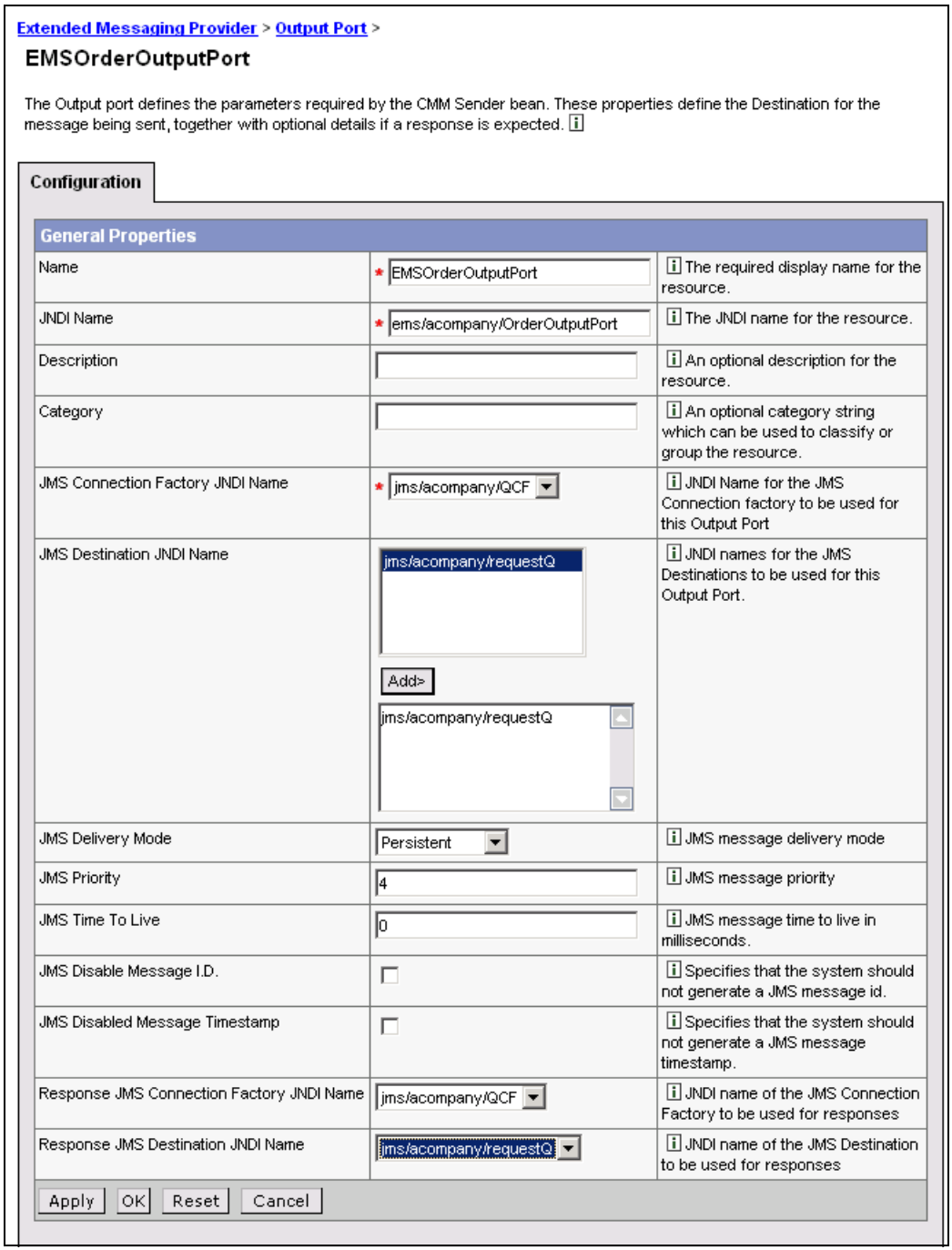


Figure 6-14 Output port config

6.4.2 Test the sample

After the configuration, it is necessary to refresh, add the ACompanyEMS project, and then restart the server.

Refresh, update and restart the server

The following steps are followed to restart the server:

1. On the Server perspective in the Navigator view, right-click the **Servers** project and select **Refresh**.
2. In the Server Configuration pane in the bottom left, right-click **ACompanyServer**, then select **Add -> ACompanyEMS**.
3. Go to the Servers view at the bottom of the Server perspective. Right-click **ACompanyServer**, and select **Restart**. This will also republish your project and restart the server.

You should see a clean log with bindings for OrderRequestQueue, OrderReplyQueue, ACompanyQCF, and EMSOrderOutputPort.

Start the IBM Universal Test Client

We will use the EJB test client provided by WebSphere Studio IE to test our application.

1. Switch to the J2EE perspective then the J2EE view in WebSphere Studio IE.
2. Select **EJB Modules -> ACompanyEMSEJB**, right-click **OrderSender**, and select **Run on Server...**
3. Select **Test EJB** local interface from the window, then click **Finish**. The IBM Universal Test Client should start.

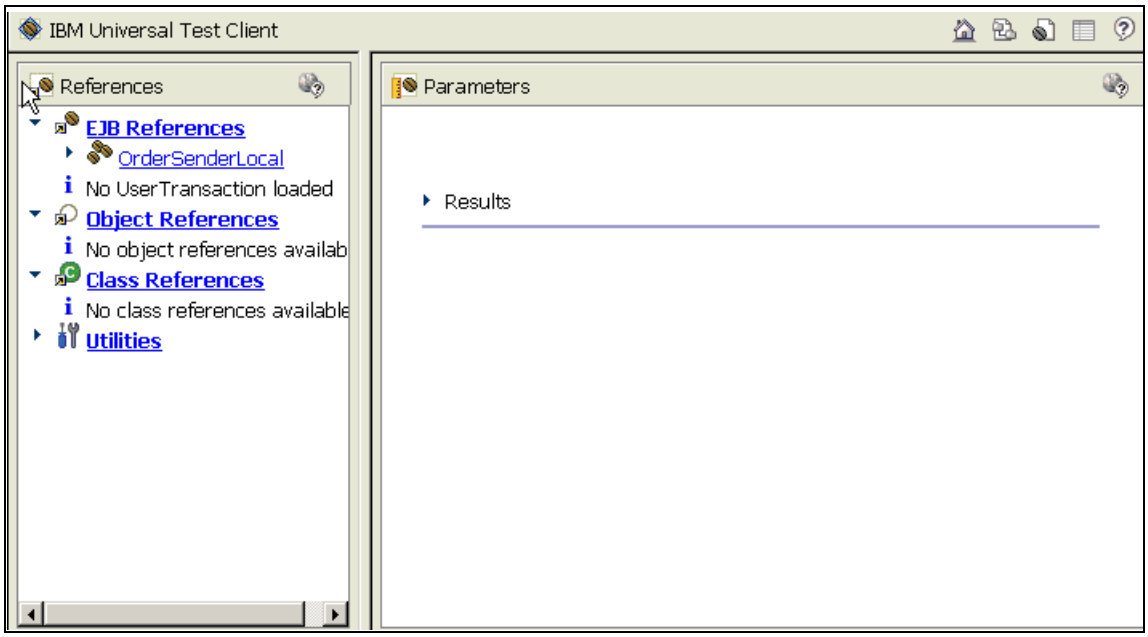


Figure 6-15 IBM Universal Test Client

Run the sample

Follow these steps to use the test client to run the sample:

1. From the References pane in the test client, select **OrderSenderLocal -> OrderSenderLocalHome -> OrderSenderLocal create()**.
2. On the right pane, click **Invoke**, then click **Work with Object** on the same window.

   The test client should now look like Figure 6-16 on page 256.

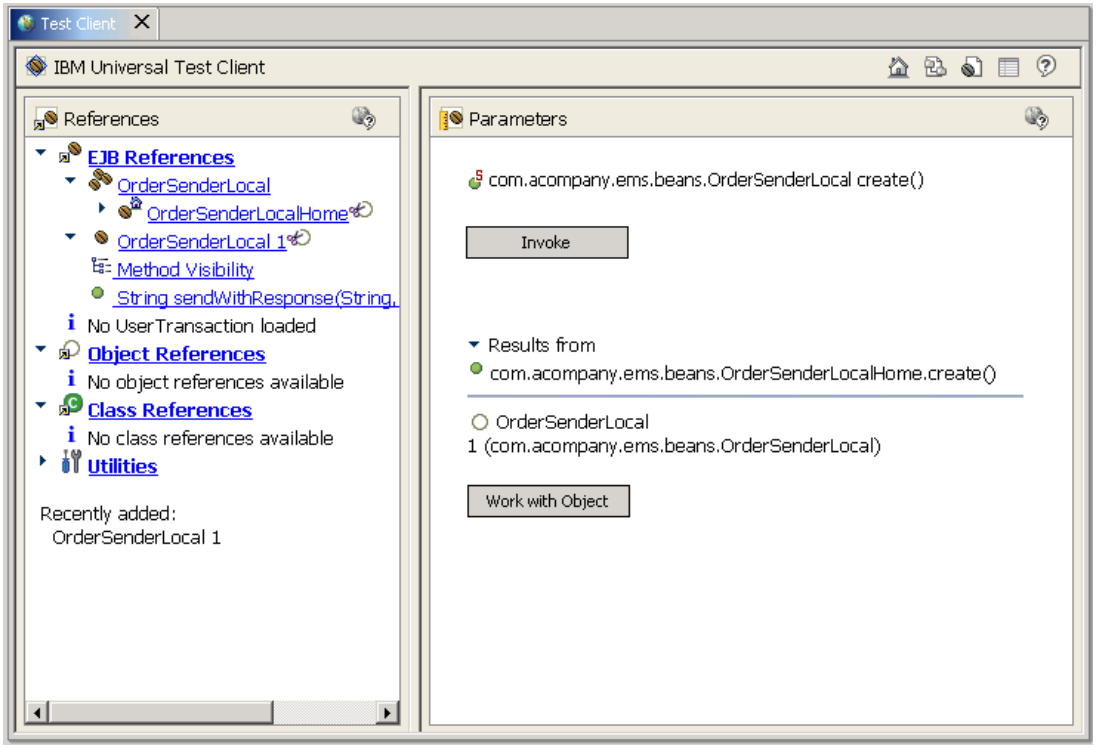


Figure 6-16 Test client ready to run the local interface

3. On the References pane, select the **String sendWithResponse()** method.
4. For testing, the parameters on the right can be any valid parameter, but for our sample application, they will represent an orderId, description, quantity, and price. The window should look like the one shown in Figure 6-17 on page 257.

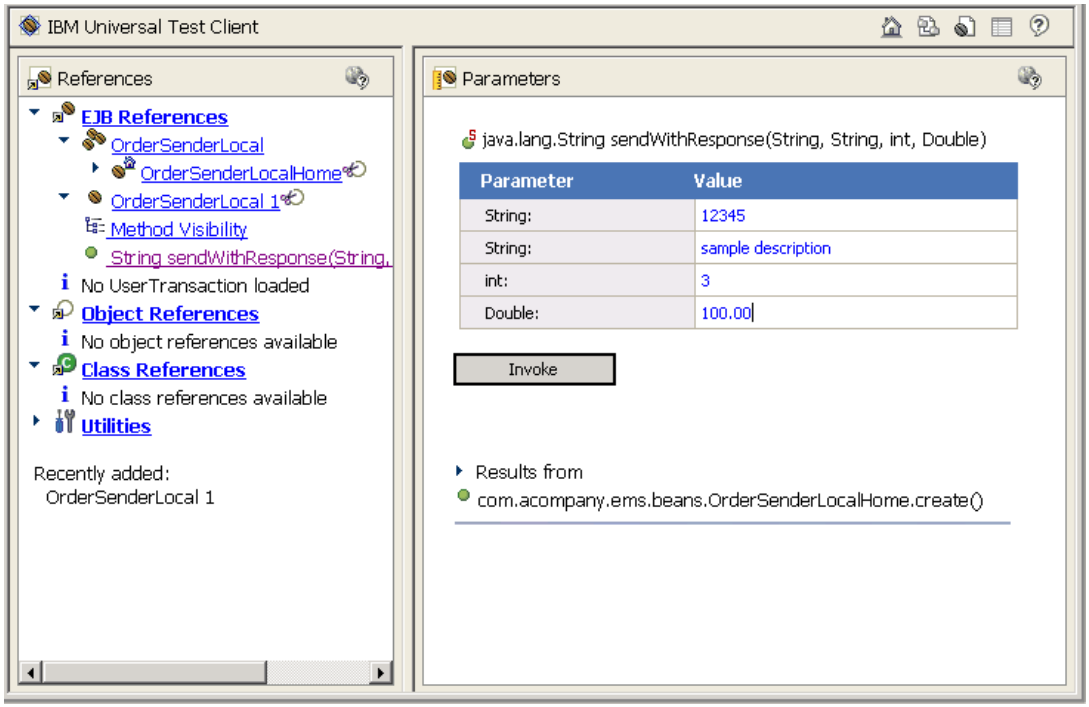


Figure 6-17 Prepare to run the sendWithResponse method

5. Click the **Invoke** button.
6. In the console, you will see the output from the ProcessOrder bean. Remember from Figure 6-4 on page 235 that we have sent a message from the OrderSender bean. Then the OrderReceiver MDB picked up the message from the request queue and sent the parsed details to the ProcessOrder stateless session bean. Finally, the OrderReceiver sends a "`process order completion`" message back on the reply queue.
7. In the Universal Test Client Parameters pane, you should see the response message "`Order successfully retrieved and ...`" at the bottom. This shows that the reply message was picked up by the sender bean.

Note: Since the OrderSender bean created by the tooling in WebSphere Studio IE is a stateless session bean, it is easy to use the Universal Test Client to test the bean without writing a test harness to test the sendWithReply method.

6.5 Configuration

WebSphere has three options for configuring messaging solutions. Embedded Messaging is provided in the base and provides single node messaging infrastructure for both point to point and publish and subscribe. External messaging providers are also supported including WebSphere MQ and non-IBM messaging products through the JMS interface.

6.5.1 Comparison of WebSphere MQ and Embedded Messaging

For most customers, Embedded Messaging will be used for development and an external JMS provider will be used for production. The EM code developed in WebSphere Studio IE will run unchanged in each environment. For more details about the Embedded Messaging features of WebSphere V5, and comparisons to WebSphere MQ and external JMS providers, see *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series,* SG24-6195.

Embedded Messaging features

Embedded Messaging is provided as a component of WebSphere Application Server V5. In 6.5.2, “Configuration with JMS Embedded Messaging” on page 259 we define Queue Connection Factories and destinations. When the server is started, a queue manager, request queue, and reply queue is started.

Some key features include:

- Support for JMS 1.0.2
- XA support
- Easy installation included with WebSphere installation

Limitations include:

- The queue manager runs on a single WebSphere node in a network deployed installation, so there is a single point of failure for that node.
- Queue manager clusters are not supported.
- Messages can only be sent and received within a WebSphere environment.
- JMS is the only supported API, so there is no support for non-Java clients.

WebSphere MQ features

WebSphere MQ can be used as an external JMS provider. WebSphere MQ is fully configurable within the WebSphere Administrative Console but does not have the limitations discussed in “Embedded Messaging features” on page 258. Typically WebSphere MQ will be used in production environments to provide clustering and high-availability configurations.

6.5.2 Configuration with JMS Embedded Messaging

This section provides instructions to configure Extended Messaging using the JMS Embedded Messaging facility in WebSphere.

The information in this section can also be used to configure the test server in WebSphere Studio IE.

Configure QueueConnectionFactory

For the test environment, we will be using the embedded JMS provider in WebSphere. The QueueConnectionFactory will be used when we define the output port. Defining a QueueConnectionFactory will start a WebSphere queue manager when the server is restarted.

1. Start the application server if it is not running.

   Note: You can use the default server, server1, for the following configuration. If you prefer to define your own server, then follow the steps from “Runtime environment” on page 675.

2. Start the Administrative Console, and log on with a valid user.
3. In the Administrative Console, expand **Resources** and select **WebSphere JMS Provider**. This represents the Embedded Messaging provider.
4. At the bottom of the WebSphere JMS Provider window, select **WebSphere QueueConnection Factories**.
5. On the WebSphere Queue Connection Factories window, select **New**.
6. Specify the following settings as seen in Figure 6-18 on page 260:
   - Name: `ACompanyQCF`
   - JNDI Name: `jms/acompany/QCF`

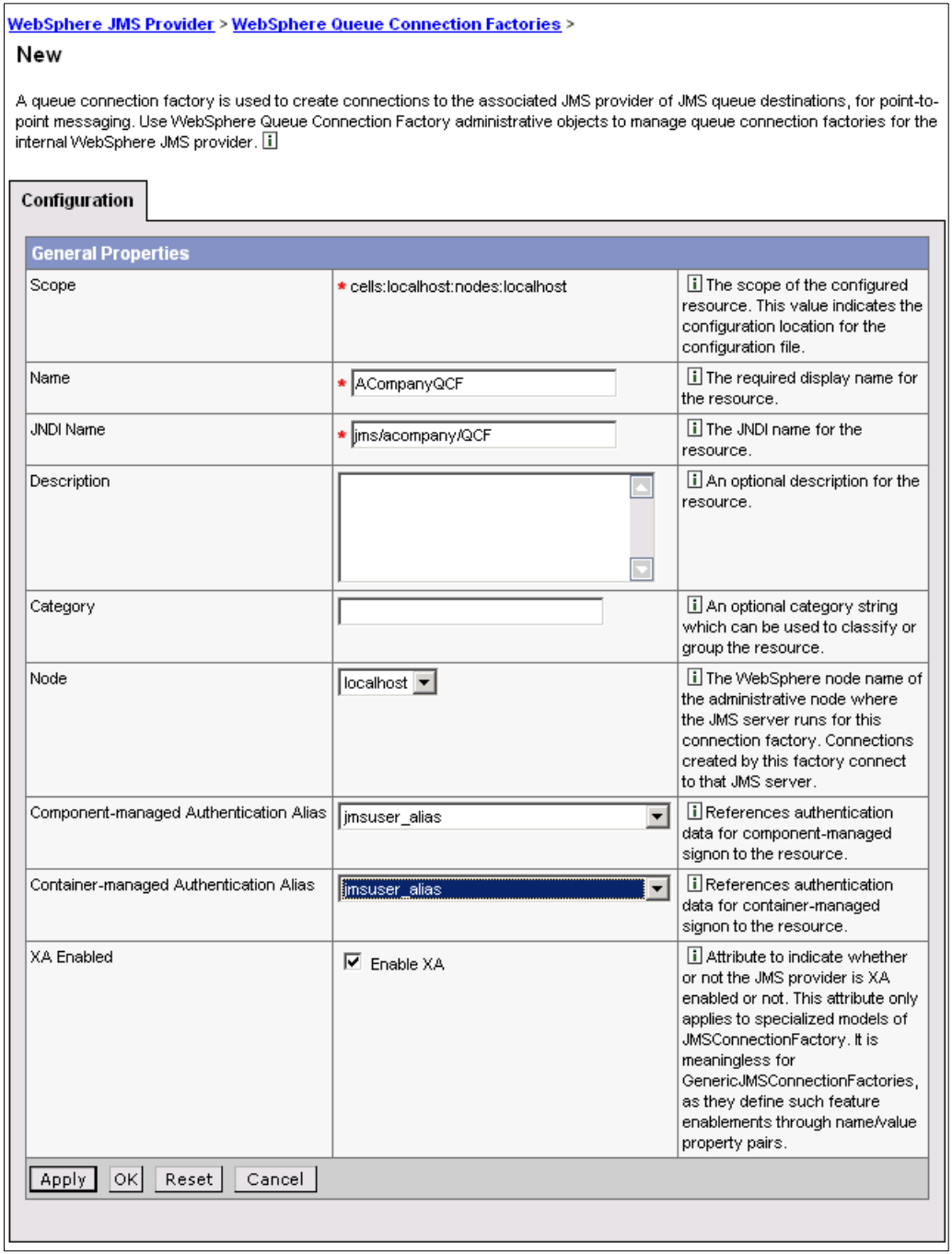


Figure 6-18 Specify QueueConnectionFactory

Note: In this particular sample, ACompanyEMS, we do not use security so you can leave the two authentication alias fields empty, although the scenario sample for the book needs security to be enabled or the QueueConnectionFactory will fail. If you want to run the scenario sample, you will have to specify the aliases for this item. You can do it right now or leave it for later.

If you do it at this point, follow the steps from Appendix B, “Sample scenario” on page 665 to create the users in your system and the necessary J2C authentication aliases.

7. At the end, select **OK**.

Configure queue destinations

Follow these steps to configure queue destinations for the EM sample:

1. Select **Resources -> WebSphere JMS Provider**.
2. At the bottom of the WebSphere JMS Provider window, select **WebSphere Queue Destinations**.
3. On the WebSphere Queue Destinations window, select **New**.
4. On the New window, specify the following:
   - Name: `OrderRequestQueue`
   - JNDI Name: `jms/acompany/requestQ`

   Click **OK**.

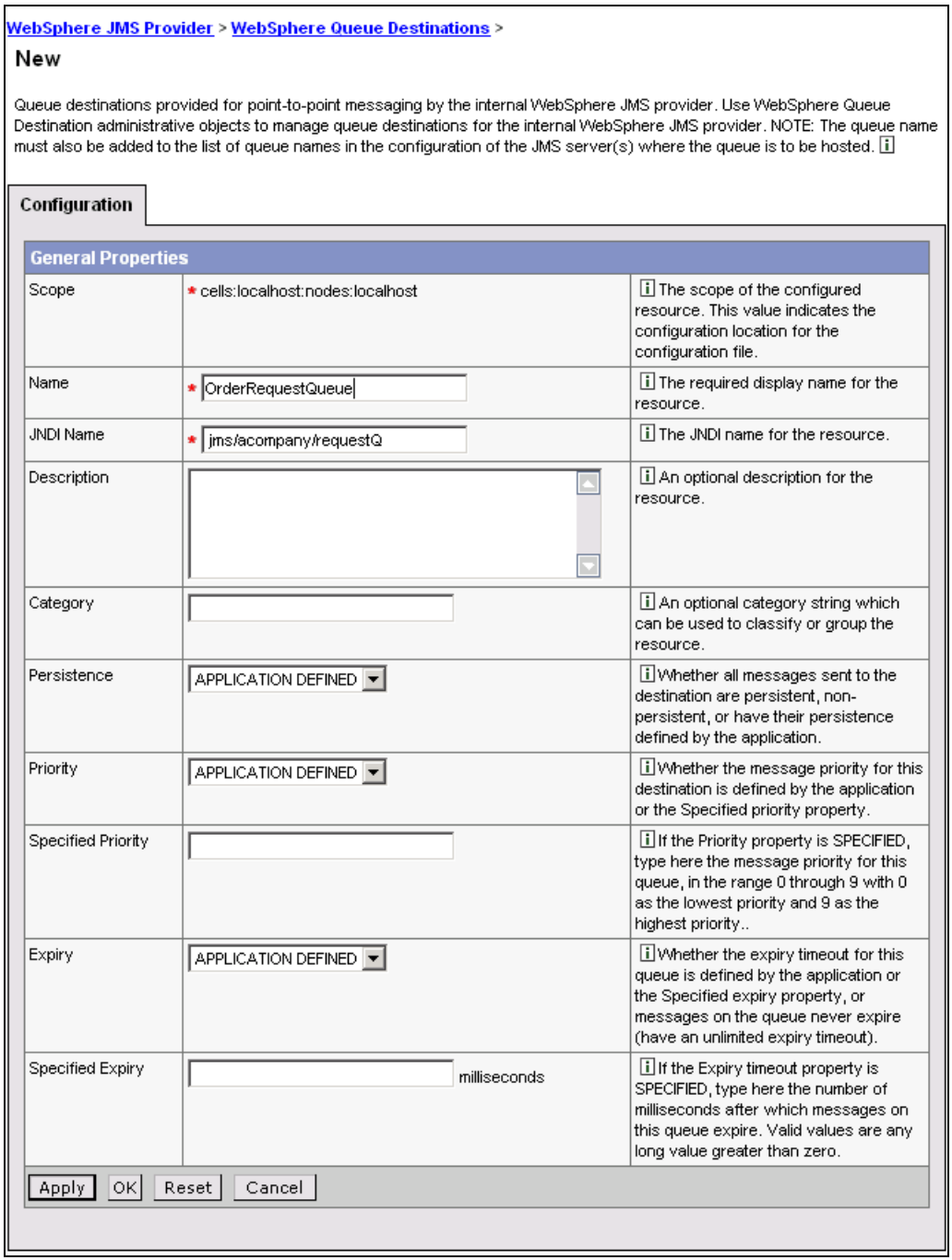


Figure 6-19 Create queue destination

5. Define the reply queue similar to the request queue by repeating the previous steps using the following details:
   - Name: `OrderReplyQueue`
   - JNDI Name: `jms/acompany/replyQ`

   Click **OK**.

Configure the output port

Remember that the output port is an abstraction for the QueueConnectionFactory, the request queue, and reply queue. If it is desired to have a message sent to multiple destinations, they are specified in the output port as well.

1. In the WebSphere Administrative Console, select **Resources -> Extended Messaging Provider**, then select **Output Port**.
2. On the Output Port window, select **New**.
3. On the New window, specify the following:
   - Name: `EMSOrderOutputPort`
   - JNDI Name: `ems/acompany/OrderOutputPort`
   - JMS Connection Factory JNDI Name: Select **jms/acompany/QCF** from the drop-down list.
   - JMS Destination JNDI Name: Select **jms/acompany/requestQ** and click **Add** to select this destination queue.
   - Response JMS Connection Factory JNDI Name: Select **jms/acompany/QCF** from the drop-down list.
   - Response JMS Destination JNDI Name: Select **jms/acompany/replyQ** from the drop-down list.

   Click **OK**.
4. Save the configuration for WebSphere.

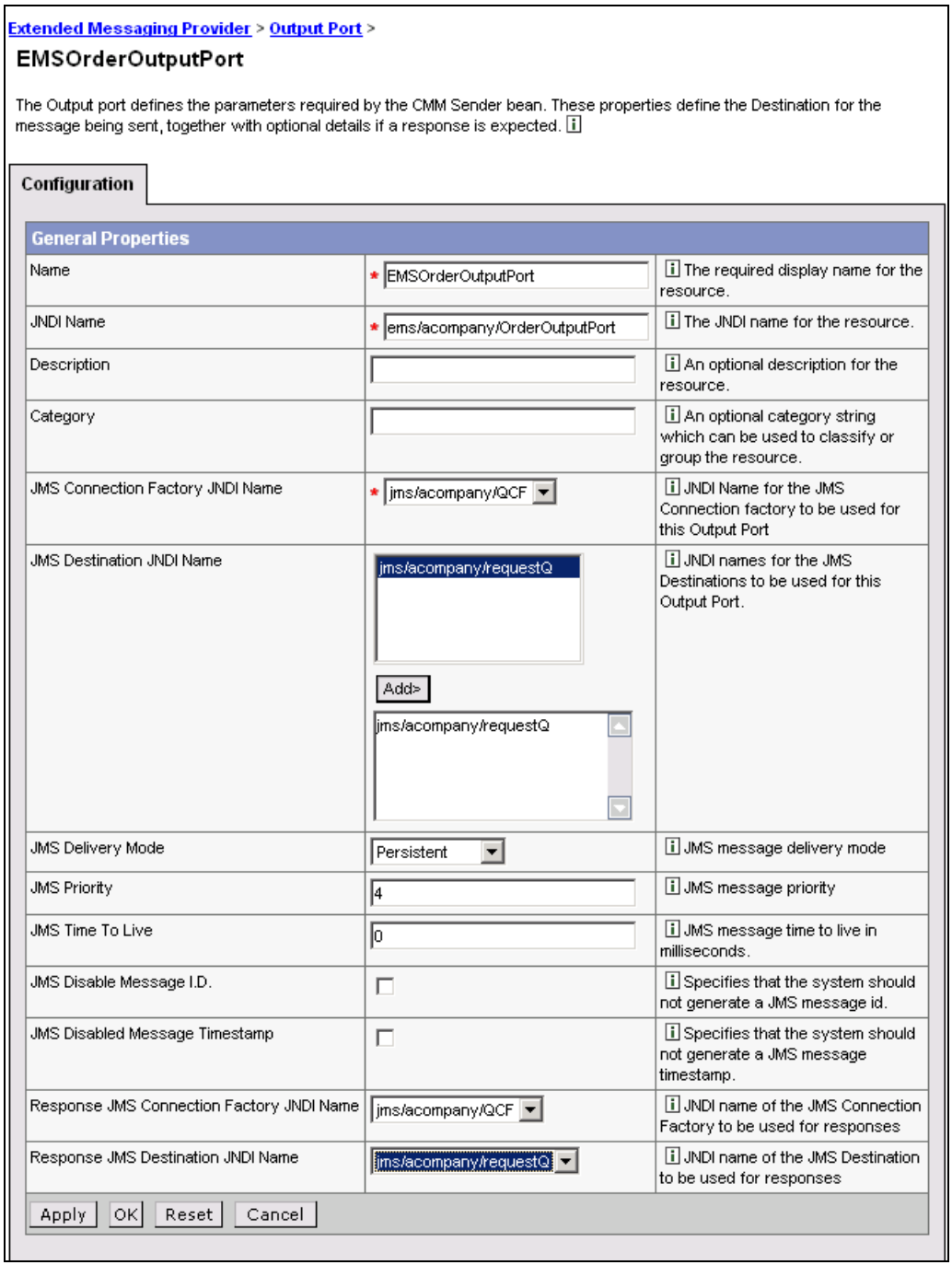


Figure 6-20 Output port config

Configure message listener service

In the Administrative Console, expand servers on the left-hand side:

1. Select **Servers -> Application Servers**.
2. In the Application Servers window, select **server1**.

   Note: If you decided to use your own server instead of the default, server1, then you will need to use a different server name in this step.

3. On the Configuration tab, select **Message Listener Service** at the bottom of the window.
4. On the Message Listener Service window, select **Listener Ports**.
5. Select **New** on the Listener Ports window, then specify the following:
   - Name: `EMSOrderListenerPort`
   - Connection factory JNDI name: `jms/acompany/QCF`
   - Destination JNDI name: `jms/acompany/requestQ`

   Click **OK**.

   Your window should look like Figure 6-21 on page 266. Unfortunately, you have to type the JNDI entries into the text fields as opposed to selecting them from a combo box.

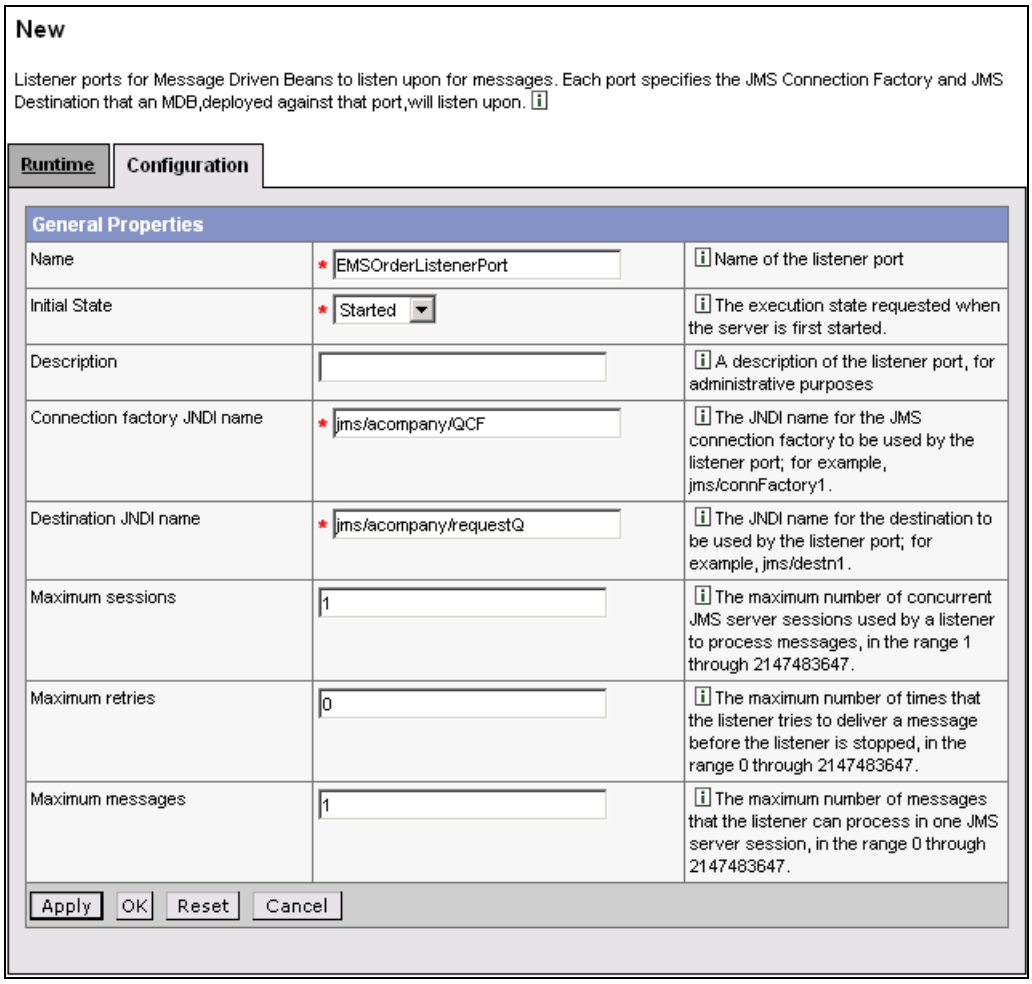


Figure 6-21 Configure Listener Port for MDB

Configure the internal JMS server

Finally, we will configure the internal JMS server. This defines the queues to the internal server so that they are started when the server starts.

1. Select **Servers -> Application Servers** then select **server1** from the server list.

   Note: If you decided to use your own server instead of the default, server1, then you will need to use a different server name in this step.

2. On the Server window, select **Server Components**.

3. On the Server Components window, select **JMS Servers**.
4. Change the initial state to Started if it is not already set.

> **Note:** You will have to restart the server if the original state for the JMS server was not started.

5. On the Internal JMS Server window, go to the text area for queue names and add two new entries at the end of the list putting each in a new line, as follows:

   ```
   OrderRequestQueue
   OrderReplyQueue
   ```

   Click **OK**.

   The Internal JMS Server window should look like Figure 6-22.

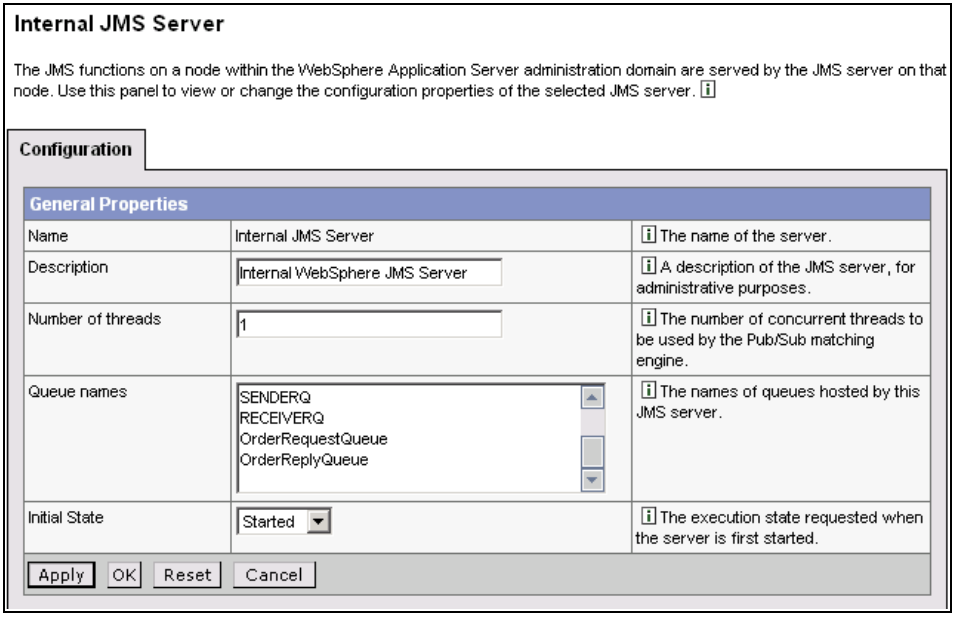


Figure 6-22 Configure internal JMS server

6. Save the configuration for WebSphere.
7. Log out and close the Administrative Console.
8. Restart the application server if you had to change the initial state for the JMS server.

At this point you are ready to deploy the sample for Extended Messaging. For deployment, follow the steps in 6.6, “Deployment” on page 273.

6.5.3 Configuration with WebSphere MQ as the JMS provider

Configuration with an external JMS provider such as WebSphere MQ is similar to the configuration performed in 6.5.2, “Configuration with JMS Embedded Messaging” on page 259. We will use the same JNDI names for the administered objects, but the definitions are slightly different.

Installation

When installing WebSphere Enterprise with an external JMS provider it is best to install the JMS provider first. For this redbook, we used WebSphere MQ 5.3.1. The steps used for installation are:

1. Install WebSphere MQ. During the install you can create a default queue manager. After the install, create two new queues for the request and reply queues. For convenience, you can name them requestQ and replyQ. Documentation for defining new queue managers and queues can be found in the WebSphere MQ help subsystem.
2. Install WebSphere Application Server V5.

   When presented with the window for Embedded Messaging, select the client only and not the server.
3. Install WebSphere Application Server Enterprise.

   Embedded Messaging will be grayed out because the client has already been installed.

Note: You can install WebSphere Application Server as part of the WebSphere Enterprise install, but you will not be given the option of installing only the client.

4. Start the application server.
5. Start the Administrative Console and log in.
6. Make sure the WebSphere environment variables point to the right location:
   a. Select **Environment -> Manage WebSphere Variables**.
   b. Adjust the MQJMS_LIB_ROOT and MQ_INSTALL_ROOT as necessary. See Figure 6-23 on page 269.

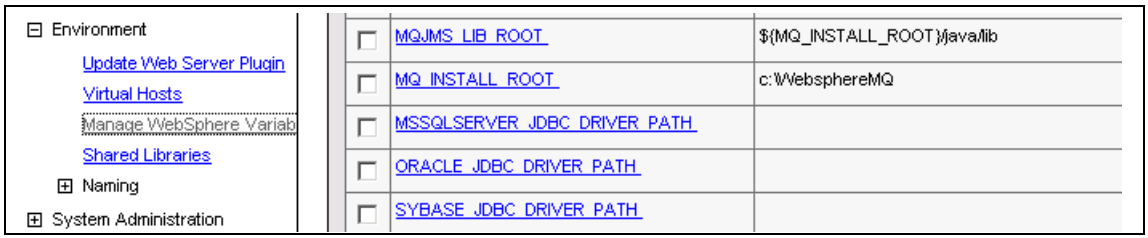


Figure 6-23 Set WebSphere MQ install variables

Configure QueueConnectionFactory

To configure the QueueConnectionFactory for WebSphere MQ, perform the following steps:

1. In the Administrative Console, select **Resources -> WebSphere MQ JMS Provider**.
2. At the bottom of the WebSphere MQ JMS Provider window, select **WebSphere MQ Queue Connection Factories**.
3. On the WebSphere MQ Queue Connection Factories window, select **New**.
4. Specify the following, as shown in Figure 6-24 on page 270:
   - Name: `ACompanyQCF`
   - JNDI Name: `jms/acompany/QCF`
   - Queue Manager: Queue manager name specified in WebSphere MQ.
   - Host: The host name of the machine where the queue manager is running.
   - Port: Port number the MQ listener is running on.
   - Channel: The server connection channel defined for the queue manager.
   - Transport type: Client or bindings; here we used `BINDINGS`.

   Click **OK**.

Note: The client transport uses TCP/IP and allows the queue manager to reside on a separate physical machine. This requires that the WebSphere MQ client be installed on the local machine. The bindings use the native interface and require that the queue manager be co-located with the WebSphere instance.

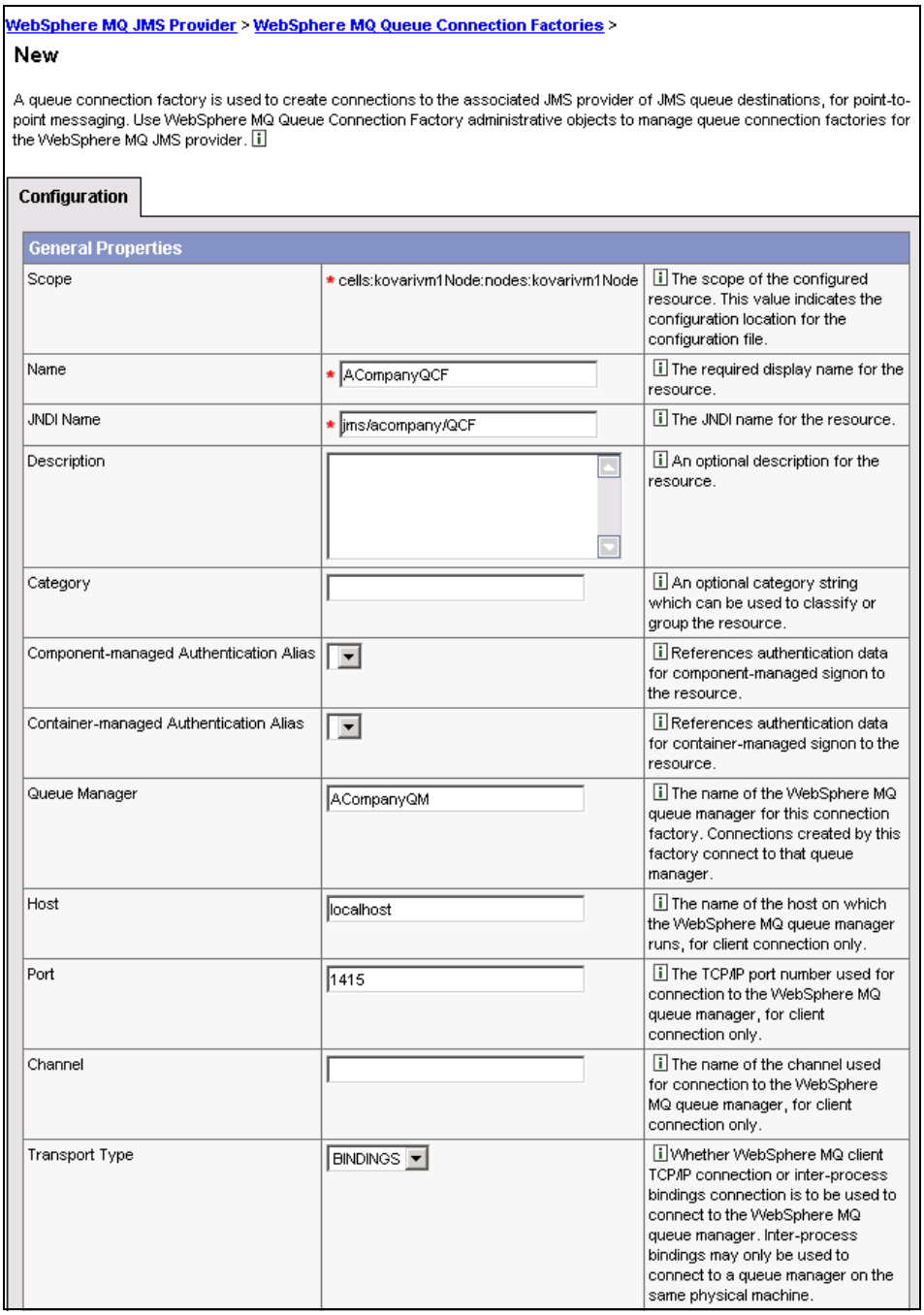


Figure 6-24 Configure queue manager

Configure MQ style queue destinations

Unlike using the embedded JMS provider, the queue destinations must first be configured in WebSphere MQ. To do this, see the help for WebSphere MQ. To configure the queue definitions for the WebSphere Application Server Enterprise runtime, do the following:

1. Go back to the WebSphere MQ JMS Provider window.
2. At the bottom of the WebSphere MQ JMS Provider window, select **WebSphere MQ Queue Destinations**.
3. On the WebSphere Queue Destinations window, select **New**.
4. On the New window, specify the following:
   - Name: `OrderRequestQueue`.
   - JNDI Name: `jms/acompany/requestQ`.
   - Base Queue Name: `requestQ`, or the queue name defined in MQ.
   - Base Queue Manager Name: Queue manager name you are using in MQ.
   - Target Client: `JMS`.
   - Queue Manager Host: `localhost` if local, or the host name where the queue manager is running.
   - Queue Manager Port: The listener port for the queue manager.
   - Server Connection Channel Name: Channel name defined for the queue manager.

   Then click **OK**. See Figure 6-25 on page 272.

		resource. This value indicates the configuration location for the configuration file.
Name	* OrderRequestQueue	The required display name for the resource.
JNDI Name	* jms/acompany/requestQ	The JNDI name for the resource.
Description		An optional description for the resource.
Category		An optional category string which can be used to classify or group the resource.
Persistence	APPLICATION DEFINED	Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application.
Priority	APPLICATION DEFINED	Whether the message priority for this destination is defined by the application or the Specified priority property.
Specified Priority		If the Priority property is set to Specified, type here the message priority for this queue, in the range 0 through 9.
Expiry	APPLICATION DEFINED	Whether the expiry timeout for this queue is defined by the application or the Specified expiry property, or messages on the queue never expire (have an unlimited expiry timeout).
Specified Expiry	milliseconds	If the Expiry timeout property is set to Specified, type here the number of milliseconds (greater than 0) after which messages on this queue expire.
Base Queue Name	* requestQ	The name of the queue to which messages are sent, on the queue manager specified by the Base queue manager name property.
Base Queue Manager Name	ACompanyQM	The name of the WebSphere MQ queue manager to which messages are sent.
CCSID		The coded character set identifier for use with the WebSphere MQ queue manager.
Native Encoding	Use native encoding	When enabled, native encoding is used. When disabled, the settings for integer, decimal and floating point are used.
Integer Encoding	Normal	If native encoding is not enabled, select whether integer encoding is normal or reversed.
Decimal Encoding	Normal	If native encoding is not enabled, select whether decimal encoding is normal or reversed.
Floating Point Encoding	IEEENormal	If native encoding is not enabled,

Figure 6-25 Configure MQ request queue destination

5. Configure the replyQ the same as the requestQ, except use the following information where applicable:
   - Name: `OrderReplyQueue`
   - JNDI Name: `jms/acompany/replyQ`
   - Base Queue Name: `replyQ`

Configure output port and message listener service

As the output port and message listener service, reference JNDI names from the queue connection factory and queue definitions. The configuration of these is the same as in “Configure the output port” on page 263 and “Configure message listener service” on page 265.

Important: The configuration of the internal JMS server, as specified in “Configure the internal JMS server” on page 266, is not required when using an external JMS provider.

Save the configuration for WebSphere. Log out, then close the Administrative Console.

6.6 Deployment

The sample that was created in 6.3.2, “Creating the sample” on page 235 is a J2EE application that includes the advanced messaging support in Extended Messaging. Deployment of this sample is the same as deploying any J2EE application using the WebSphere Administrative Console. To deploy the sample, do the following:

1. Start WebSphere Application Server V5 if it is not running.
2. Start the Administrative Console.
3. Follow the steps in 6.5, “Configuration” on page 258 to configure EM using either the Embedded Messaging or external WebSphere MQ.

To facilitate testing of the sample application in WebSphere, you can use the sample enterprise application included in the downloadable code and skip to step 7 on page 275. If you want to continue to use the EJB sample you created in WebSphere Studio IE from “Extended Messaging” on page 227, use the sample .WAR file provided with the downloadable code and proceed to step 4.

4. Using WebSphere Studio IE, import the ACompanyEMSWeb.war into the ACompanyEMSWeb project.
   a. Select **File -> Import... -> War file**, click **Next**.

b. Select the ACompanyEMSWeb from your local disk.

c. For Web project, select **existing**.

d. Browse to the **ACompanyEMSWeb** project.

e. Select **Overwrite existing resources**.

f. Click **Finish**.

The import window should look like Figure 6-26.

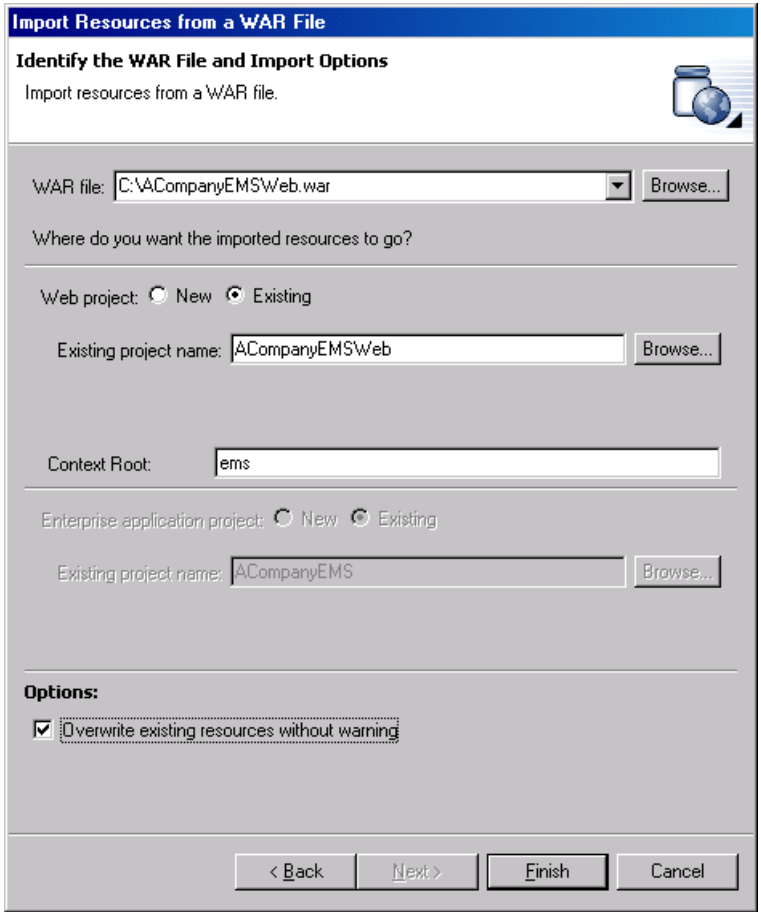


Figure 6-26 Import ACompanyEMSWeb.war

5. Update the Java build path for the ACompanyEMSWeb project, as follows:

   a. In the J2EE Hierarchy view, right-click **ACompanyEMSWeb** and select **Properties**.

b. Select **Java Build Path**.

c. Open the **Projects** tab and select **ACompanyEMSEJB**.

d. Select the **Libraries** tab, then click the **Add Variable** button.

e. In the New Variable Classpath Entry window, select **WAS_EE_V5**, then click the **Extend** button.

f. In the Variable Extension window, select **/lib/cmm.jar**, then click **OK**.

g. In the Properties for ACompanyEMSWeb window, click **OK** to apply the changes. If you see errors in your project, you may need to rebuild the project.

6. Export the ACompanyEMS enterprise application, as follows:

   a. Select **File -> Export**, then select **EAR file**.

   b. For resources, select **ACompanyEMS**.

   c. For the location, select the location of your choice.

   d. Click **Finish**.

7. Open the WebSphere Administrative Console if not already open.

8. Select **Applications -> Install New Application**.

9. At the specify the EAR panel, browse the **ACompanyEMS.ear** file either provided with the download, or exported from WebSphere Studio IE.

10. Select **Next**.

11. Continue selecting **Next**. You will see the JNDI definitions for the listener port, and EJB definitions.

12. Select **Finish** on the last window.

13. At the end, save the configuration for WebSphere.

14. Start the ACompanyEMS enterprise application in WebSphere.

15. Open a browser and go to `http://localhost:9080/ems/OrderInput.html`.

You should see the following window. The form will call a servlet that calls the sendWithReply method of the OrderSender bean. The servlet will return the response to a corresponding JSP.

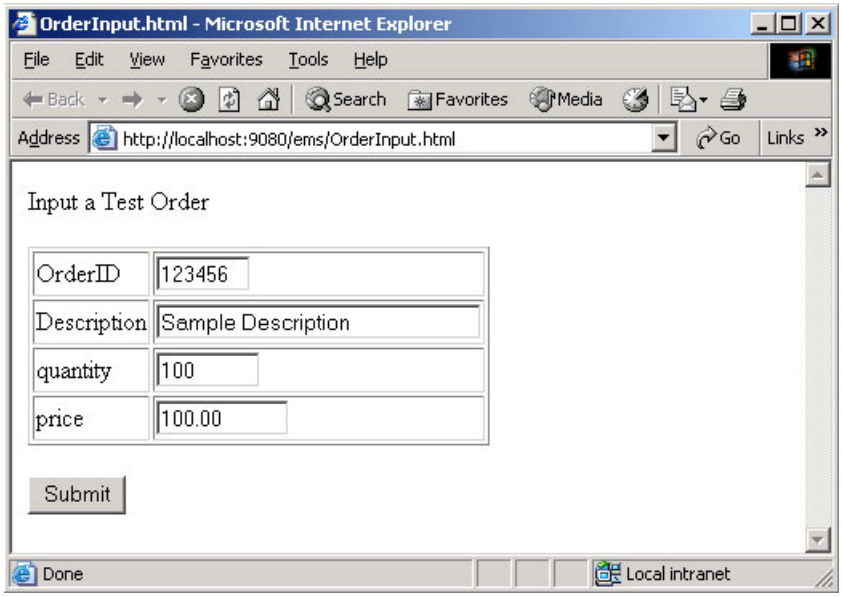


Figure 6-27 Testing the application

16. Click the **Submit** button, and you should see a response that the order was processed successfully.
17. Examine the logs under your server, and you should see the SystemOut.log file in Example 6-5.

Example 6-5 Output log

```
[3/26/03 11:09:01:609 CST] 777615e6 SystemOut     O OrderID: 123456
[3/26/03 11:09:01:609 CST] 777615e6 SystemOut     O Description: Sample
Description
[3/26/03 11:09:01:609 CST] 777615e6 SystemOut     O Quantity: 100
[3/26/03 11:09:01:609 CST] 777615e6 SystemOut     O Price: 100.0
```

If your don't see the confirmation window, look at the SystemOut.log and SystemErr.log in *<WebSphere_root>*/logs/server1. See "Problem determination and troubleshooting" on page 284 for additional details.

6.7 Transactions and workload management

Because messaging systems are inherently asynchronous, propagating transactions on sender and receiver beans can present complications. Table 6-2

on page 277 lists the type of request and the transactional ramifications of that request.

Table 6-2 Transaction and workload management support

Type of Request	Transactional Interaction	Workload Management	Prerequisites /Limitations	Notes
Synchronous Send with no response	Send is transactional. Put is conditional on completion of transaction.	None	None	None
Synchronous Send with Response	Send is *not* transactional.The get of response is transactional.	None	A temporary reply queue should be avoided for failover situations.	None
Send with deferred response	Both send and get of response are transactional.	None	Required to have a specific reply to queue. Temporary response queue is not permitted.	None
MDB style receiver	Both receive and reply are transactional.	Receive is not a problem. Reply to Connection Factory and destinations must be specified on all servers in cluster.	None	Request and response either both work or both fail. In general you should always set the listener retry higher than the queue's default retry limit.

Type of Request	Transactional Interaction	Workload Management	Prerequisites /Limitations	Notes
Application Callable Receiver with Response	Both receive and reply are transactional.	As long as the connection factory and queue destination are defined consistently, a response may be sent from a host other than the one that provided the get.	Non-JMS messages use the Reply-to queue as specified in the input port when a reply is sent.	If two transactions are used, it is possible to consume the request and lose the reply. (Not possible if one TX is used for both get and put.)
Handle Late Response	Response is transactional.	Workload management could be problematic, since the late response handler writes the correlation ID to a flat file on the JMS server node.		

6.8 Handling late responses

Late response handling provides a sender bean with the ability to have response messages redirected to a defined MDB for response messages that cannot be delivered in the requested timeout interval. This section is taken from the help section of WebSphere Studio IE that details late responses.

6.8.1 Late response description

If an application uses a sender bean to send a message, it can optionally retrieve a response to the message. The sender bean can either wait for the response or defer retrieval of the response. Sometimes a response is delayed within the messaging infrastructure, and therefore the application cannot receive the response. Extended Messaging can retrieve such a response message (referred to as a late-response message) when it does arrive and pass it to an MDB provided by the application to handle late responses. The MDB used to handle

the late response is a standard EJB 2.0 MDB or a receiver bean deployed as an MDB. The deployed MDB can then perform its processing on the message.

Late responses should not be considered normal application behavior.

For Extended Messaging to handle late responses for an application, the sender bean must be deployed with the Handle Late Responses option enabled.

Definition of a late response

A late response occurs when the application is no longer able to retrieve responses to messages that it has sent, as follows:

- Send with deferred response:

  The application (enterprise bean) repeatedly tries to retrieve a response until it ends. When the application no longer wants to retry to get a response, it can register a request for Extended Messaging to handle the late response, by calling a registerLateResponse() method on the sender bean. The registerLateResponse() method is created by the WebSphere Studio IE wizards when the generate method for late response check box is enabled. See Figure 6-28 on page 281.

- Send with synchronous response handling:

  When the sender bean sends a message, it waits for the response. The result of this is that either the sender bean retrieves the response message or a timeout error occurs. If the system raises a timeout error, the application can no longer retrieve a response to the message. At this time the Extended Messaging service registers the message for a late response.

Handling responses

Extended Messaging handles responses in the following stages:

1. Requesting a late response when it is available.

   To ask the system to handle late responses for a sender bean, you deploy the sender bean with the Handle late responses extension to the Deployment Descriptor. If selected, the Handle Late Responses option defines that Extended Messaging should pass the response, when it becomes available, to the MDB provided by the application to handle late responses. When the sender bean is deployed, a specialized listener port is associated with the bean. This listener port is known as a handle late response listener port. If the option is not selected, then the system does not handle late responses, and it is the application's responsibility to handle any late responses.

2. Starting a JMS listener to retrieve the message when it is available, which then drives the message bean to handle the JMS message.

The listener port must be defined with the following properties:

a. The same JMS destination is specified as the JMS response destination on the output port used by the sender bean.

b. A listener port extension with Handle Late Responses enabled.

Note: You cannot use a temporary destination for late responses.

3. If a request is made to handle a late response, the Extended Messaging service immediately registers a LateResponse message request with the extended message consumer for the given listener port. The message request is registered independently of any transaction context that the sender bean has. A request record (containing the MessageID of the late response) is added to the AsyncMessageLog log. When the message is eventually received, it is passed to the MDB deployed against the specified late response listener port.

6.8.2 Configuration of late response

The configuration of a late response consists of configuring a late response policy, listener port, and late response extension.

Select late response handling

To enable late response handling, select the **Generate method to register late responses** check box when defining the sender bean. This check box will be enabled when a response type is selected, as shown in Figure 6-31 on page 284.

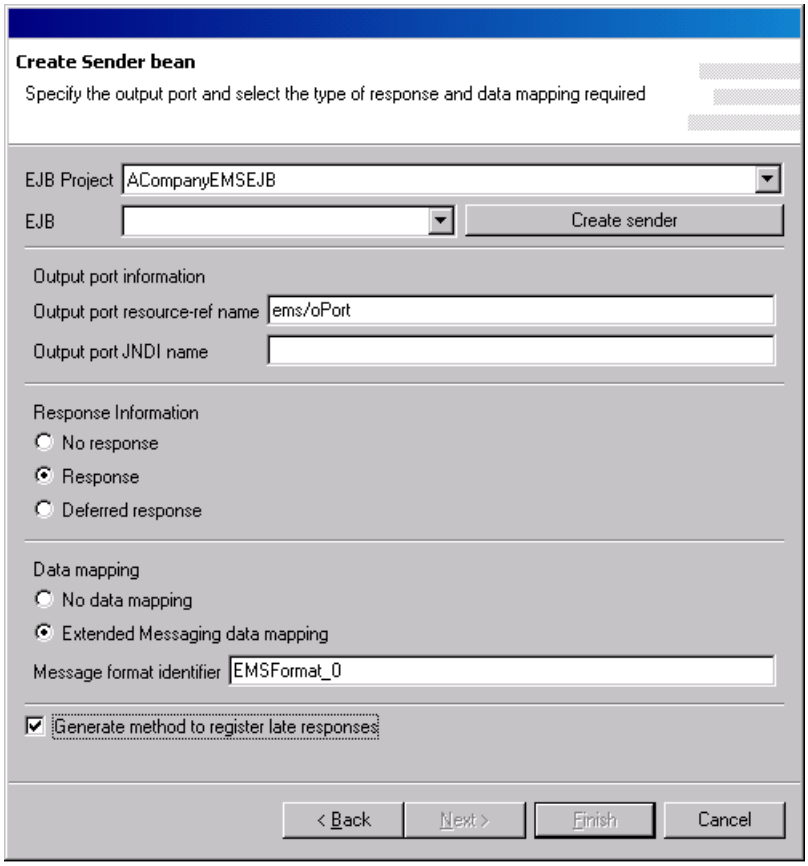


Figure 6-28 Select late response handling

Late response policy

After the sender bean has been created, it is necessary to configure the late response handler policy in the Deployment Descriptor for the EJB module.

1. On the J2EE perspective in the J2EE hierarchy view, select the **EJB module**.
2. Right-click the **EJB module**, then select **Open with -> Deployment Descriptor Editor**.
3. Go to the Extended Messaging tab and specify a late response handler policy for the sender methods, as shown in Figure 6-29 on page 282.

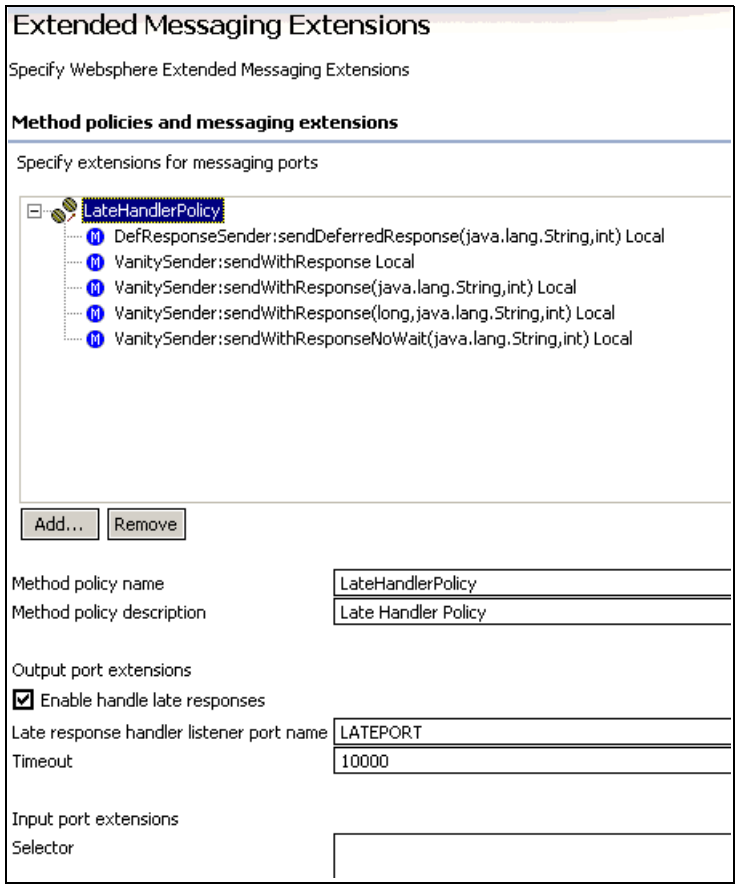


Figure 6-29 Late handler policy

The sample above included both a deferred sender as well as a traditional sender for testing purposes. Generally you would select any of the sender methods that should handle late responses.

Late response listener port

Late response handler MDBs have their own listener ports that listen to the reply queue. This allows the late response MDB to pick up the unhandled late message when it hits the reply queue after the timeout. The listener port will be configured as any traditional listener port, but it will be listening on the reply queue instead of the request queue. In the sample shown in Figure 6-29, a port was specified called “LATEPORT”. The listener port definition would look like Figure 6-30 on page 283.

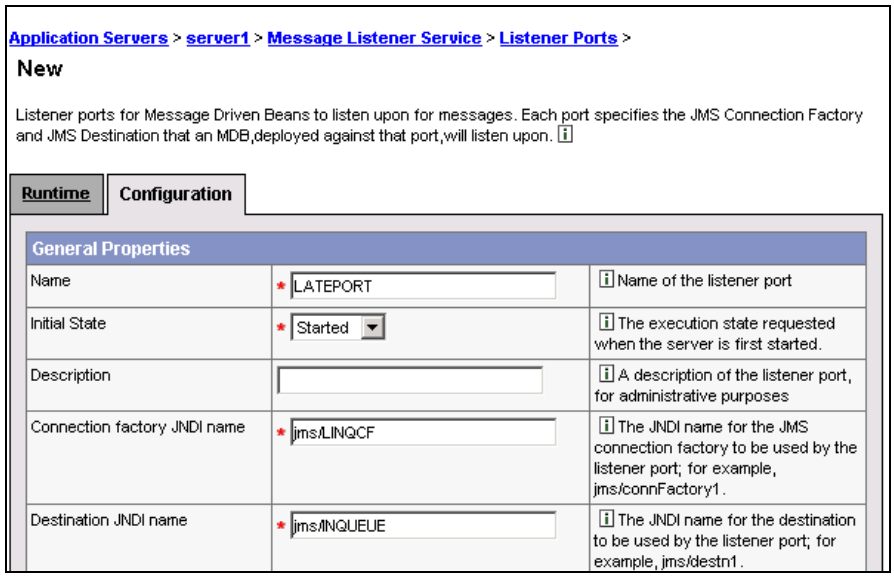


Figure 6-30 Late handler listener port

Late response extension editor

When creating the listener port, it is also necessary to configure the listener port extension, as follows:

1. Go to the listener port you have configured.
2. Select this listener port and scroll down to the Additional Properties at the bottom of the right pane.
3. Select **Listener Port Extensions**.
4. Define the late response handling attributes per Figure 6-31 on page 284.

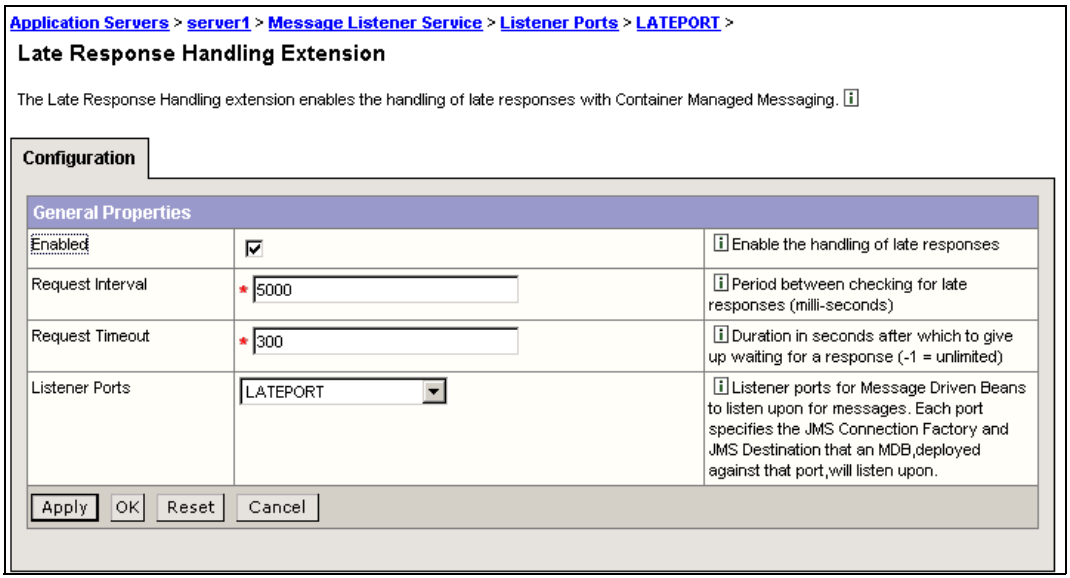


Figure 6-31 Late response extended configuration

You have now completed the necessary steps for configuring late response handling.

6.9 Problem determination and troubleshooting

Troubleshooting is covered in both the help for WebSphere Studio IE and the help for WebSphere Application Server Enterprise V5. A few common mistakes configuring Extended Messaging are:

- NameNotFound exception: This is seen if there is a mismatch between the JNDI name in the application code and the JNDI names specified for the listener, input, or output ports.
- When using Embedded Messaging, if the internal JMS server is not configured with the appropriate request and reply queues, then these queues will not be started and you will see errors trying to connect to the queues.
- MQ2058: This is seen when using WebSphere MQ for the JMS provider, and WebSphere cannot connect to the queue manager. This is usually caused by not starting the configured queue manager in WebSphere MQ.

6.10 Security considerations

Because Extended Messaging is a higher level abstraction than the JMS API, the security configuration is the same for configuring JMS security for WebSphere. When WebSphere security is on, it is necessary to configure credentials for the queue connection factory and queue definitions. See Chapter 7, “Securing Enterprise Integration components”, in *IBM WebSphere V5.0 Security WebSphere Handbook Series*, SG24-6573.

7

Asynchronous Beans

Asynchronous Bean functionality is a new feature in WebSphere Enterprise Version 5. It is also a clear and significant differentiator between WebSphere and all of its competitions. There is no other product that has this ability to offer an asynchronous programming model integrated as part of a J2EE application server. We introduce this technology in this chapter.

7.1 Planning

Have you ever felt the need to spawn threads to do some work asynchronously inside your servlet or EJB code, just as you are able to accomplish in normal Java client code? The answer is most likely yes. However, the J2EE specifications strongly recommend against trying to spawn other threads in the container.

> **Quote:** “This division of responsibilities requires that the application components do not contain functionality that would clash with the functions provided by the J2EE platform. If an application component tried to provide a function that the J2EE platform implements, the J2EE platform could not properly manage the function.
>
> “Thread management is one example of functionality that would clash with the J2EE platform’s function. If enterprise beans were allowed to manage threads, the J2EE platform could not manage the life cycle of the enterprise beans, and it could not properly manage transactions.”
>
> - from the *J2EE Spec 6.2.1: Programming Restriction*

This J2EE constraint has a valid reason: the presence of unmanaged threads in the application server can seriously undermine the ability of the application server to ensure a stable, optimized, and scalable execution environment. Another key issue associated with application code creating threads is that the J2EE context of the application (security, local name space, and so on) does not “flow” to the newly spawned thread. So what the executed code can do is very limited, since the thread doesn’t have access to the full and rich J2EE programming model.

The Asynchronous Bean framework provided by WebSphere Enterprise resolves these two issues by providing J2EE components access to managed threads, and also allowing their J2EE contexts to be propagated to a separate thread. By using Asynchronous Beans, your J2EE components will be able to submit code to be run on a separate thread and asynchronously. The code will execute in a full J2EE execution context derived from your main thread of work, and thus can use the full J2EE APIs. The work will execute on threads taken from the WebSphere thread pool, and thus avoid the overhead of having to create threads on the fly. Most importantly, WebSphere has control of all these threads, allowing for better utilization of application server resources.

7.1.1 What are Asynchronous Beans?

An Asynchronous Bean is a Java object or enterprise bean that can be executed asynchronously by a J2EE application, using the J2EE context of the bean's creator. The ability to flow J2EE context to the newly spawned thread is very important, because it gives to the executed code the rich and full J2EE programming model and API.

The Asynchronous Bean model represents a very interesting compromise between the loosely coupled approach used with messaging, where there is no propagation of context, and the tightly coupled approach of traditional J2EE programming, which requires a single thread of execution.

7.1.2 Asynchronous Beans programming interfaces

Asynchronous Beans provide full support of application controlled threading, asynchronous callbacks, scoped alarms, and subsystem monitors, and yet it is simple to use interfaces to your servlet or EJB code. Figure 7-1 on page 290 shows the hierarchy of the Asynchronous Beans interface.

```
interface com.ibm.websphere.asynchbeans.Alarm
interface com.ibm.websphere.asynchbeans.AlarmListener
interface java.util.EventListener
   interface com.ibm.websphere.asynchbeans.WorkListener
interface com.ibm.websphere.asynchbeans.EventSource
   interface com.ibm.websphere.asynchbeans.AlarmManager
   interface com.ibm.websphere.asynchbeans.AsynchScopeManager
      interface com.ibm.websphere.asynchbeans.AsynchScope
      interface com.ibm.websphere.asynchbeans.WorkManager
   interface com.ibm.websphere.asynchbeans.SubsystemMonitor
interface com.ibm.websphere.asynchbeans.EventSourceEvents
   interface com.ibm.websphere.asynchbeans.AlarmManagerEvents
   interface com.ibm.websphere.asynchbeans.AsynchScopeEvents
   interface com.ibm.websphere.asynchbeans.SubsystemMonitorEvents
   interface com.ibm.websphere.asynchbeans.WorkManagerEvents (also extends
com.ibm.websphere.asynchbeans.WorkEvents)
   interface java.lang.Runnable
interface com.ibm.websphere.asynchbeans.Work
   interface java.io.Serializable
interface com.ibm.websphere.asynchbeans.WorkWithExecutionContext
interface com.ibm.websphere.asynchbeans.SubsystemMonitorManager
interface com.ibm.websphere.asynchbeans.WorkEvent
interface com.ibm.websphere.asynchbeans.WorkEvents
   interface com.ibm.websphere.asynchbeans.WorkManagerEvents (also extends
com.ibm.websphere.asynchbeans.EventSourceEvents)
interface com.ibm.websphere.asynchbeans.WorkItem
```

Figure 7-1 AsynchBean API hierarchy tree

We can group these interfaces in the following manner:

- EventSource and EnventSourceEvents related
- WorkManager, Work, WorkEvent, WorkItem and WorkListener related
- AlarmManager, Alarm and AlarmListener related
- AsynchScope related
- SubsystemMonitor related

The key inheritance relationships in the Asynchronous Bean framework are illustrated in Figure 7-2 on page 291.

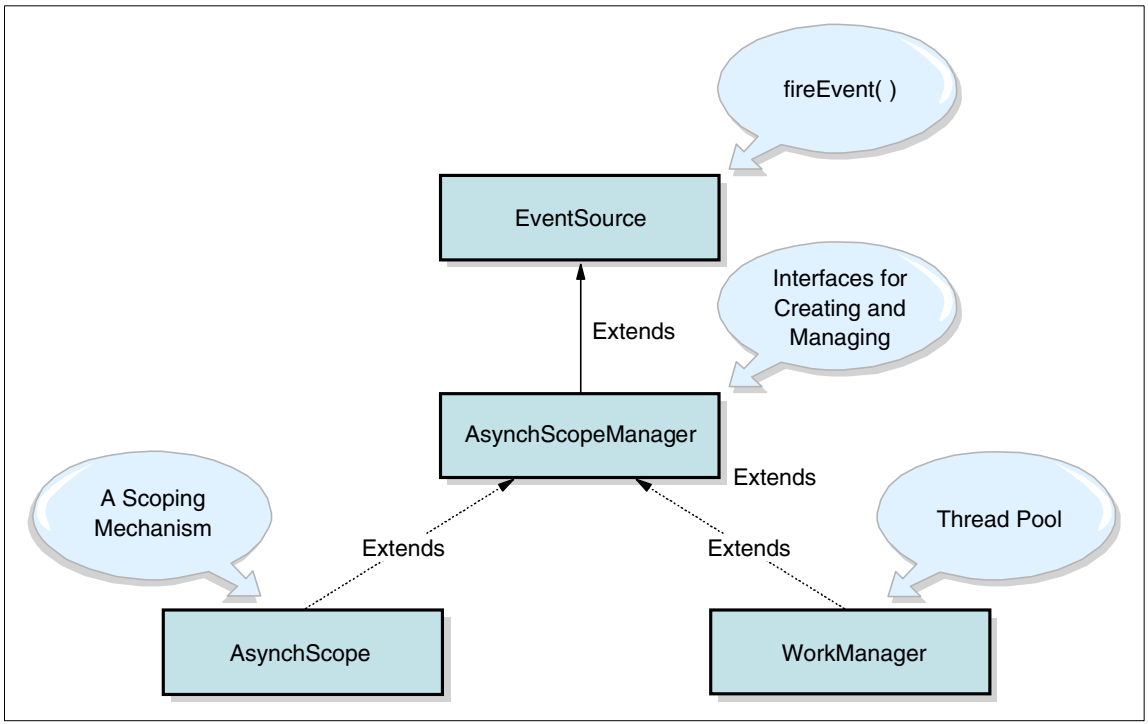


Figure 7-2 Key interfaces and relationships

Important: Note the following two points:

1. AsynchScope and WorkManager extend AsynchScopeManager, so they can be used to create AsynchScope, which allows you to build the hierarchy.
2. AlarmManager, AsynchScope, WorkManager and SubsystemMonitor are also EventSources, which means all could fire asynchronous events.

WorkManager and Work

At the heart of the APIs is the WorkManager interface, because it is the “anchor point” that allows us to create all the various Asynchronous Beans. The WebSphere administrator should create WorkManagers for J2EE applications that require them. The administrator specifies the properties of the thread pool and the “sticky” context policy for any Asynchronous Beans using this WorkManager. There can be many of these WorkManagers. The administrator binds each one in a unique place in JNDI.

Applications will look up the WorkManager using resource references (resource-ref). Here the component has a resource-ref called “wm/myWorkManager”, which will be bound to physical WorkManager JNDI name at deployment time.

Example 7-1 Resource reference for a WorkManager

```
<resource-ref id="ResourceRef_1">
   <description>My WorkManager</description>
   <res-ref-name>wm/myWorkManager</res-ref-name>
   <res-type>com.ibm.websphere.asynchbeans.WorkManager</res-type>
   <res-auth>Container</res-auth>
   <res-sharing-scope>Shareable</res-sharing-scope>
...
```

The res-ref-name is, as usual, the relative name used in the component for its java:comp/env to find the resource, in this case a WorkManager. The res-auth and res-sharing-scope are specified but ignored for a WorkManager. You can add this to a Web application’s web.xml or an EJB in the ejb-jar.xml files. You can also use Application Assembly Tool or WebSphere Studio IE to specify them in the editor for these files by simply adding a reference and typing the type as `com.ibm.websphere.asynchbeans.WorkManager`. Example 7-2 shows the code a client uses to look up a WorkManager.

Example 7-2 Look up a WorkManager

```
InitialContext ic = new InitialContext();
WorkManager wm = (WorkManager)ic.lookup("java:comp/env/wm/myWorkManager");
// now we can use it.
```

A WorkManager in its simplest form is a thread pool. Its special nature allows us to use it to run Work instances asynchronously and to transfer J2EE context to those threads. It supports the following operations:

- **startWork()** is used to initiate asynchronous work. This operation is heavily loaded to enable a variety of different ways of controlling the asynchronous work, including with the current execution state or previous execution state, with or without a WorkListener, and with or without setting a timeout for when the asynchronous work must be started. An instance of WorkItem will be created and returned from this operation, to represent the specific thread of asynchronous work created with this operation.
- **doWork()** is used to initiate synchronous work. This operation is similarly loaded to enable a variety of different ways of controlling the work. The primary use for this operation is to execute a piece of work in an execution context that was captured earlier.

- **create()** can be used to create a WorkWithExecutionContext, capturing the current execution context from the calling thread of execution and combining that with work that you want associated with that context. The captured WorkWithExecutionContext can be used later with either the startWork() or doWork() operation to initiate that work.
- **join()** can be used to block until either any or all of the outstanding asynchronous work has completed. This can be used to idle the main thread while parallel work is completing.

EventSources and Event

Asynchronous Beans also provide a generic event notification framework, which is essentially an implementation of the observer pattern. An application can create generic listeners and subscribe those listeners to monitor certain events produced by an EventSource.

Several special classes in WebSphere Enterprise are already an EventSource, such as AlarmManager, AsynchScope, WorkManager and SubSystemMonitor. These event sources can fire their specific events, such as AlarmManagerEvent, AsynchScopeEvent, WorkManagerEvent and SubSystemMonitorEvent respectively. Applications interested in monitoring those events can implement the event listeners, and register with the event sources.

There is also a special event source that you can use for intra-application notification. This event source is included in each Enterprise Application. You can look it up by using JNDI reference java:comp/websphere/ApplicationNotificationService. This intra-application event notification service makes it possible for components that belong to the same application to communicate with each other through notification. For example an EJB could subscribe a listener, while another EJB could fire an event.

Restriction: Keep in mind the intra-application notification is not valid for communications that involves multiple EARs, where instead JMS should be utilized.

The EventSource provides “type-safe” notification. You can ask an EventSource to fire an event and target only the listeners that implement a certain interface.

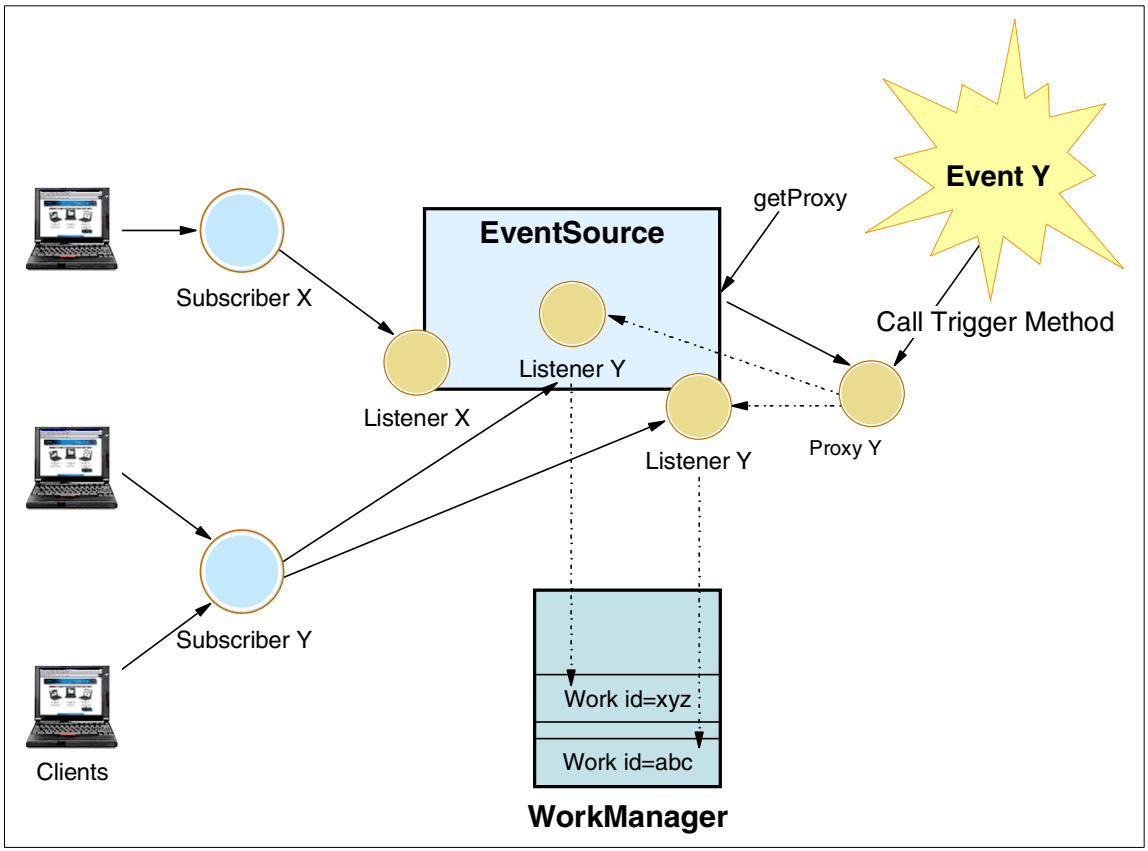


Figure 7-3 EventSource and Event

AsynchScope

An AsynchScope is a scoping mechanism. It owns an AlarmManager and a SubSystemMonitorManager. If the scope is destroyed, then any alarms/subsystemmonitors managed by its alarm manager and subsystem monitor manager are also destroyed. Properties can be stored in an AsynchScope, giving J2EE applications a way to store a non-serializable state that otherwise could not be stored in a session bean (Alarms, WorkItems, application data).

AsynchScope can also have children that are also AsynchScopes. These can be useful for scoping data underneath the parent. If the parent is destroyed, then the children are destroyed also. AsynchScopes are named, and all scopes at the same level of the tree must be uniquely named. Ultimately, a WorkManager owns all AsyncScopes. This hierarchical structure is useful when you need to monitor complex subsystems that may have a hierarchical structure themselves.

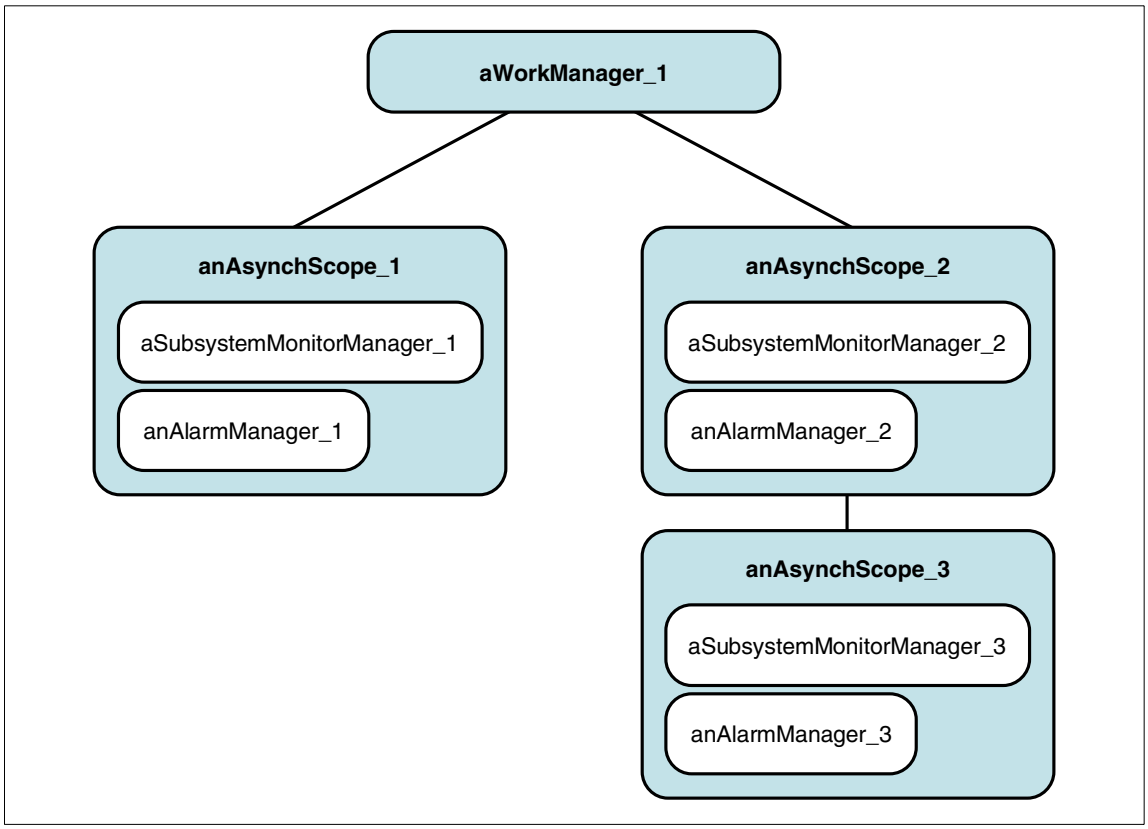


Figure 7-4 AsynchScope hierarchy

Note: The WorkManager is the root of all of the Asynchronous Beans framework. It can used by itself, or it can be used to create AsynchScope. The AsynchScopes will utilize thread pool provided by the WorkManager.

AlarmManager and Alarm

A special application of the EventSource and Listener pattern is provided by the Alarms. An application can get hold of an AlarmManager configured within every AsynchScope. Example 7-3 shows a code snippet to get an AlarmManager, where `wm` is an instance of WorkManager you looked up through JNDI.

Example 7-3 Lookup AlarmManager

```
AsynchScope as = wm.findAsynchScope("ItsoScope");
if (as == null) {
   as = wm.createAsynchScope("ItsoScope");
```

```
}
//get the AlarmManager
AlarmManager am = as.getAlarmManager();
```

AlarmManager provides the create() method to create a new alarm. This method has three parameters:

- An AlarmListener, which is the target for the alarm and the fired method on this is called when the alarm goes off.
- A Context object for the alarm, which is useful for supplying alarm-specific data to the listener and allows one listener to be created and used for multiple alarms.
- An integer (int) specifies the milliseconds in which the alarm fires.

Alarm is a device that sits on its own thread in the WorkManager and will go off after the specified number of milliseconds. When the alarm goes off, it will call the fired() method on the listener, using the J2EE context of the alarm creator. The creator can interact with the alarm manager, reset, or cancel the alarm.

Alarms are high-performing and transient. The application that is using them need to recreate them after server shutdown and restart.

Restriction: Alarms are not persistent in WebSphere Enterprise. If you need to provide a persistent definition of time-activated tasks, the Scheduler Service should be used instead.

SubsystemMonitor

A subsystem monitor is returned to allow an application to interact with the monitor created on its behalf. This is an EventSource so an application can register a listener to it. An application can get hold of a SubsystemMonitor through a SubsystemMonitorManager, which is configured within every AsynchScope. Example 7-4 is the code snippet, where `wm` is an instance of WorkManager you looked up through JNDI.

Example 7-4 Get a SubsystemMonitor

```
AsynchScope as = wm.findAsynchScope("ItsoScope");
if (as == null) {
       as = wm.createAsynchScope("ItsoScope");
}
//get the AlarmManager
SubsystemMonitorManager ssmm= as.getSubsystemMonitorManager();
SubsystemMonitor ssm = ssmm.create(“name”, heartBeatInterval,
   missedBeatsForStale, missedBeatsForDead);
```

A subsystem monitor is basically a set of alarms. When it is created, the application tells it how often heartbeats should be expected, how many beats missed means that it is stale, and how many beats missed means it is dead. The monitor then sets up alarms to track this status. If the ping method is called, then these alarms are reset. If the alarm fires, then this means that ping has not been called and no heartbeat was received by the application for the subsystem this monitor is “watching”.

When the number of beats for stale has elapsed without a ping, then we fire a stale event. Later, if the number of beats for dead elapse without a ping, then we fire a dead event. If a ping is received after a stale or dead notification, then we send a fresh event indicating the subsystem is alive again and carry on as normal.

If the stale beats equal the dead beats, then no stale event is published, and we just publish a dead event. The number of dead beats should always be greater or equal to the number of stale beats. What constitutes a ping is application specific. A destroy notification is also published.

The subsystem monitor can generate events. The events that can be generated are documented in the SubsystemMonitorEvents interface. Implement the Events interface and add an instance of this object using the SubsystemMonitor.addListener method.

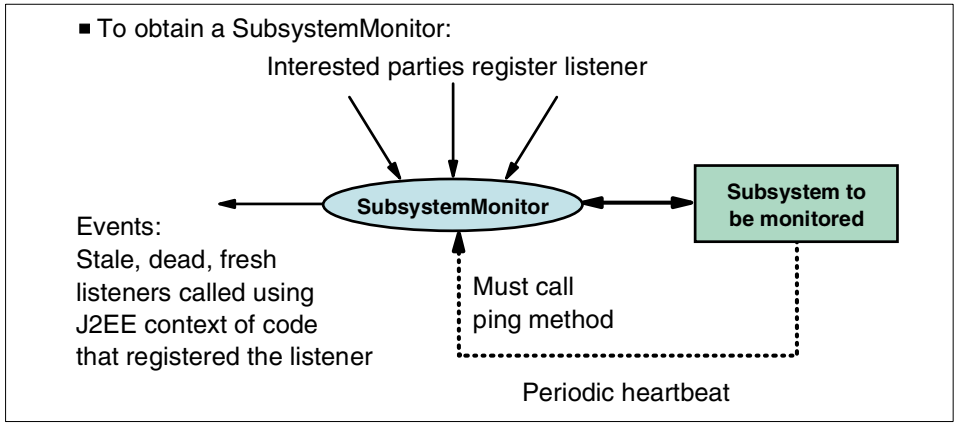


Figure 7-5 SubsystemMonitor

The flavors of Asynchronous Beans

There are three different flavors of Asynchronous Beans, which are explained in the following sections.

Work

This is an object that implements the com.ibm.websphere.asynchbeans.Work interface, and represents the work that you want to execute asynchronously. The Work class is derived from java.lang.Runnable. You must implement the following method for any Work:

- **run():** The work manager will call the run() method on your Work object when it spawns the thread, having already set the execution context for this thread. The run() method should be implemented as though it were a J2EE client, and should include any logic you want to execute in the asynchronous thread.
- **release()**: The work manager will call this method when it wants to terminate the thread of your Work. You should implement this method to terminate any work that you are performing in the run() method, the run() method should return because of the release() method having been called.

The Work is also registered with its own EventSource once it is started. The WorkItem.getEventTrigger can be used to return a proxy that can be used to fire events to the running Work. A Work object should implement any necessary event interfaces if this mechanism is to be exploited. The WorkItem.getEventTrigger method can then be used to fire events to those interfaces. If the Work does not implement the interface supplied to WorkItem.getEventTrigger, then nothing happens and the event is ignored.

AlarmListener

This is an object that implements the com.ibm.websphere.asynchbeans.AlarmListener interface. This is called when a high-speed transient alarm expires. Alarm listeners are special listeners that will be “fired” when an alarm goes off. Any listener must implement the following method:

- **fired()**: This method is called when an alarm fires. You implement any business logic that should occur inside this method. Also you need to know if the alarm is a one-time event or a repeating one. If the alarm is a one-time event or is the last alarm in a periodic alarm, then the alarm should be cancelled. Cancelling the alarm allows it to be reused, thus saving memory and increasing performance. If the alarm is a repeating alarm, then the Alarm.reset() method should be called to schedule the alarm to fire again later. This again improves performance through memory optimizations.

In the Asynchronous Beans framework, an alarm is basically a timer that can be created by a J2EE application and that can be associated with a listener to invoke in case the alarm times out. Once again, the listener will be invoked on a separate thread other than the alarm creator’s thread, but it will inherit the same J2EE context as the alarm creator.

EventListener

Event listener are asynchronous by nature. An application can create an event listener and subscribe it to monitor the occurrence of a certain event. When the event occurs, the listener will be notified and will run a certain logic to handle the event. This is a lightweight asynchronous notification mechanism for asynchronous events within a single JVM. Its main use is envisioned to be J2EE components within a single EAR signaling each other about various application asynchronous events.

Listeners can implement any interface; there are no restrictions. However, the application that originates the events needs to know which method corresponds to the event on the listener's interface and call it. The event originator will do so by acquiring a "proxy" from the EventSource. Calling a method on the proxy will cause the same method to be invoked on all the listeners that are registered, and that implement the requested interface.

The listener's method will be executed in its own thread, but it will run under the J2EE context of the component that registered the listener itself. It will not use the J2EE context of the application that is firing the event.

7.1.3 Asynchronous Beans: simple Java objects or EJBs?

An Asynchronous Bean can be either a Java object or an EJB. Applications that are comfortable with the servlet-only approach may find the Java object approach to be most desirable. Applications more comfortable with EJBs may elect to use Asynchronous Beans implemented with stateless session beans or entity beans.

There are several differences in behavior between the two choices. Table 7-1 summarizes them.

Table 7-1 Java or EJB Asynchronous Beans comparison

	Java Beans	**EJB**
Transactions	If created by a servlet then java:comp/UserTransaction is available. If created by an EJB then only TX_NOT_SUPPORTED is allowed and a "buddy" EJB must be used for full global transaction support.	The support is what is specified by the descriptor for the EJB and the J2EE specification.

	Java Beans	EJB
Security	The credentials on the thread that created the Asynchronous Bean are used when the bean is invoked.	The credentials on the thread that created the Asynchronous Bean are used. However, the descriptor for the bean can override this with the Run-as role attribute.
Application Profile	The profiles active for the creating component are used.	The profiles active for the creating component are used but they may be augmented by specifying additional ones on the target EJB method.
Java:comp scope	The Java:comp of the component that created the Asynchronous Bean are always available to the Asynchronous Bean.	The java:comp of the creating component is ignored. The java:comp of the async EJB is always used.

There is not much difference from a performance point of view. The performance is roughly equivalent to a local method call in both cases.

An EJB-based Asynchronous Bean is basically more independent of the creating component.

7.1.4 Asynchronous Beans: programming model

Asynchronous beans represents a very interesting compromise between the loosely coupled approach used with messaging, where there is no propagation of context, and the tightly coupled approach of traditional J2EE programming, which requires a single thread of execution.

Access to J2EE component metadata

The J2EE component metadata of the creating component is available to the Asynchronous Bean when it is a simple Java object. Obviously, if the Asynchronous Bean is a J2EE component such as a session bean then this is the metadata that is active when a method is called.

However, when the object is a simple Java object, then it is allowed to look up the java:comp name space as its creator would. This allows it to look up connection factories and EJBs in the normal J2EE way. The environment properties of the creating component are also available. The java:comp name space is identical to the one available to the creating component. All connection factories use the same resource sharing scope as the creating component also. The only

exception to this rule is the java:comp/UserTransaction, which is only available to the Asynchronous Bean when the J2EE component that created it was a servlet. It is not visible when the owner was an EJB even if it was using bean-managed transactions.

Other “sticky” J2EE contexts

We want Asynchronous Beans to inherit some of the J2EE contexts from the creating component. The following J2EE contexts can be made sticky:

- Internationalization context
- WorkArea context
- Application profile
- Security context

Table 7-2 lists the methods that capture the J2EE context and thus create an Asynchronous Bean.

Table 7-2 When J2EE context lifetime

Method	Description
Work.startWork	Start an Asynchronous Bean on another thread
AlarmManager.create	Run the Asynchronous Bean when the alarm expires
EventSource.addListener	Run the Asynchronous Bean when a matching event is published on the EventSource

All of these methods remember the J2EE context when they are called by an application. It is this J2EE context that is used when the Asynchronous Bean is invoked asynchronously later. The J2EE context when the Asynchronous Bean was created by the application is not important. It is only when the object is passed to one of the above methods that the J2EE context is remembered.

Transactions

Due to the asynchronous nature of this programming model, transactional contexts are not passed to the Asynchronous Beans. Every Asynchronous Bean method is called using its own local transaction. This is very similar to container-managed transactions in a normal EJB when the TX_NOT_SUPPORTED attribute is specified. The runtime starts a local transaction containment before invoking the method. If the Asynchronous Bean needs to make a global transaction, then there are several choices:

- If the Asynchronous Bean was created by a servlet then it can use the UserTransaction object in JNDI at java:comp/UserTransaction to do this or it can use a helper EJB method.
- If the Asynchronous Bean was created by an EJB, then it cannot use java:comp/UserTransaction, even if the owner EJB was bean managed. It must also use a helper session or entity bean in this case. The work to be performed inside the transaction is contained in the helper EJB method.
- If the Asynchronous Bean is actually an EJB, then the normal J2EE transaction rules apply. Here we have a stateless session bean whose local interface implements an Asynchronous Bean interface such as Work, EventSourceEvents or AlarmListener. When this is invoked, then the normal J2EE rules apply. The transaction (TX) settings that apply to the method or EJB apply. This is the most flexible way to deal with transactions with Asynchronous Beans.

Security

If security is enabled, you have two options when configuring the WorkManager:

1. The WorkManager can be configured to remember the credential of the thread that created the Asynchronous Bean. This is very useful, since the Asynchronous Bean is allowed to access any resource that the creator was allowed to. If we did not allow this, then when security is enabled the Asynchronous Bean would be severely limited in terms of what resources it can access either locally or remotely due to security restrictions.
2. Configure the WorkManager to never propagate the security credential. Here, when the Asynchronous Bean is executed it runs as an unauthenticated or anonymous credential which doesn't time out. However, since it is unauthenticated you won't be able to call EJBs that have roles attached. But, it is permissible for the Asynchronous Bean to call EJBs with interfaces whose methods are set to RUN_AS_CALLER.

7.1.5 When to use Asynchronous Beans

Asynchronous Beans are a powerful innovative concept and extension to the current J2EE specification. They enable the construction of stateful, “active” and event-driven J2EE applications. These applications address a segment of application space that J2EEs have not previously addressed: advanced applications that require application threading, active agents within a server application, and distributed monitoring capabilities. The following sections describe possible scenarios where different Asynchronous Beans could be utilized.

Partition tasks so that they can run in parallel
You could execute a complex database task using multiple threads; an example would be a complex calculation over a large set of rows. The set can be partitioned and then each partition can execute in parallel. The EJB can block until all threads finish and then aggregate the results and then return.

Integrate non-JMS messaging middleware
Applications can integrate a third-party messaging solution that does not support JMS but has a Java API.

Dynamically listen for JMS queues and topics
If an application needs to subscribe on demand to queues or topics that were unknown when the application was deployed, this is now possible because threads can be started to block and receive those messages.

The use of background processing for performance
An application that needs to perform persistent message logging could use a background task to write batched insert operations to a database in a single transaction. The background task can wake every 100 ms to perform this. Foreground tasks will store the log message in a synchronized data structure and then continue without waiting for the message to be persisted. This is more efficient because previously, each log operation resulted in a “begin, insert msg, commit” operation and also blocked the application during this process, which slowed the application significantly when logging was enabled. Now, the background thread sends the following: “begin, insert msg1,msg2,msg3,msg4, commit”. Batching the insert operations together like this significantly lowers the impact of this on both the application server and the database.

7.2 Design

In this section, we will design an Asynchronous J2EE application. We will start with a base and standard J2EE application, and look into different ways we can enhance it by incorporating asynchronous behaviors in it. We are trying to make this application simple enough to follow, and yet as logical as possible to cover all three different kinds of Asynchronous Beans discussed previously. After studying this sample, you should have a better understanding of the differences among the three types of Asynchronous Beans, how to identify J2EE applications that Asynchronous Beans address and apply to, and most importantly, how to create advanced J2EE applications that require active agents, application threading, and asynchronous notification capabilities.

This sample application is not complicated. For a more comprehensive and real-world Asynchronous Beans scenario, refer to the WebSphereTrader

application shipped with the WebSphere Enterprise V5 product. This application also use the Startup Bean service of WebSphere Enterprise V5. For more information about Startup Beans, refer to Chapter 12, “Startup Bean” on page 489.

7.2.1 Base application overview

Our base application is a standard J2EE application with one CMP entity bean (Department), one singleton Java object (EntityBeanCacheSingleton), one startup session bean (AppBootstrapBean), and one JSP (ListDepartment.jsp). It uses some standard Java/J2EE patterns:

- CMP cache: Some J2EE applications use entity beans. It is a common design pattern for performance reasons to cache the CMP data, especially if the underlying table data does not change frequently.
- Singleton class: There will be only one instance of the class in the JVM. Singleton patterns can be used in many different ways. One use is to hold cache data.
- Startup Bean/class: Often used at server startup to do some preprocessing. In our case, it is used to initialize the singleton class, thus the cache for CMP.

Figure 7-6 is a simple interaction diagram of these components.

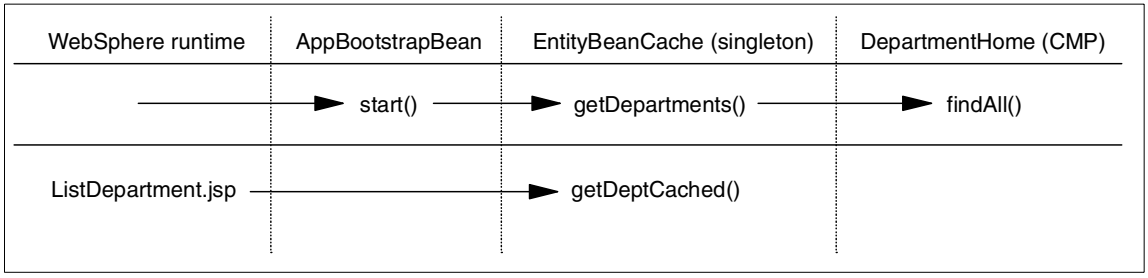


Figure 7-6 Interaction diagram for the sample

Our application get initialized when the WebSphere server starts. WebSphere runtime invokes the start() method of our AppBootstrapBean, which in turn creates an instance of our singleton class EntityBeanCacheSingleton. The singleton class will call the findAll() method in the home interface of the entity bean Department, and will store the result in the cache. So, when WebSphere server is up and running, our application is also initialized with CMP cache in the singleton. Whenever we invoke ListDepartment.jsp, it will get data from the cache held by the singleton class.

7.2.2 Asynchronous patterns

So far, we have a working J2EE application with some good design patterns. But there are two potential problems with the design and this application:

- We are initializing our application cache on the server main thread. If, for some reasons, our Startup Bean takes a long time to finish, this will affect greatly our server runtime, which may appear hanging. This could happen in lots of scenarios: either there are problems with a database connection, or just because of the large size of the cache.
- After we have loaded our application cache through a CMP bean, it will stay in our singleton cache class for the lifetime of the JVM. Even if the underlying data changes, our cache does not know about this. So, the application will get stale data. We need a way to refresh our cache if any data has changed.

These are the scenarios our Asynchronous Beans come to rescue. Here are three asynchronous patterns we will be using in our sample application:

- Background processing using Work

  In this sample application, we want to offload the task of looking up CMP entity beans from the AppBootstrapBean on the main thread of execution. We will actually create a Work object InitializeCacheWork. This Work object will invoke the EntityBeanCacheSingleton, which will in turn do the heavy-duty work of finding all instances of the entity beans, and store the cache in memory. The AppBootstrapBean just needs to initiate the work, then return the control of execution to the main thread of server runtime, without blocking and waiting for the cache to be actually initialized.
- Intra-application notification using EventListener

  We will also implement asynchronous notification to refresh our CMP cache in case the underlying data changes. It works in the following way:

  - First, we define an UpdateCacheEventListener interface with an method/event updateEntityCache.
  - Our EntityBeanCacheSingleton object is interested in this event and wants to be notified when such event fires, so it implements the EventListener interface UpdateCacheEventListener.
  - Whenever the data in the database changes, updateEntityCache event will be fired, and our Singleton object will be called to update its copy of the cache of CMP data.
- Transient, time-based activity using AlarmListener

  In order to simulate the data change behavior in our sample application, we will be implementing an AlarmListener UpdateDbAlarmListener, which will be called every 30 seconds when an associated alarm fires. The AlarmListener

will use the Department CMP to insert a record into the database, and then fire the updateEntityCache event.

7.3 Development - base application

This section provides step-by-step details on how to develop and test the sample application using Asynchronous Beans.

7.3.1 Set up the base application

The sample application will be created in WebSphere Studio Application Developer Integration Edition V5.

Import the base enterprise application project

1. Launch WebSphere Studio IE with a workspace for the Asynchronous Beans sample.

   ```
   C:\WSADIE5\wsappdevie.exe -data C:\SG246932\ItsoAsynchBeansBase
   ```

2. Switch to or open the J2EE perspective.
3. Select **File -> Import** from the menu.
4. Select **Existing Project into Workspace** on the Import window, then click **Next**.
5. Browse for the ACompany folder under the directory where you have the project extracted, in our example: C:\SG246932\ItsoAsynchBeansBase\ItsoAsynchBeans. Click **Finish**.

   The directory gets imported to the workspace. You will see a list of warnings and errors in the Task view, but do not worry about them at this moment.
6. Import the following directories following the steps 3 to 5:
   - ItsoAsynchBeans
   - ItsoAsynchBeansWeb
   - ItsoAsynchBeansEJB
7. Select **Project -> Rebuild All** from the menu to rebuild the application.

Note: Note that the Web module (ItsoAsynchBeansWeb) is using the EJB module (ItsoAsynchBeansEJB).

The EJB module includes the following libraries in the Java Build Path:

- WAS_EE_V5/lib/asynchbeans.jar
- WAS_EE_V5/lib/asynchbeansImpl.jar
- WAS_EE_V5/lib/startupbean.jar

8. Create the database mapping for the EJBs. On the J2EE perspective, J2EE Hierarchy view, right-click **EJB Modules -> ItsoAsynchBeansEJB** and select **Generate -> EJB to RDB Mapping**.
   a. Select **Create a new backend folder** and click **Next**.
   b. Select **Top Down** and click **Next**.
   c. For the target database, select your database (in our case, DB2 Universal Database V8.1). The database name is SAMPLE. The schema name is the user name that created the sample database (in our case ADMINISTRATOR). Click **Finish**.
   d. Close the generated Map.mapmxi file.
9. Generate the deployed code for the EJBs. Right-click **EJB Modules -> ItsoAsynchBeansEJB** and select **Generate -> Deploy and RMIC Code**.
   a. Click **Select All**.
   b. Click **Finish**.

7.3.2 Understand the base application

We have discussed the architecture and structure of our base application in the design section. In the next sections, we take a quick look at our base application code.

The Department CMP Bean

This is a standard EJB 2.0 entity bean with container-managed persistence. In this sample, we created only local home and remote interfaces for it, since it is not accessed remotely by other clients from outside the container. We also created a custom finder findAll() in its home interface using EJBQL language.

The AppBootstrapBean Startup Bean

We use one Startup Bean in this sample to initialize the singleton class and thus the CMP cache. The code in Example 7-5 on page 308 is from the start method of the bean.

Example 7-5 Startup Bean, start() method

```
public boolean start() {
   try {
      System.out.println(
         "**********APPBOOTSTAPBEAN HAS BEEN CALLED**********");

      EntityBeanCacheSingleton s = EntityBeanCacheSingleton.instance();
   } catch (Exception e) {
      System.out.println(
         "AppBootstrapBean.start(): Exception starting app: "+ e.toString());
      e.printStackTrace();
      return false;
   }
   return true;
}
```

The EntityBeanCacheSingleton object

When the instance of this class is created, it will call the findAll() method in the home interface of the entity bean Department, and store the result in a static data structure.

Example 7-6 EntityBeanCacheSingleton.java

```
package com.ibm.itso.was5e.asynchbean;

import java.util.Collection;
import java.util.Iterator;
import java.util.Vector;
import javax.naming.InitialContext;

public class EntityBeanCacheSingleton {
   static private EntityBeanCacheSingleton _instance = null;
   private Vector deptCache = null;

   protected EntityBeanCacheSingleton() {
      System.out.println("**********INITIALIZE THE CMP CACHE**********");
      deptCache = this.getDepartments();
   }

   static public EntityBeanCacheSingleton instance() {
      if (null == _instance) {
         _instance = new EntityBeanCacheSingleton();
      }
      return _instance;
   }

   private Vector getDepartments() {
```

```
        Vector v = new Vector();
        try {
           InitialContext ic = new InitialContext();
           DepartmentLocalHome cbHome =
              (DepartmentLocalHome) ic.lookup("java:comp/env/ejb/Department");
           Collection c = cbHome.findAll();
           Iterator i = c.iterator();
           while (i.hasNext()) {
              DepartmentLocal dl = (DepartmentLocal) i.next();
              DepartmentData dd = new DepartmentData();
              String dno = ((DepartmentKey) dl.getPrimaryKey()).getDeptno();
              dd.setDeptNo(dno);
              dd.setDeptName(dl.getDeptname());
              dd.setMgrNo(dl.getMgrno());
              v.add(dd);
           }
        } catch (Exception e) {
           System.out.println(
              "AppBootstrapBean.start(): Exception starting app: "
                 + e.toString());
           e.printStackTrace();
        }
        return v;
     }

     public Vector getDeptCache() {
        return deptCache;
     }

     public void setDeptCache(Vector deptCache) {
        this.deptCache = deptCache;
     }
}
```

The ListDepartment JSP

As we discussed in the design section, the ListDepartment.jsp will only call the EntityBeanCacheSingleton, and display data from the cache.

Example 7-7 ListDepartment.jsp

```
<%
 out.println("<b>DEPARTMENT LIST</b>: " + new java.util.Date() );
 out.println("<table border='1' cellspacing='1' cellpadding='7'>");
 out.println("<tr><td width='25%'><b>Department No</b></td>");
 out.println("<td width='50%'><b>Department Name</b></td>");
 out.println("<td width='25%'><b>Manager No</b></td>");

 java.util.Vector v = EntityBeanCacheSingleton.instance().getDeptCache();
```

```
java.util.Iterator i = v.iterator();
while (i.hasNext()) {
   DepartmentData d = (DepartmentData) i.next();
  String deptNo = d.getDeptNo();
  String name = d.getDeptName();
  String mgrNo = d.getMgrNo();

   out.println("<tr><td width='25%'>" + deptNo + "</td>");
   out.println("<td width='50%'>" + name + "</td>");
   out.println("<td width='25%'>" + mgrNo + "</td>");
 }
out.println("</table>");
%>
```

7.3.3 Configure the base application

In the following sections are the step-by-step instructions to configure the base application.

Configure a database for the application

We assume that you have installed DB2 for Windows and created the SAMPLE database, because the application will be using the Department table in this database.

1. If you have not done so, install DB2 first.
2. Make sure you are logged in with a user that has permissions to create a database.
3. Start the First Steps application to create the sample database.
4. Select the **Create Sample Database** option and create the sample database.
5. Once the creation process is done, close the First Steps application.

Note: The user who created the sample database sets the schema name for the tables. For example you have created the sample database under the db2admin user, and the schema name is DB2ADMIN.

Configure a test server in the Studio

1. Follow the steps from “Create the test server” on page 670 to create the test server.
2. In the Server Configuration view, expand **Servers**, right-click the new test server, then add the ItsoAsynchBeans project to the server configuration.

Define the data source

1. Open the server configuration for the server.
2. Select the **Security** tab, then add a new JAAS authentication entry with the following details:
   - Alias: `SampleDBAlias`
   - User ID: it is the name of user who created the sample database, in our case `db2admin`
   - Password: password for the user ID, in our case `passw0rd`
3. Select the **Data source** tab, select the **DB2 JDBC Provider (XA)**, then create a new data source with the following details:
   - Version: `Version V5.0 data source`
   - Name: `SampleDS`
   - JNDI name: `jdbc/Sample`
   - Component-managed authentication alias: `SampleDBAlias`
   - Container-managed authentication alias: `SampleDBAlias`
   - Resource property - databaseName: Sample
4. Save and close the server configuration.

7.3.4 Run the base application

1. Start the test server **ACompanyServer**, and monitor the console output:

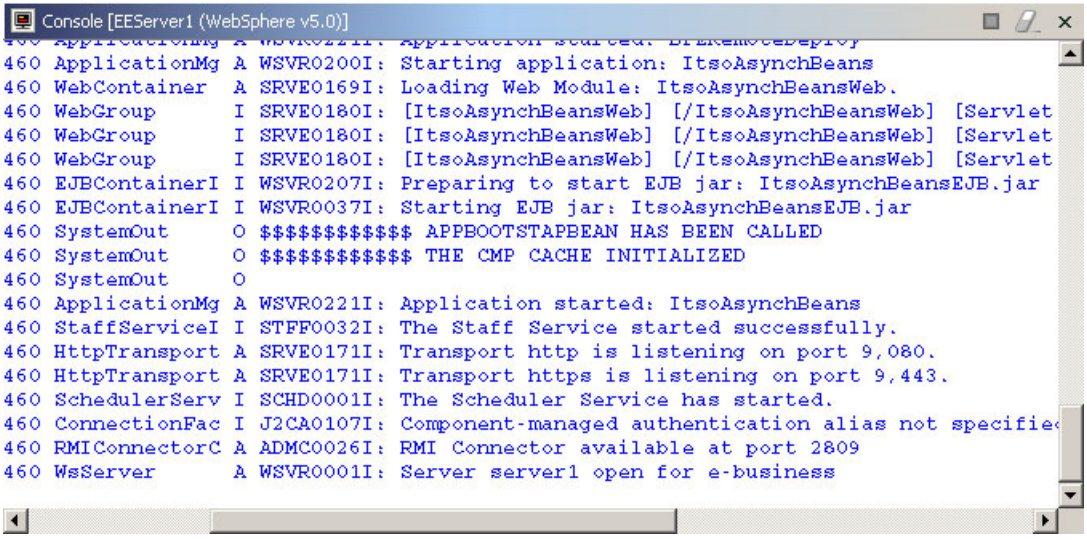


Figure 7-7 Server console with the base application

Important: As you can see from the server output, our Startup Bean AppBootstrapBean is invoked by WebSphere runtime, and it will delegate to our singleton cache class to call findAll() method of CMP Department's home interface, and store the result in the cache of our Singleton object.

2. Select the **ListDepartment.jsp**, right-click, and from the drop-down context, select **Run on Server**. You will see the window in Figure 7-8.

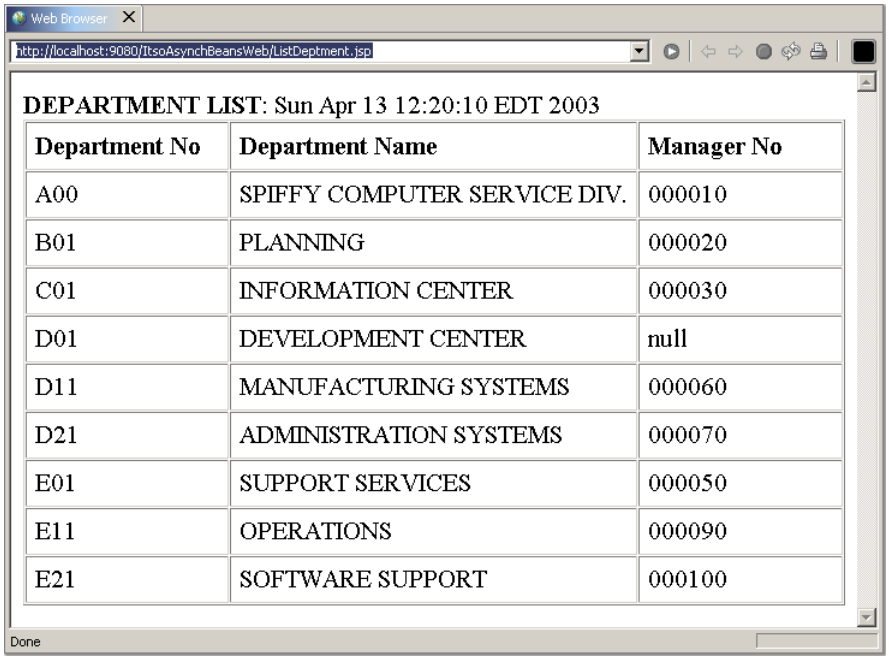


Figure 7-8 Base application department list display

3. You can reload this page, and you will always get the same result. Remember, our JSP is using cache for a CMP Department.

7.4 Development: "Asynchronize" the base application

In this section we enhance the base application and implement the features described in 7.2.2, "Asynchronous patterns" on page 305.

The ready-made extended application is also available as a WebSphere Studio IE workspace. For more information about how to use the extended application, refer to 7.4.4, "Set up the extended application" on page 325.

7.4.1 Asynchronously initialize the cache using Work

In this section, we will create a Work object InitializeCacheWork to do the background processing, and initialize our CMP cache asynchronously.

Create the Work in Studio

1. Switch to the **J2EE Navigator** view, and expand **ItsoAsynchBeanEJB**.
2. Right-click **ejbModule** and select **New** -> **Class**.
3. Specify `com.ibm.itso.was5e.asynbean` for the package and `InitializeCacheWork` for the class name.
4. Click **Add...** for the interfaces and select the **Work** interface on the subsequent window (start typing the word "`Work`" to facilitate the selection). Click **OK**.

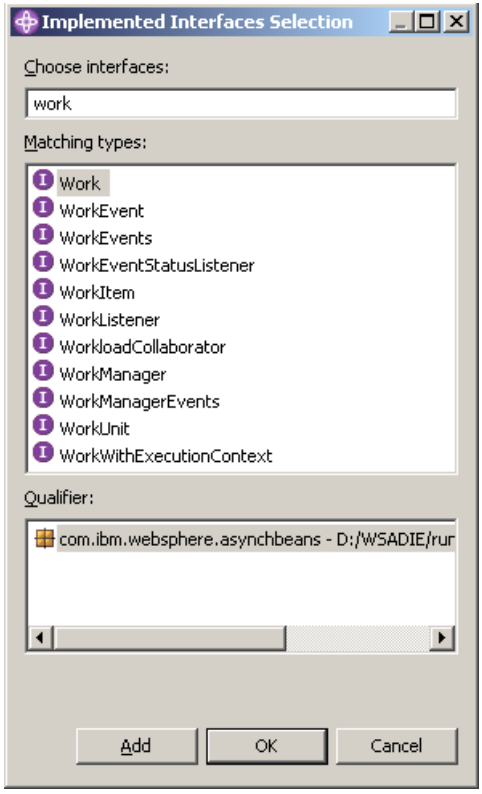


Figure 7-9 Add Work Interface

5. Click **Finish**.

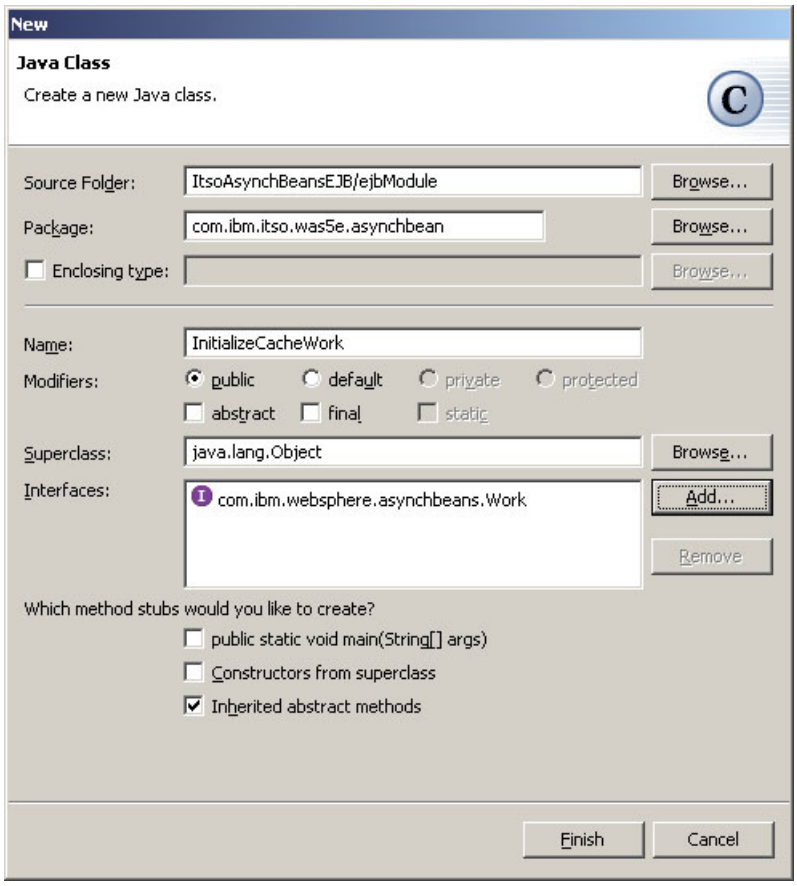


Figure 7-10 Create InitializeCacheWork class

6. The InitializeCacheWork.java source file should now be open in the source editor.
7. You should complete the Work class with its implementation of the run() method. For our Work object, you just need to add one line to this method as shown in the code sample in Example 7-8.

Example 7-8 the run() method of IntializeCacheWork object

```
package com.ibm.itso.was5e.asynchbean;
import com.ibm.websphere.asynchbeans.Work;

public class InitializeCacheWork implements Work {
   public void release() {
   }
   public void run() {
```

```
      EntityBeanCacheSingleton s = EntityBeanCacheSingleton.instance();
   }
}
```

8. Save and close the file.

Change the Startup Bean to use Work object

1. Open our singleton class AppBootstrapBean.java for editing.
2. Add the import statement to the code com.ibm.websphere.asychbeans.* package.
3. Comment out the line to call the Singleton class directly - EntityBeanCacheSingleton.instance() as in the base application.
4. Add the code to look up a work manager, create an instance of our IntializeCacheWork class, and then submit the work to the work manager to be processed asynchronously. When you finish, the start() method of the Startup Bean should look like Example 7-9.

Example 7-9 Modified start() method for the Startup Bean

```
public boolean start() {
  try {
    System.out.println(
      "$$$$$$$$$$$$ APPBOOTSTAPBEAN HAS BEEN CALLED.");

    //EntityBeanCacheSingleton s = EntityBeanCacheSingleton.instance();
    InitialContext ic = new InitialContext();

   //get the WM
   WorkManager wm =
   (WorkManager) ic.lookup("java:comp/env/wm/WorkManager");

   //Start an Asynchronous work to initialize the cache
   InitializeCacheWork pw = new InitializeCacheWork();
   wm.startWork(pw);
   System.out.println(
      "$$$$$$$$$$$$ STARTING ASYNCHRONOUS WORK TO INITIALIZE CACHE");

  } catch (Exception e) {
   System.out.println(
      "AppBootstrapBean.start(): Exception starting app: "
            + e.toString());
   e.printStackTrace();
   return false;
    }
   return true;
}
```

5. Save your file.

Test the application with asynchronous work

In this step, we need to create a resource reference from the Startup Bean to the WorkManager, and bind the resource reference to the actual resource JNDI name.

1. In the J2EE Hierarchy view, expand the **EJB modules** folder, and double-click **ItsoAsynchBeansEJB** to open it up with the Deployment Descriptor editor.
2. Switch to the References tab and select **AppBootstrap**.
3. Click **Add** and then select the **EJB Resource Reference** radio button. Click **Next**.
4. Type `wm/WorkManager` for the name. This is the name you used in the code, “java:comp/env/wm/WorkManager”.
5. Type `com.ibm.websphere.asynchbeans.WorkManager` for the type field. Do not try to select it from the pull-down list as this type is not available.
6. Select **Application** for the Authentication. Your window should look like Figure 7-11 on page 317.

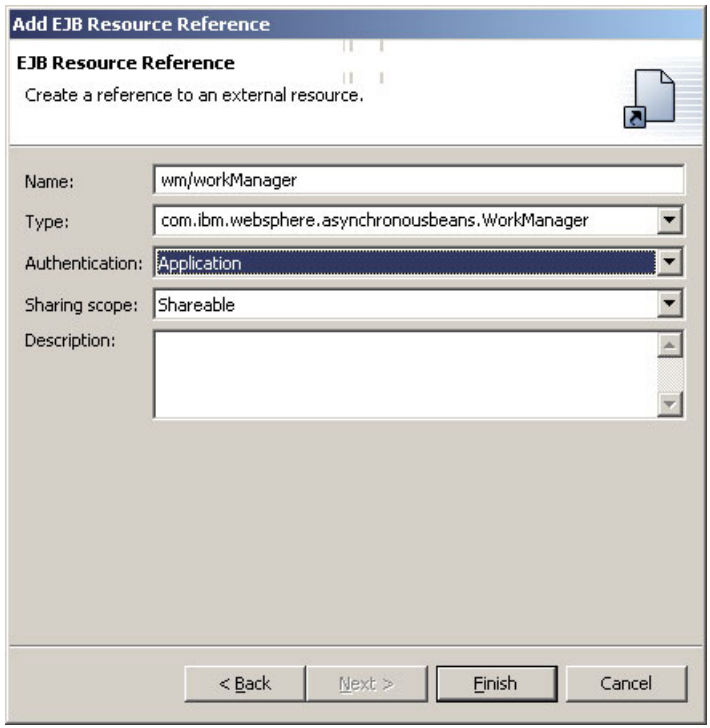


Figure 7-11 Define EJB resource reference

7. Click **Finish**.
8. Expand **AppBootstrap** and click the recently created reference.
9. On the right-hand side, you will see an empty field called JNDI name. Type `wm/default` in this field.

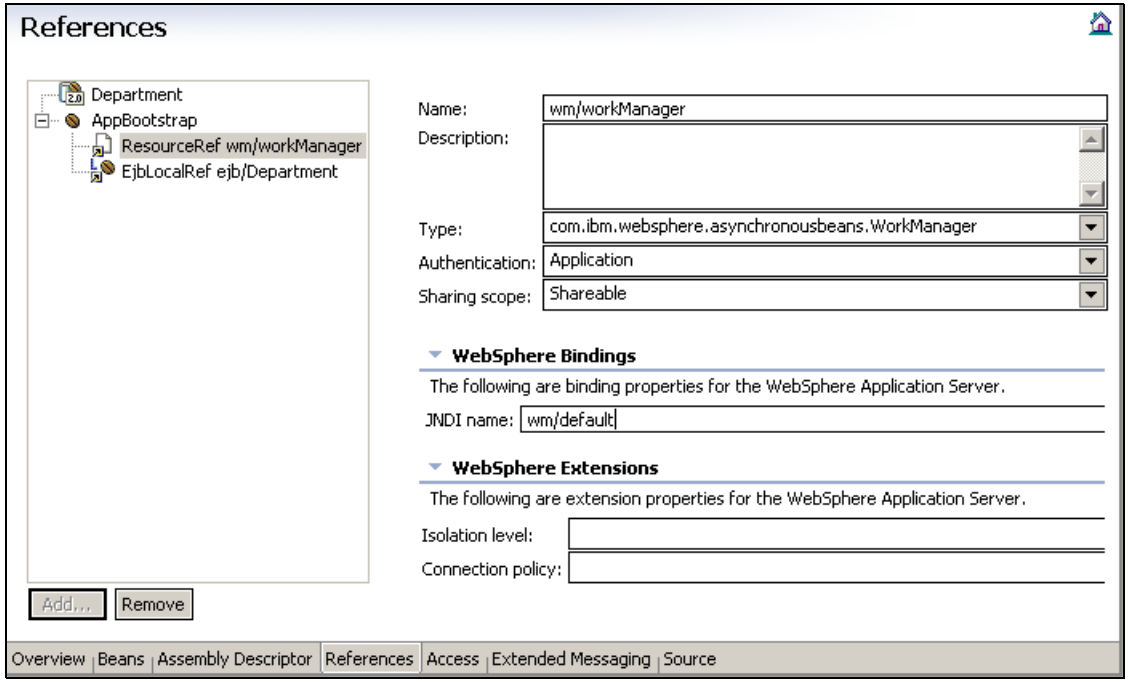


Figure 7-12 Bind resource reference to JNDI name

10. Save the EJB Deployment Descriptor.
11. The next step is to test the application with the asynchronous work implemented. Start the test server and monitor the console output.

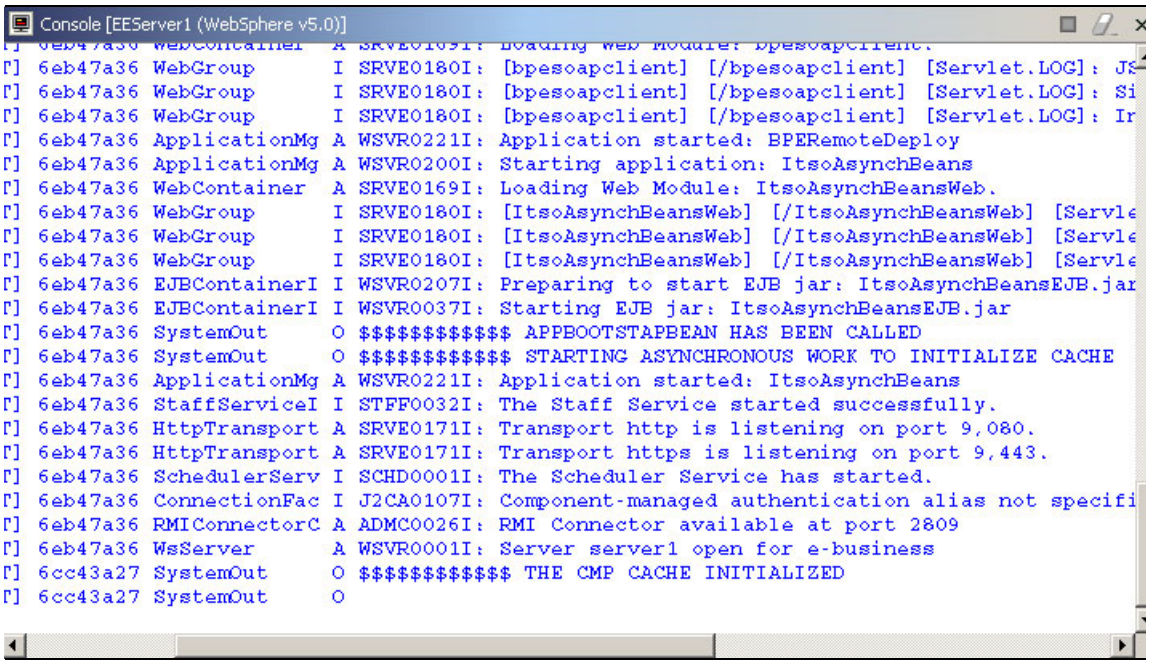


Figure 7-13 Server console with asynchronous work

Important: If you look carefully, you will notice that Cache is now initialized after the main server thread finishes with the message "`Server server1 open for e-business`". It is also executed on a different thread. You could tell this by looking at the thread hash number. This tells us that the operation of retrieving cache is done in background asynchronously, not blocking the main thread execution. Compare this with Figure 7-7 on page 311.

12. Select the ListDepartment.jsp, right-click, and from the drop-down context, select **Run on Server,** and you will get the same result as shown in Figure 7-17 on page 327.

7.4.2 Asynchronously keep cache updated using EventListener

Asynchronous Beans provide a generic event notification framework. The EventSource provides "type-safe" notification. You can ask an EventSource to fire an event and target only the listeners that implement a certain interface.

To facilitate intra-application notification, WebSphere Enterprise V5 provides a special type of EventSource, which is included in each enterprise application.

This EventSource can be found using JNDI lookup in any servlet or EJB code in the application.

You can follow the steps in the next section to implement this pattern in our sample application.

Define the EventListener interface

1. Switch to the **J2EE Navigator** view, and expand **ItsoAsynchBeanEJB**.
2. Right-click **ejbModule** and select **New** -> **Interface**.
3. Specify `com.ibm.itso.was5e.asynbean` for the package and `UpdateCacheEventListener` for the type name, and click **Finish**.
4. The interface file should be open in the editor. We need to add just one method in the interface: public void updateEntityCache().

Implement the EventListener interface

The EntityBeanCacheSingleton object in our base sample is interested in the updateEntityCache event, so it will implement the interface UpdateCacheEventListener.

1. Open the source code of the singleton class in an editor.
2. First we add to the end of the line of class declaration: `implements UpdateCacheEventListener`
3. The updateEntityCache() method should be implemented as shown in Example 7-10.

Example 7-10 Implementing updateEntityCache() method for the EventListener

```
public void updateEntityCache() {
  System.out.println("$$$$$$$$$$$$ HANDLE ASYNCHRONOUS EVENT: UPDATING
CACHE......");
  setDeptCache(getDepartments());
  System.out.println("$$$$$$$$$$$$ THE CMP CACHE UPDATED.");
  System.out.println();
}
```

Register the Listener with Application EventSource

Now that we have implemented the listener, we still have to register it with our application-specific EventSource to indicate the interest in receiving the updateEntityCache event. This is done inside of the Singleton object constructor.

Example 7-11 Register listener in singleton constructor

```
protected EntityBeanCacheSingleton() {
  try {
```

```
    deptCache = this.getDepartments();
    System.out.println("$$$$$$$$$$$$ THE CMP CACHE INITIALIZED");
    System.out.println();

    //regiter this as a listener to asynchronous event
    InitialContext ic = new InitialContext();
    EventSource appES =(EventSource) ic.lookup(
          EventSource.APPLICATION_NOTIFICATION_EVENT_SOURCE);
    appES.addListener(this);
   } catch (Exception e) {
   e.printStackTrace();
   }
}
```

Tip: Inside the code, you can also do a lookup like this:

```
ic.lookup("java:comp/websphere/ApplicationNotificationServer")
```

EventSource.APPLICATION_NOTIFICATION_EVENT_SOURCE is just a convenient static constant.

At this stage, our EventListener EntityCacheSingleton is all set and ready to handle any updateEntityCache event. In the next section, we will create an AlarmListener, which simulates updating database and firing the updateEntityCache event.

7.4.3 Asynchronously update database using AlarmListener

AlarmListener is the third kind of Asynchronous Bean. Its logic gets executed when the associated alarm goes off. In this example, we create an alarm that fires every 30 seconds, so our listener UpdateDbAlarmListener would get invoked every 30 seconds also. The EventListener/Asynchronous Bean will try to insert a record into the database through the Department entity bean, and then get hold of the application EventSource to fire the updateEntityCache event.

Develop the AlarmListener

1. Switch to the **J2EE Navigator** view, and expand **ItsoAsynchBeanEJB**.
2. Right-click **ejbModule** and select **New** -> **Class**.
3. Specify `com.ibm.itso.was5e.asynbean` for the package and `UpdateDbAlarmListener` for the class name.
4. Click **Add** for the interfaces and select the **AlarmListener** interface on the subsequent window (start typing the word "`AlarmListener`" to facilitate the selection). Click **OK**.

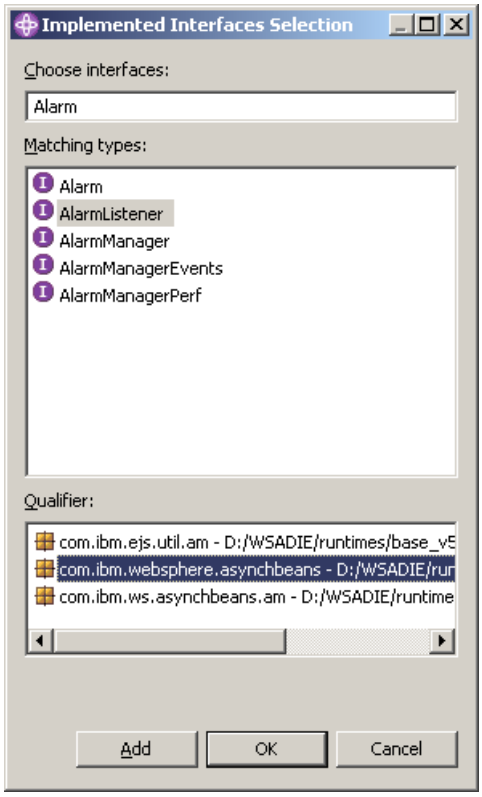


Figure 7-14 Add AlarmListener Interface

Important: Make sure to select the **AlarmListener** interface in the com.ibm.websphere.asynchbeans package.

5. Click **Finish**.

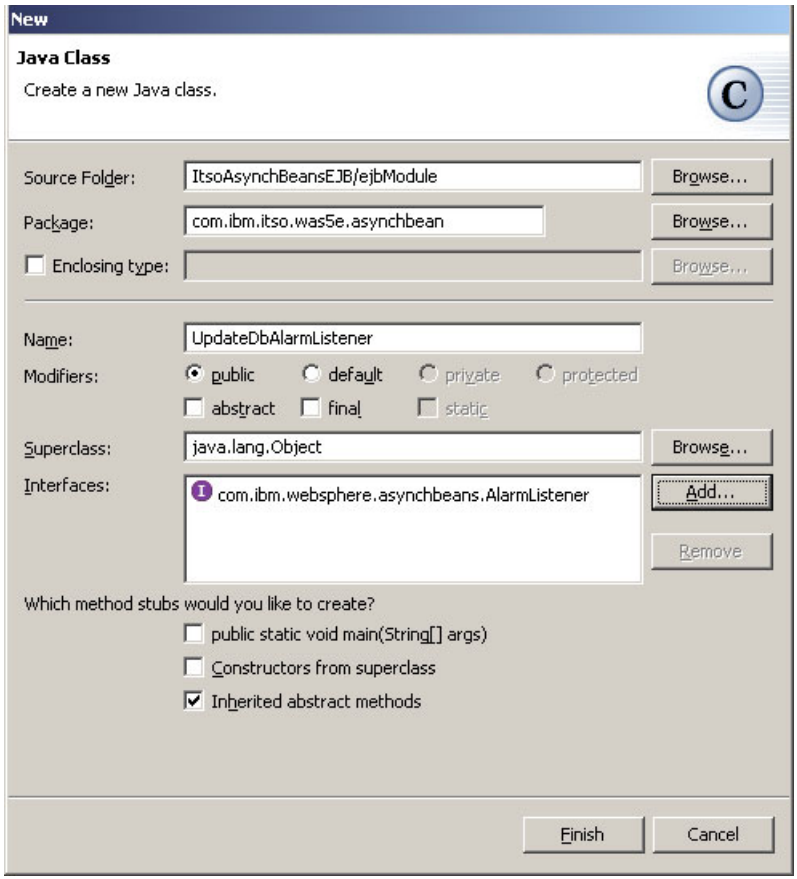


Figure 7-15 Create UpdateDbAlarmLister class

6. The UpdateDbAlarmListener.java source file should now be open in the source editor.
7. You should complete the AlarmListener class with its implementation of the fired() method.
8. First we need to import javax.naming.* and com.ibm.websphere.asynchbeans.EventSource. This is needed because we will fire asynchronous event in this Alarm listener.

Example 7-12 fired() method for AlarmListener

```
public class UpdateDbAlarmListener implements AlarmListener {
  static int  i = 20;
  public void fired(Alarm arg0) {
    try {
     System.out.println("$$$$$$$$$$$$ UPDATEdbALARM FIRED.");
```

```
    System.out.println("$$$$$$$$$$$$ HANDLE ALARM: INSERTING NEW ROW INTO
TABLE, AND .....");
    InitialContext ic = new InitialContext();
   DepartmentLocalHome cbHome =
      (DepartmentLocalHome) ic.lookup("java:comp/env/ejb/Department");
   DepartmentLocal c = cbHome.create("F"+i, "MATH--" +i, "");
   i++;

     System.out.println("$$$$$$$$$$$$ FIRING ASYNCHRONOUS UPDATE CACHE
EVENT......");
     EventSource appES =(EventSource) ic.lookup(
               EventSource.APPLICATION_NOTIFICATION_EVENT_SOURCE);
   UpdateCacheEventListener eventProxy =
            (UpdateCacheEventListener) appES
.getEventTrigger(UpdateCacheEventListener.class);
   eventProxy.updateEntityCache();
   System.out.println();

   //reset the Alarm;
   arg0.reset(30000);

   }catch (Exception e) {
       e.printStackTrace();
   }
 }
}
```

Note: The listener code first invokes the create() method of the entity bean's home interface, and this will actually insert a row into the database. After the database has been updated, we want to notify all registered event listeners about the updateEntityCache event. We achieve this first by looking up the application EventSource, getting a proxy for our listener interface (this step essentially does the filtering), and then calling the updateEntityCache() method on the proxy, which will propagate the calls to every registered listener, including our EntityBeanCacheSingleton listener.

Develop the Alarm

We have developed our AlarmListener UpdateDbAlarmListener**,** and now we need to associate it with an Alarm. As we have mentioned, the Alarm will go off every 30 seconds, and it is created in our Startup Bean when the server starts.

1. Open the Startup Bean **AppBootstrapBean.java** for editing.
2. We need to add the code in Example 7-13 on page 325 to the start() method of the bean just below where we start the asynchronous work.

Example 7-13 Code to create Alarm

```
//get the AS
AsynchScope as = wm.findAsynchScope("ItsoScope");
if (as == null) {
   as = wm.createAsynchScope("ItsoScope");
}
//get the AlarmManager
AlarmManager am = as.getAlarmManager();
UpdateDbAlarmListener updateListener = new UpdateDbAlarmListener();
//create an Alarm to go off every 30 seconds
Alarm a1 = am.create(updateListener, this, 30000);
```

3. In order to create an Alarm, we need to get an instance of AlarmManager from an AsynchScope, which is created by the work manager.
4. Save and close the file.

7.4.4 Set up the extended application

The extended application developed in this section is available as a copy of WebSphere Studio IE workspace.

The extended application does not require any additional configuration compared to the base application.

For instructions on importing the workspace, and creating and configuring the test server, see 7.3.1, “Set up the base application” on page 306.

7.5 Unit test environment

We have enhanced our base application with some asynchronous behaviors, and we are ready now to test the application.

1. Start the test server and monitor the console. When the server finishes, you will see the same output as in Figure 7-16 on page 326. We know our CMP cache initialized immediately after the server runtime is up.
2. Right-click the **ListDepartment.jsp**, and from the drop-down menu, select **Run on Server,** and you will also see the same result in the browser as shown in Figure 7-17 on page 327.
3. Wait approximately 30 seconds. If you check the console, it will look like Figure 7-16 on page 326.

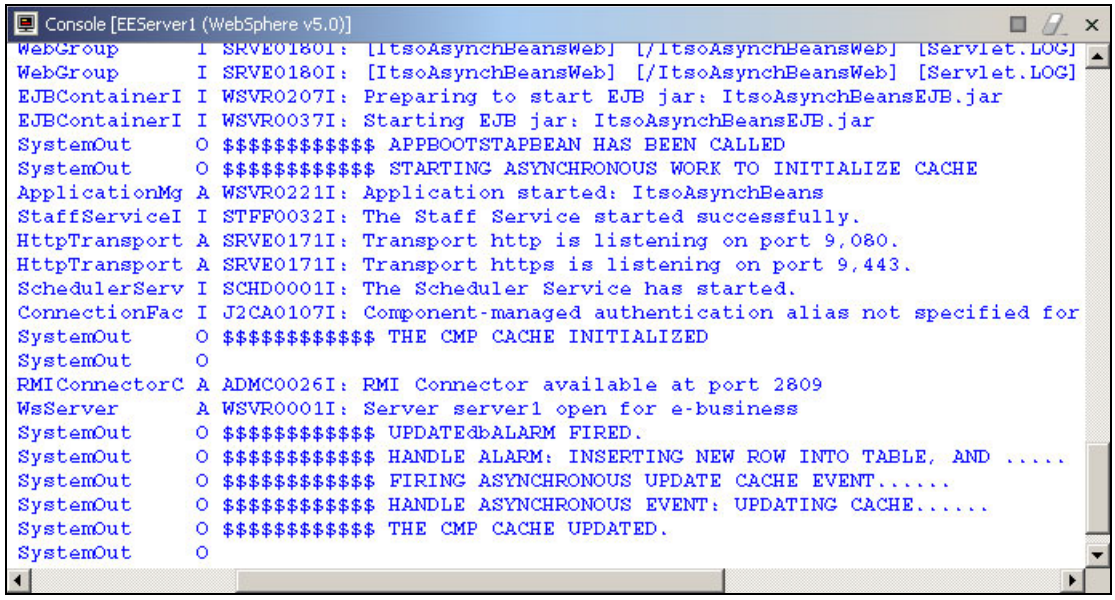


Figure 7-16 Server console with Asynchronous functions

4. You can tell from the output when our Alarm fires, the database gets updated, and an event is sent out to refresh the CMP cache.
5. If you reload the ListDepartment.jsp now, you will see some new records in the table. Remember, our JSP file always pulls data from the cache. This time our cache gets refreshed, so we will not get stale data.

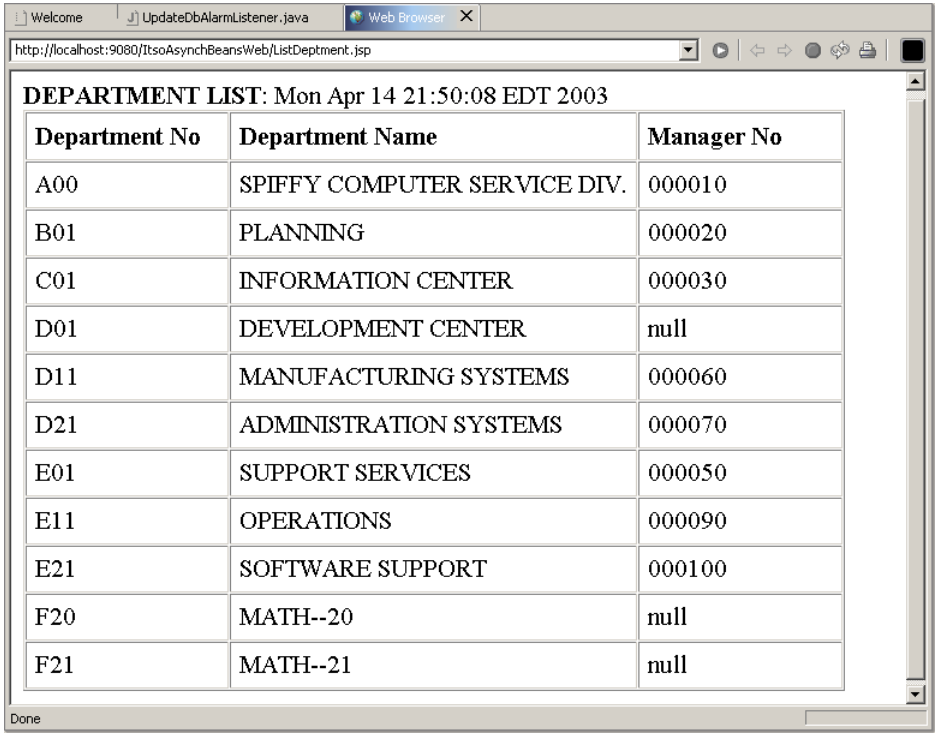


Figure 7-17 Department listing

6. If you wait for a while, and go back to check the console, you will notice our Alarms keep going off, and our cache get updated whenever there is a database update.

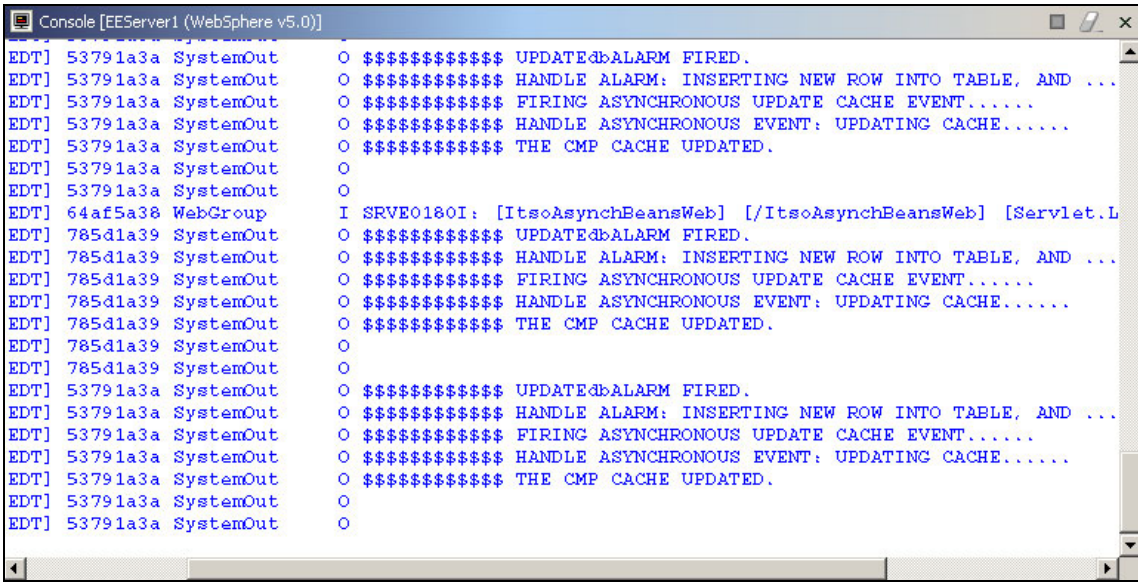


Figure 7-18 Server console with Asynch functions

7. When we reload our JSP, you will see more records in the tables.

7.6 Assembly

We need to package and export our application for installation in WebSphere.

1. Click **File -> Export**.
2. Select **EAR File** and click **Next**.
3. Select **ItsoAsynchBeans** for the resource to export, and browse to directory where you want to save the ItsoAsynchBeans.ear file.
4. Select **Export Source Files**.
5. Click **Finish**.

When we test the application in WebSphere Studio IE, we bind our WorkManager reference to the default WorkManager defined in a server with the JNDI name of wm/default. If you have created another physical WorkManager with a different JNDI name, and want to use that, you need to do the following steps:

1. Launch the Application Assembly Tool.

2. On the resulting windows, click the **Existing** tab, and click **Browse** to find the **ItsoAsynchBeans.ear** we exported just now, and click **OK**.
3. Expand **ItsoAsynchBeans -> EJB Modules -> ItsoAsynchBeansEJB -> SessionBeans -> AppBootstrap**, and select **Resource Reference**.

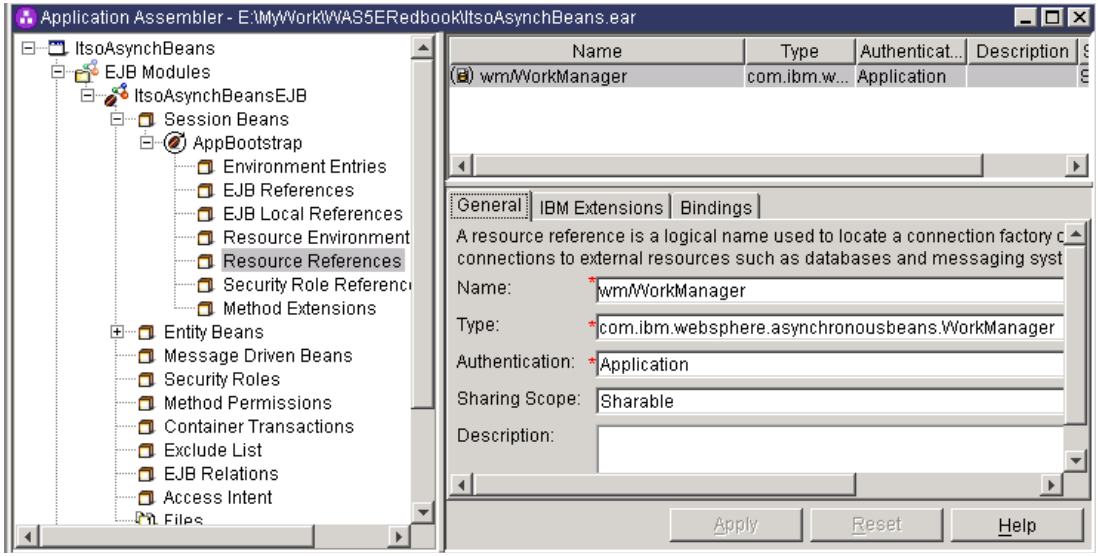


Figure 7-19 Application Assembly Tool

4. Switch to the Bindings tab, change the JNDI name of WorkManager to whatever you have defined in the WebSphere Application Server. In our case, the name is wm/ItsoAsynchBeansWM. Click **Apply**.
5. Save and close the EAR file.

7.7 Configure

The starting point for configuration is the ItsoAsynchBeans EAR file. We will first create a data source and WorkManager for the application using the WebSphere Enterprise Administrative Console. Subsequently, we will install the application in WebSphere Application Server and test it.

Create a data source for the ItsoAsynchBeans application

1. Start WebSphere Application Server Enterprise. Open a command prompt and change the directory to \WebSphere\Application Server\bin, and issue the following command:

```
startServer server1
```

2. Launch the Administrative Console for WebSphere, then log in.
3. Create a new data source for the server with the following details:
   - Provider: `DB2 JDBC Provider`
   - Classpath `C:/sqllib/java/db2java.zip`, or where your db2java.zip file resides.
   - Data source name: `SampleDS`
   - JNDI name: `jdbc/Sample`

   Check **Use this Data Source in CMP**.

   Click **OK**.
4. Click the Data Source you just created and scroll down. Click **J2C Authentication Data Entries**. Create a new entry with the following details:
   - Alias: `SampleDSAlias`
   - User ID: The name of user who created the sample database, in our case `db2admin`
   - Password: password for the user ID, in our case `passw0rd`
5. Click the **SampleDS** link at the top of the frame to go back to the data source definition. Scroll down, select **<node name>/SampleDSAlias** for the Component-managed Authentication Alias.
6. Click **OK**.
7. Save the configuration for WebSphere.

Create a WorkManager for the ItsoAsynchBeans application

1. Expand **Resources** and click **WorkManager.** Click **New** to create a new item. Use the following details for the entry:
   - Name: `ItsoAsynchBeansWM`
   - JNDI name: `wm/ItsoAsynchBeansWM`
   - Number Of Alarm Threads: `5`
   - Minimum Number Of Threads: `1`
   - Maximum Number Of Threads: `10`
   - Thread Priority: `2`

   Check the **Security** check box in the Service Names field. This will ensure the security context, if any, will be propagated to the Work implementation.

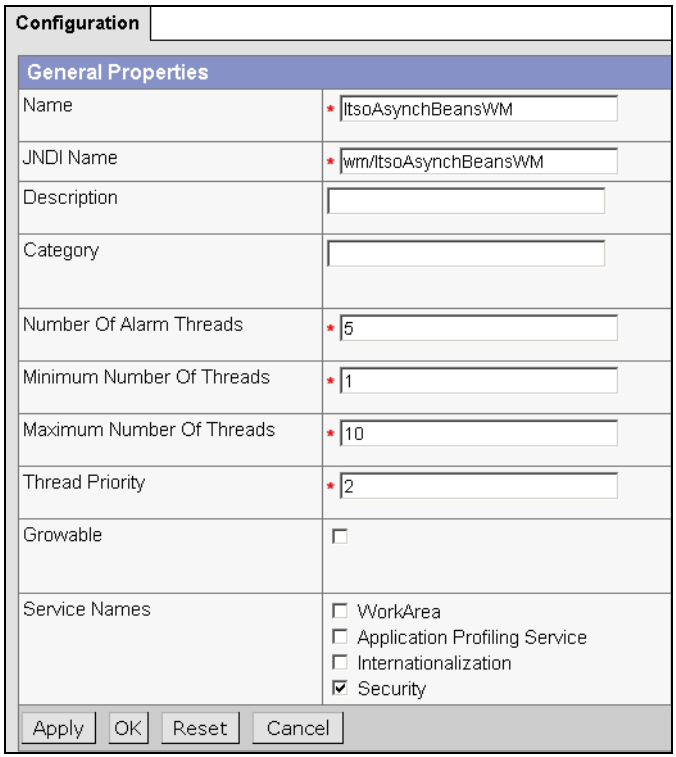


Figure 7-20 WorkManager definition

2. Save the configuration for WebSphere.
3. Restart the application server.

7.8 Deployment

We are ready now to install and test our application in WebSphere Enterprise Server.

1. Launch the Administrative Console and log in.
2. Expand **Applications** and then click **Install New Application**.
3. In the main window, click **Browse**, and find the ItsoAsynchBeans.ear file.
4. Click **Next,** and click **Next** again, since we do not need to change anything.
5. Click **Step 3: Provide default datasource mapping for modules containing 2.0 entity beans**, specify the data source JNDI name `jdbc/Sample`, which we created in the previous section.

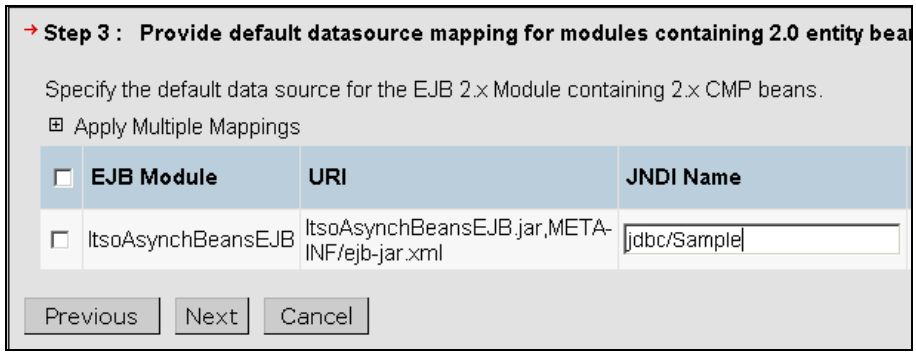


Figure 7-21 Data source mapping

6. Click **Step 9**, the last step, and click **Finish**.
7. Once the deployment is finished, save the configuration for WebSphere.
8. The next step is to start the application. Select **Applications -> Enterprise Applications.**
9. From the list of enterprise applications already installed, check **ItsoAsynchBeans** and click **Start**.
10. Our application is now loaded and running. If you check the SystemOut.log in the server logs directory, you should find the following lines.

Example 7-14 Server SystemOut.log

```
[4/15/03 21:29:35:288 EDT] 5c8c3c8a WebGroup      I SRVE0180I: [ItsoAsynchBeansWeb]
[/ItsoAsynchBeansWeb] [Servlet.LOG]: InvokerServlet: init
[4/15/03 21:29:35:308 EDT] 5c8c3c8a SystemOut     O $$$$$$$$$$$$ APPBOOTSTAPBEAN HAS BEEN
CALLED
[4/15/03 21:29:35:338 EDT] 5c8c3c8a SystemOut     O $$$$$$$$$$$$ STARTING ASYNCHRONOUS WORK TO
INITIALIZE CACHE
[4/15/03 21:29:35:338 EDT] 5c8c3c8a ApplicationMg A WSVR0221I: Application started:
ItsoAsynchBeans
[4/15/03 21:29:37:241 EDT] 2059fcaf SystemOut     O $$$$$$$$$$$$ THE CMP CACHE INITIALIZED
[4/15/03 21:29:37:241 EDT] 2059fcaf SystemOut     O
[4/15/03 21:30:05:341 EDT] 353c3ca1 SystemOut     O $$$$$$$$$$$$ UPDATEdbALARM FIRED.
[4/15/03 21:30:05:341 EDT] 353c3ca1 SystemOut     O $$$$$$$$$$$$ HANDLE ALARM: INSERTING NEW
ROW INTO TABLE, AND .....
[4/15/03 21:30:05:351 EDT] 353c3ca1 SystemOut     O $$$$$$$$$$$$ FIRING ASYNCHRONOUS UPDATE
CACHE EVENT......
[4/15/03 21:30:05:361 EDT] 353c3ca1 SystemOut     O $$$$$$$$$$$$ HANDLE ASYNCHRONOUS EVENT:
UPDATING CACHE......
[4/15/03 21:30:05:542 EDT] 353c3ca1 SystemOut     O $$$$$$$$$$$$ THE CMP CACHE UPDATED.
```

11. This log tells us that our CMP cache is loaded asynchronously when our application starts, and also the cache is updated whenever our Alarm fires.
12. So when our JSP file is loaded now, we are sure it is always refreshed data, even if the database is updated.

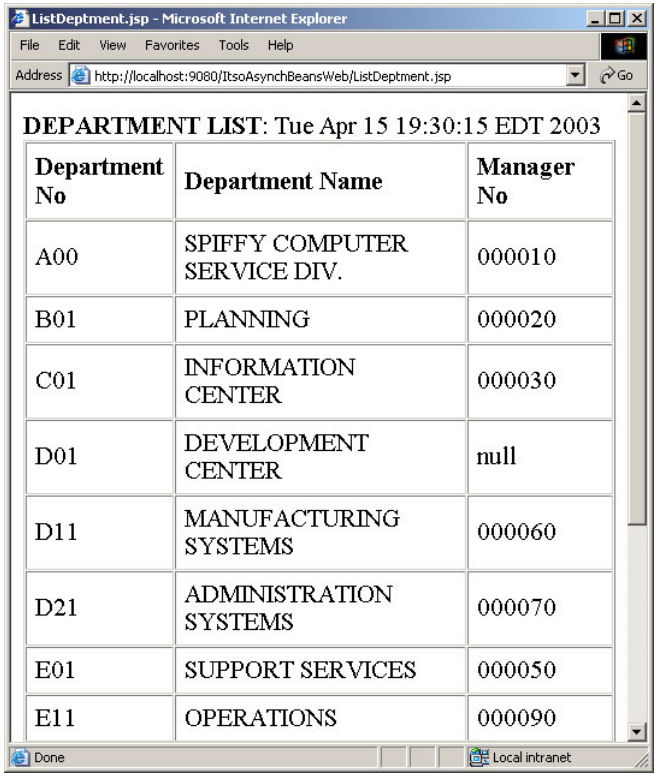


Figure 7-22 Department list

13. If you keep reloading the JSP file every 30 seconds, you will see the list changing.

7.9 QoS (Quality of Service) considerations

This section discusses the Quality of Service (QoS) considerations for Asynchronous Beans.

7.9.1 Multiple WorkManagers

An application will sometimes wish to reserve resources for various qualities of service. An application can specify the need for multiple WorkManagers by having more than one resource-ref for WorkManager resources.

When an application wants to execute multiple operations in parallel, then it uses a WorkManager to start a Work for each operation. The administrator can specify a maximum number of threads for the WorkManager. If this is fixed at N threads, then at most N operations will be executed in parallel. If there are more than N items, then N are executed first and the others queued. The queued Works are executed in sequence as the N completes.

So, a WorkManager can be used by an administrator to limit the number of concurrent parallel tasks. Therefore, an application can specify more than one resource-ref and then choose one depending on the classification of the Work. This gives great flexibility to the administrator.

Small machines configuration

The administrator can choose to make a single WorkManager and then bind it to all resource-refs in the application. So, the developer has specified and uses multiple “logical” WorkManagers, but the administrator has decided to use a single WorkManager for all.

Large machines configuration

The administrator makes a WorkManager for each quality of service. He/she also sizes each WorkManager. The user is free to set a maximum number on the threads in the pool as well as the thread priority. The administrator then binds each WorkManager to the appropriate resource-ref when the application is deployed.

This allows the administrator to carefully tune each WorkManager to the quality of service.

7.9.2 Dynamically tuning WorkManagers at runtime

There is an MBean available for each WorkManager in a JVM. There is actually an MBean for each thread pool in a JVM. This MBean allows an administrator using wsadmin to tune the size of a WorkManager at runtime without restarting the server. These changes are purely transient and will be lost if the server is restarted. The admin object for the WorkManager should be modified if the changes are to be permanent. For details on how to use wsadmin and MBeans, refer to the WebSphere System Administration documentation.

7.10 Security considerations

Using Asynchronous Beans technology, the security context is stored in the database between each call and reused when the next activity is executed. Asynchronous Beans have a feature that allows Work objects to store the associated security context and rebuild and reuse it later.

We discussed some security issues associated with Asynchronous Beans in 7.1.4, “Asynchronous Beans: programming model” on page 300. Refer to that section.

7.11 An additional sample

This additional sample for Asynchronous Beans is part of the extended sample application. For further details about using the extended sample application, refer to Appendix B, “Sample scenario” on page 665.

In the sample application, some of the purchase orders need approval if the price is higher than a certain amount. The approver checks if the request is acceptable. At this time, the approver might need to check if there are better products. For this reason, we provide the product search UI for the approver. On this UI, the users can search the product information from four different resources: the catalog of this system, the other catalogs provided by Company A and Company B, and Amazon.com. The following steps show the usage of the new UI.

1. Invoke a Web browser and open the following page:

   `http://<your_server>/approval`

2. Enter the keyword, select the resource that you want to search, and click **Search**.

Figure 7-23 Search window

3. The result window is returned even if all searches are not done. If there are unfinished searches, the page shows "`Searching...`" in each result field. This page is reloaded until all searches are done.

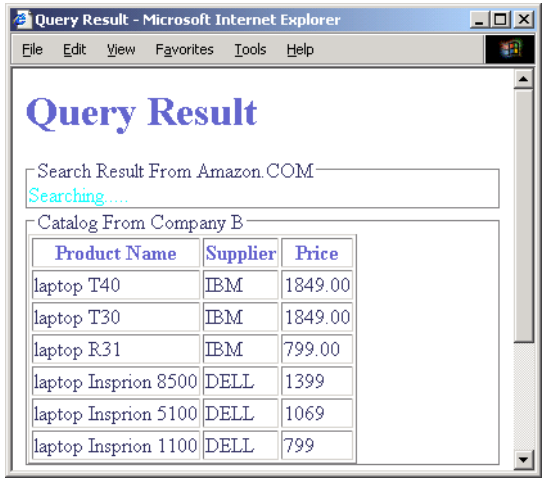


Figure 7-24 Result window

On the traditional Web site, the result is blocked until all queries are done. By using Asynchronous Beans, the users can get part of the results without waiting for all queries to finish. Of course, because of the parallel execution, the queries can finish faster than the sequential execution.

7.11.1 Implementation details

The SearchServlet accepts the request from the query page. The keyword and the targets of the queries are sent on the request. The SearchServlet instantiates the objects that run queries, passes the keyword and starts the query. The queries run asynchronously and SearchServlet forwards the request to result.jsp without waiting for the queries to finish. The result.jsp gets the results of queries from SearchWork. If the query haven't finished, the result.jsp returns "`Searching...`" instead of the result of query.

The result page is reloaded every 5 seconds until all queries are finished.

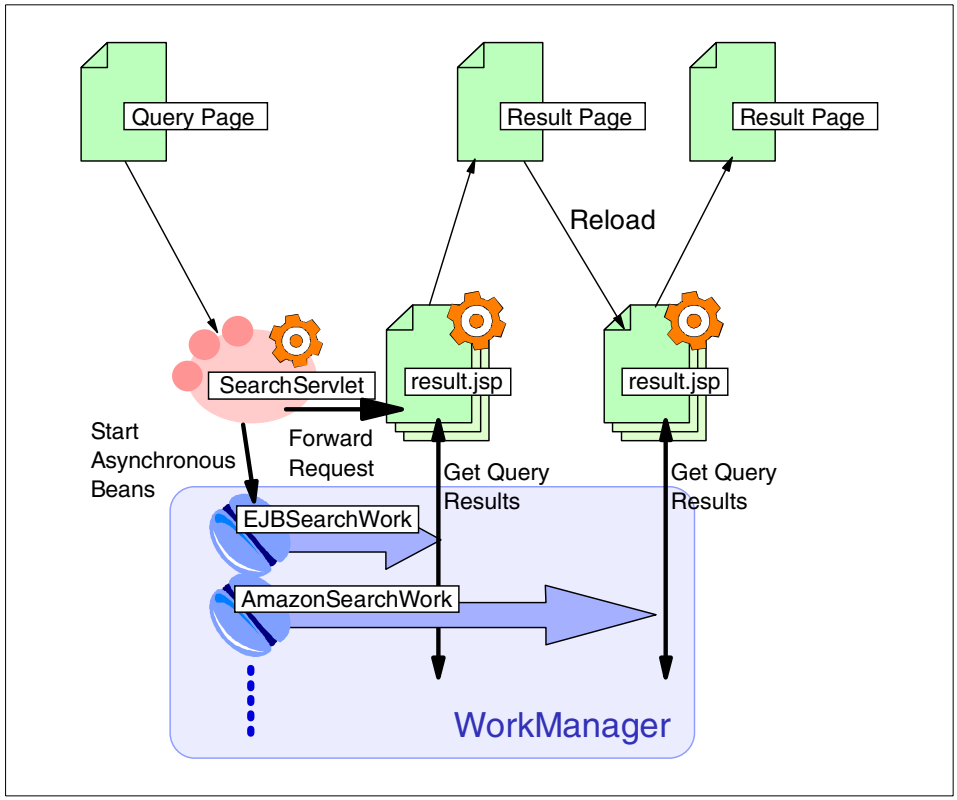


Figure 7-25 The flow of asynchronous search

The Java source codes, JSPs and the Deployment Descriptors are in the ACompanyApprovalWeb project. To compile this project, asynchbeans.jar and wsexception.jar must be in your CLASSPATH, and i18nctx.jar also must be in it because this project has a dependency on the Internationalization service.

SearchServlet

This servlet is invoked by the request from the query page. The servlet gets the keyword and the list of target from the request.

The servlet instantiates the SearchWork objects. The SearchWork extends com.ibm.websphere.asynchbeans.Work interface. Each SearchWork object is an Asynchronous Bean and implements a search of each resource.

Each SearchWork starts on the WorkManager. The WorkManager is obtained from the JNDI repository.

After starting the queries, the servlet forwards the request to the result.jsp.

The code of this servlet is shown in Example 7-15.

Example 7-15 SearchServlet.java

```
......
public class SearchServlet extends HttpServlet {
......
   public void doPost(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {
      req.setCharacterEncoding("UTF8");
      String keyword = req.getParameter("keyword");
      String[] target = req.getParameterValues("target");

      Map queryMap = new HashMap();
      if (target != null && target.length > 0) {
         try {
            InitialContext ctx = new InitialContext();
            WorkManager wm =
               (WorkManager) ctx.lookup("java:comp/env/wm/default");

            for (int i = 0; i < target.length; i++) {
               SearchWork work = null;
               if (target[i].equals("ejb")) {
                  work = new EJBSearchWork();
                  queryMap.put("ejb", work);
               } else if (target[i].equals("file")) {
                  work = new FileSearchWork();
                  queryMap.put("file", work);
               } else if (target[i].equals("jdbc")) {
                  work = new JDBCSearchWork();
                  queryMap.put("jdbc", work);
               } else if (target[i].equals("amazon")) {
                  work = new AmazonSearchWork();
                  queryMap.put("amazon", work);
               }
               if (work != null) {
```

```
                    work.initialize(keyword);
                    wm.startWork(work);
                 }
              }
           } catch (NamingException ne) {
              throw new ServletException(ne);
           } catch (WorkException we) {
              throw new ServletException(we);
           }
        }

        HttpSession session = req.getSession();
        session.setAttribute("queryMap", queryMap);

        ServletContext context = getServletContext();
        String jspFile =
           I18nUtils.getI18nUtils().getLocalizedFileName(
              "/result",
              ".jsp",
              context);
        RequestDispatcher rd = context.getRequestDispatcher(jspFile);
        rd.forward(req, resp);
     }
}
```

EJBSearchWork

When you select “Our Catalog” on the search page, this class is called. This class is an Asynchronous Bean. The search function is implemented in the run() method and is called asynchronously.

The catalog data is stored as an entity bean named Catalog. The search function calls the finder method findByKeyword(String keyword). It returns the list of EJBs and the query result is retrieved from these EJBs.

When using a Java object as an Asynchronous Bean, it only has a local transaction on the thread. This means that if it makes a call to an EJB method that uses TX_REQUIRES, then the container will wrap that call in its own global transaction that will be committed when the method returns. When you call the finder method on the CMP beans, then access the returned CMP beans, it fails because the transaction is already committed when the finder method returns. In this case, you must have a global transaction surrounding the finder and the codes using the returned CMP beans.

JDBCSearchWork

When you select **Catalog From Company A** on the search page, this class is called. This class is an Asynchronous Bean. The search function is implemented in the run() method and is called asynchronously.

The catalog data is stored in one table in the database. The search function retrieves the data by sending the SQL query. This class uses the JDBC functions.

FileSearchWork

When you select **Catalog From Company B** on the search page, this class is called. This class is an Asynchronous Bean. The search function is implemented in the run() method and is called asynchronously.

The catalog is stored in a file as the comma-separated data. The search function opens the file and reads it sequentially. On each line, the search function pattern-matches with the keyword.

AmazonSearchWork

When you select **Search From Amazon.COM** on the search page, this class is called. This class is an Asynchronous Bean. The search function is implemented in the `run()` method and is called asynchronously.

The search function calls the keyword search function of Amazon Web Services to get the product information. The client proxy of the Web Service is created by WebSphere Studio using WSDL which is provided by Amazon.COM. For more detail about the Web Services, see the redbook *WebSphere Version 5 Web Services Handbook,* SG24-6891.

For more detail about the Amazon Web Services, see the following Web site:

`http://associates.amazon.com/exec/panama/associates/ntg/browse/-/1067662/ref=gw_hp_ls_1_3/`

result.jsp

This page shows the results of the queries. The result of each query is stored in HttpSession and is shown in the separated JSP, which is included into this page. If all queries are not done, the following element is put at the head element. Because of this element, this page is reloaded every 5 seconds until all queries are done. The element for reloading the page in HTML code is:

```
<META HTTP-EQUIV="REFRESH" CONTENT="5;URL=result.jsp">
```

7.11.2 Configuration and requirements

This application uses a WorkManager, a data source and an EJB. The references to these resources must be in the web.xml. The details of the resources are the following:

- The SearchServlet accesses a WorkManager to execute the Asynchronous Beans. We use the default WorkManager and need to make the reference to it. The name of the reference is wm/default, its JNDI name is wm/default, and its type is com.ibm.websphere.asynchbeans.WorkManager.
- The JDBCSearchWork accesses a data source to retrieve the data from the database. We use the same data source as the EJBs use and need to make the reference to it. The name of the reference is jdbc/redbookDS, its JNDI name is jdbc/redbookDS and its type is javax.sql.DataSource.
- The EJBSearchWork accesses the Catalog entity bean to get the catalog information via local home and local interfaces. We need to make the local reference to it. The name of the reference is ejb/Catalog, the JNDI name is ejb/Catalog and it links to the Catalog bean in the ACompanyEJB.jar.

To run this application, you must have an installation of the WebSphere Enterprise Version 5 configured with the Scheduler and Asynchronous Beans option. After you install WebSphere Enterprise Version 5, you must configure one data source that is bound to jdbc/redbookDS on the JNDI repository. If you need more information about the installation, see Appendix A, “Installation and configuration” on page 653.

The JDBCSearchWork accesses one table in the REDBOOK database. The table is created during the normal database setup process for the sample application.

8

Application Profiling and Access Intent

This chapter introduces the benefits of Application Profiling and Access Intent and how this feature of WebSphere Application Server Enterprise V5 fits your business needs.

This chapter covers the following topics:

- What Application Profiling and Access Intent is
- Why you should use Application Profiling and Access Intent
- How Application Profiling and Access Intent answers your needs
- Detailed description of this Programming Model Extension

This chapter is organized different from other PME chapters, because this PME does not require configuration, deployment, or runtime discussion. It is purely related to application design, development, and assembly.

This chapter also uses a different sample application that does not relate to the sample scenario used in the majority of the book. This PME requires a different set of components from the components in the ACompany sample application.

8.1 Overview

One of the main drawbacks of EJBs has always been performance. EJBs are reusable components by nature. But this reusability introduces a performance problem. In a common scenario, different clients access the same set of Entity EJBs. The intent of each client is completely different.

WebSphere Enterprise V5 adds the capability to the EJB container to be configured to provide optimal performance based on a specific type of EJB use. Various Access Intent hints can be declared at assembly time to indicate to WebSphere resources (such as the container and the persistence manager) to provide the appropriate Access Intent services for every EJB request.

Access Intent Policies are hints not only to the persistence manager but also to the EJB container and the Relational Resource Adapter:

- Persistence manager makes decisions about isolation level, cursor management, and so on.
- EJB container influences the management of EJB collections.
- Relational Resource Adapter (RRA) provides prefetch hints in defined increments to control the number of rows read from database at a time. WebSphere does not provide paging nor does the RRA provide prefetch. Instead, a prefetch hint is passed to the database telling how many rows are going to be read so that the database can optimize the access.

Profiling levels

Profiling can be configured at different levels, depending on your needs. Table 8-1 shows the different Profiling configurations available as they are seen in the Application Assembly Tool and their granularity level.

Table 8-1 Profiling configurations

PME	Scope
Dynamic Query	Entity
Access Intent	Method
Application Profiling	Unit of Work

Important: WebSphere V5.0.0 and V5.0.1 define Dynamic Query intent at bean level. This intent is used whenever the Dynamic Query service is invoked. In WebSphere V5.0.2 this function disappears and is substituted by default Access Intent. This intent is used when no other intent can be applied even if the bean load is done through the Dynamic Query service. This book refers exclusively to WebSphere V5.

You can set different Profiling configurations on the same EJB, so apparently there will be collisions. But a clear priority hierarchy has been defined upon these three levels of Profiling. Notice that there are three types of Profiling, each with a different level of granularity.

Access Intent

Access Intent lets you associate an Entity CMP EJB method with an Access Intent Policy. An Access Intent Policy is a set of properties defining how the EJB container should access the persistence layer.

This feature is provided in WebSphere Application Server V5 as well as in the Enterprise. It is a step forward in performance tuning but yet limited.

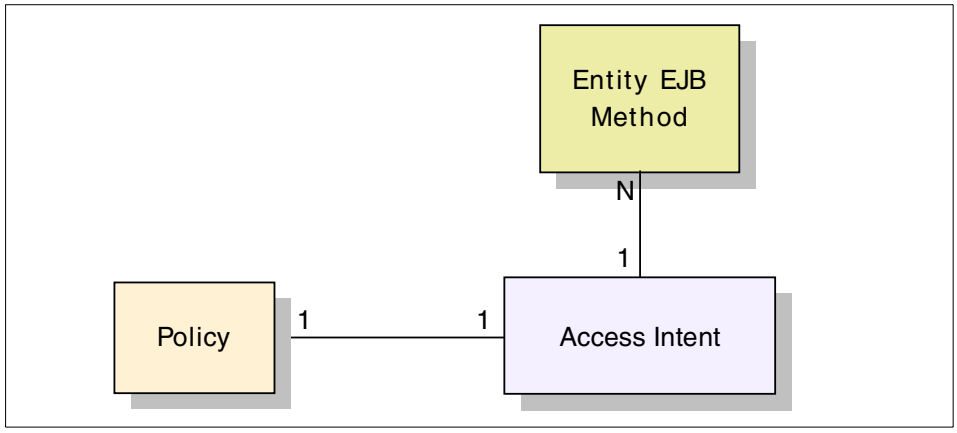


Figure 8-1 Access Intent diagram

Because Access Intents are defined at method level, you can assign different Access Intent Policies to EJB's create, remove, setter and getter methods, achieving some performance tuning. If you are unable to decide which Access Intent to apply in deciding upon the caller or client of the Entity EJB, this issue is covered by Application Profiling.

Application Profiling

Application Profiling lets you associate task names to Session and Entity EJB methods and group a set of tasks under a set of Access Intents. This provides the capability of associating at runtime an Entity EJB method with a specific Access Intent depending on the task under which it is called. So the binding is caller dependent and resolved dynamically at runtime.

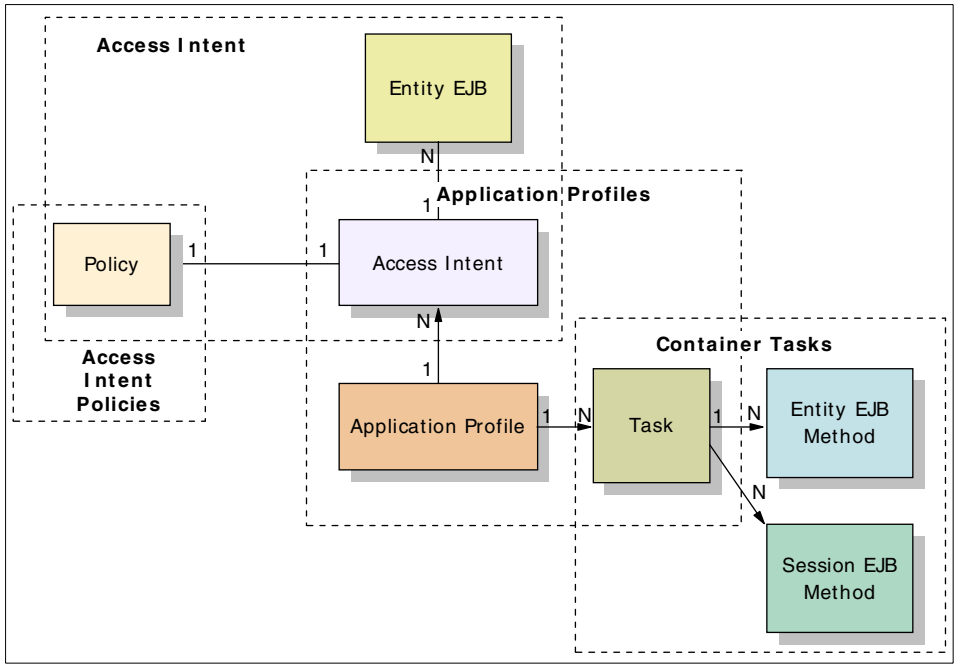


Figure 8-2 Application Profiling diagram

A task is associated with an Application Profile at deploy time. At runtime, the task name gets propagated and its arrival identifies the caller. An Application Profile associated with that caller task defines the Access Intents to be applied.

Default Access Intent

There is another level of configuration for Access Intent. You can set an Entity EJB to be accessed with a specific Access Intent when it is loaded as a result of a Dynamic Query. In fact, this configuration is used by default if no other Access Intent is specified. This Access Intent is defined at bean level.

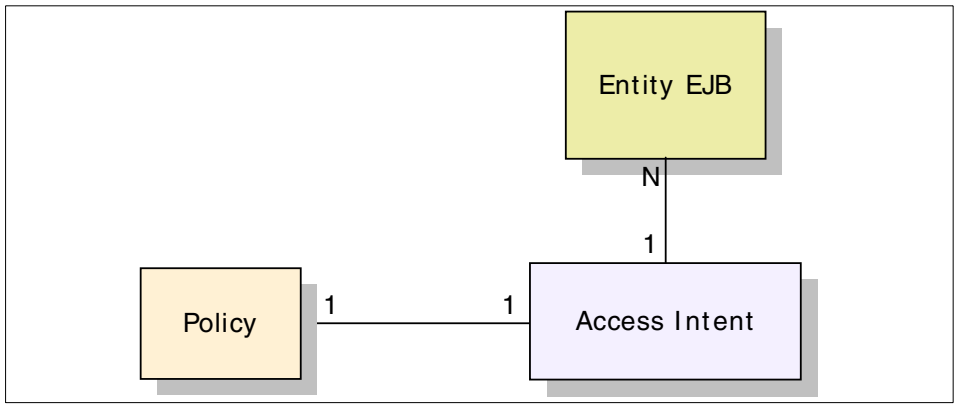


Figure 8-3 Default Access Intent, Dynamic Query Access Intent diagram

Note: Access Intents within an Application Profile and default Access Intents are defined at bean level. Access Intents are defined at method level. Compare the Access Intent diagrams in Figure 8-1 on page 345, Figure 8-2 on page 346, and Figure 8-3.

Profiling prioritization

When different Access Intents are defined on the same entities, prioritization is applied to resolve which Access Intent to apply. Figure 8-4 on page 348 shows the decision algorithm used to apply an Access Intent when many Access Intents have been defined or are applicable for the current request.

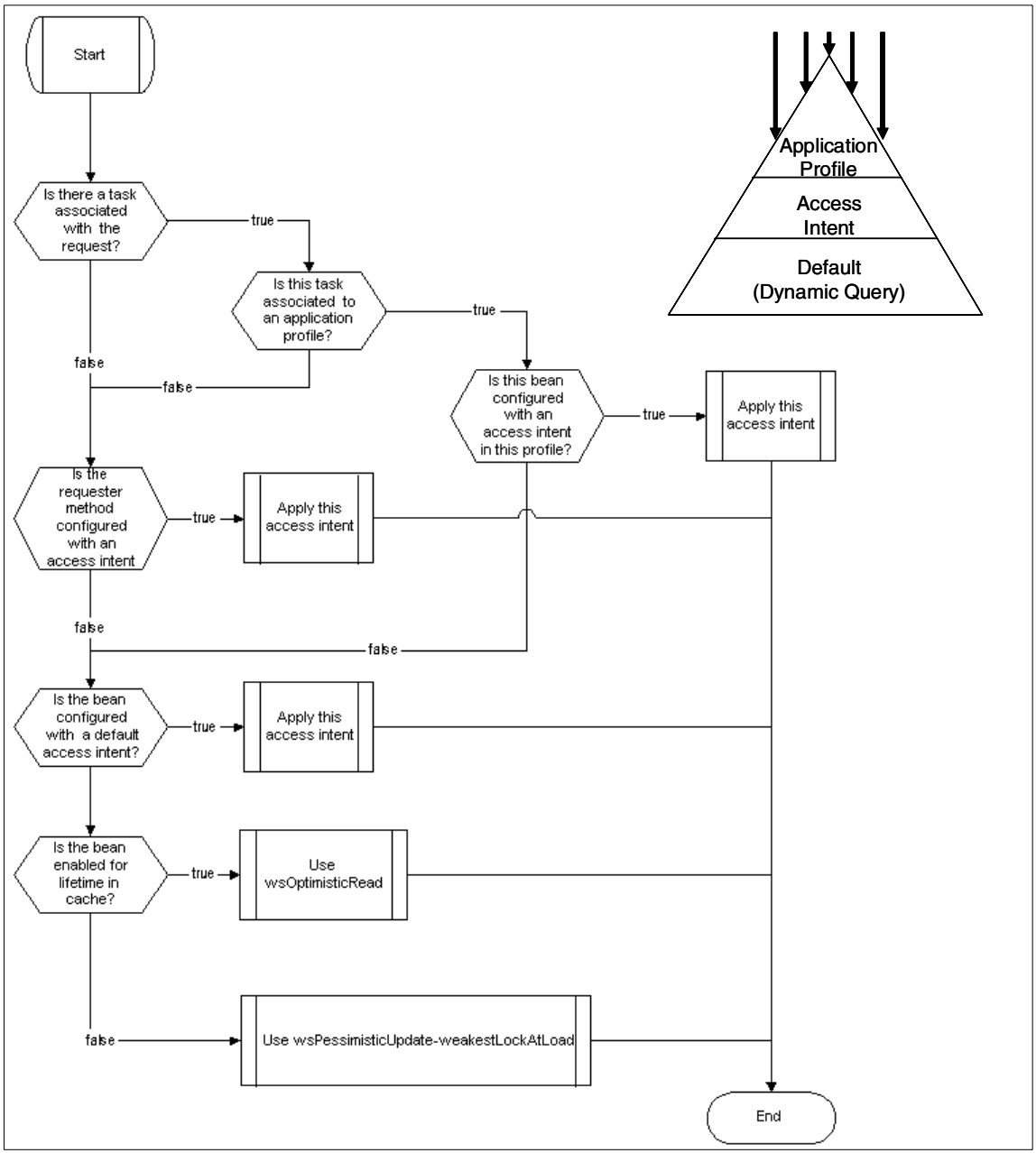


Figure 8-4 Application Profiling prioritization

8.2 Planning

Application Profiling and Access Intent (from now on called Application Profiling or just Profiling) are meant to be configured at deployment time through the extended Deployment Descriptors. For flexibility, an API is exposed for Entity BMP EJBs. This section shows you how to configure Application Profiling and how to work with the exposed API.

The steps to be taken to prepare an application for Profiling are:

1. Create a custom Access Intent Policy to fit your needs
2. Create a Container Task
   a. Associate the Task with EJB methods
3. Create an Application Profile
   a. Associate the Profile with a set of Tasks
   b. Create a set of Access Intents
      i. Associate Entity EJB methods with a custom or predefined Access Intent Policy

An association graph between client methods, Tasks, Profiles, Access Intents and Entity EJBs methods is shown in Figure 8-2 on page 346.

Access Intent Policies

Access Intent Policies are defined by four attributes that serve as an access hint for the persistence manager.

Access Type

Access Type is configured upon two parameters that define the type of access on the bean and the concurrency control applied.

- Concurrency Control
  - Pessimistic

    A lock is acquired on the instance. No one can modify or delete the instance before the end of the transaction. This ensures integrity of read data at the expense of concurrency.
  - Optimistic

    Lock is not acquired until the commit. If the instance is changed during transaction, an exception will be thrown on commit. This generally improves concurrency but an application must be able to handle exceptions.

- Access Type
  - Read

    If pessimistic read intent is specified and an update method is called on a CMP EJB, the container throws an com.ibm.ws.ejbpersistence.utilpm.UpdateCannotProceedWithIntegrityException.
  - Update

    Pessimistic update has additional qualifiers to configure:
    - Weakest Lock at Load

      Although update intent is specified, a container acquires read lock first. If needed, it will try to escalate the lock to update. This is the default Access Intent for all CMP EJBs. This allows better concurrency than if an update lock was attempted at the beginning, but it may result in dead locks.
    - Exclusive

      This is the highest level of locking, which results in the most restrictive isolation levels. It is used to prevent phantom reads.
    - No Collision

      No concurrency control. Selects without locks and updates without checks. This option can be used safely only if database tables where EJBs are persisted are not shared with any other software other than WebSphere. Do not use unless mandatory.

Collection Increment

When you call a finder method you get a lazy collection back from the container. The collection is loaded in memory in chunks, so that as you iterate through it the data you need is already in memory. This parameter controls how many elements are loaded in a single operation in the WebSphere Application Server cache.

Collection Scope

As already mentioned, when you call a finder method you get a lazy collection. It will expire at the transaction boundary, as a default. But you can extend this scope by specifying activity session.

- Transaction

  Is the default value. The collection will no longer be usable at the end of the transaction that created it. References will be dropped and objects will be ready for garbage collection.

- ActivitySession

  The collection may be accessed until the end of the Activity Session under which the collection was retrieved, spanning multiple transactions.

Resource manager Prefetch Increment

This parameter suggest how many rows should be retrieved from the underlying relational database in a single operation. If set to zero, the JDBC driver will ignore it. This parameter may be completely ignored by some database implementations that have their own optimization mechanism. It is a hint to the JDBC driver and may or may not be taken into account.

Read Ahead

This parameter involves multiple Entity EJBs. It is only available for Entity EJBs with a Container Managed Relationship (CMR) defined on them. Possible values are the names of any CMR field on the current Entity EJB. The container uses the read-ahead parameter to prefetch the data of the related entities. This only takes effect when the loading of the entities takes place as a result of invoking a findByPrimaryKey() method.

WebSphere Enterprise comes with several predefined Access Intent Policies, as noted in Table 8-2.

Table 8-2 Predefined Access Intent Policies

Access Intent Policy	Transaction Isolation	Description
wsPessimisticRead	read committed	Read locks are held for the duration of the transaction. Updates are not permitted.
wsPessimisticUpdate	repeatable read	Gets update lock at the beginning of the transaction. Prevents dead locks.
wsPessimisticUpdate-Exclusive	serializable	Read or update locks are held for the duration of the transaction on the entire range of data affected by the SQL statement. Misuse may result in dead locks.

Access Intent Policy	Transaction Isolation	Description
wsPessimisticUpdate-noCollision	read committed	No locks held but updates permitted. Provides no concurrency control. Can lead to data corruption if misused.
wsPessimisticUpdate-weakestLockAtLoad	repeatable read	Initial read lock held. Locks escalated at storage time if updates are made. This is the default policy.
wsOptimisticRead	read committed	No locks are used. Updates are not allowed.
wsOptimisticUpdate	read committed	No locks are held. Updates are allowed. If data originally queried has changed since the read, the update will produce an exception and will not take place.

Note: Concurrency control and access type parameters have not been included, since they are part of the Access Intent Policy name and therefore are obvious. All collection scopes are set to `Transaction`. No prefetch increment or read-ahead is defined. All pessimistic updates have a collection increment value of 1, except wsPessimisticUpdate-NoCollision, which has a value of 25. The rest have a value of 25 (A value of 1 is used with SELECT FOR UPDATE and a value of 25 for SELECT).

Important: The transaction isolation levels are only true for the following JDBC drivers: DB2, Sybase, Informix®, Cloudscape, and SQLServer.

A read committed transaction isolation level ensures only committed data is read. No locks are maintained for the duration of the transaction. This implies that another process may change and commit the data you are reading, leading to a non-repeatable read. Also rows matching your criteria may be inserted or

deleted, causing a phantom read. In database terminology, this isolation level is called cursor stability.

The repeatable read transaction isolation level works like the read committed but it also ensures repeatable reads. Another process will not be able to update a row you obtained until your transaction is complete. However, phantom reads are still possible. In database terminology, this isolation level is called read stability.

A serializable transaction isolation level ensures repeatable reads and disallows phantom reads by preventing other processes from inserting or deleting rows that would modify your result set until your transaction has completed. In database terminology, this isolation level is called repeatable read.

Optimistic concurrency always implies that dead locks are possible, but are less likely. Under pessimistic concurrency control, transactions started with read intent are not capable of supporting an update.

Important: Do not confuse the *lock escalation* concept in an Access Intent context with *lock escalation* in a database context. In a database context, escalation of locks refers to the internal mechanism of the database that reduces the number of locks. In a single table, locks may be escalated to a table lock from many row locks. In an Access Intent context, *lock escalation* is a synonym of *lock conversion* in a database context. It happens when a more restrictive lock than the one held is needed. A read lock on an object may be converted to an update lock.

Figure 8-5 shows a better approach to decide which Access Intent should be chosen, taking into consideration the intention of the transaction and the isolation level that the transaction requires.

Access Type / Transaction Isolation Level	read	update
read commited	wsOptimisticRead	wsPessimisticUpdate-NoCollision
	wsPessimisticRead	wsOptimisticUpdate
repeatable read	**wsPessimisticUpdate**	**wsPessimisticUpdate-weakestLockAtLoad**

Identifies access intents that hold locks.
All locks are help for the duration of the transaction.

Figure 8-5 Access Intent decision table

Tip: Notice that the provided Access Intent Policies cover all the possible combinations between pessimistic and optimistic concurrency control and read and update access types. When defining an Access Intent you can override all the other configurable attributes of the Access Intent Policy of your choice. There is no need to create custom policies but for Dynamic Query Access Intents. Dynamic Query Access Intents do not let you override Access Intent attributes.

8.3 Performance report

Note: The performance data shown here is part of *IBM WebSphere Application Server Enterprise, Version 5 Performance Report*, an IBM internal document.

To demonstrate the concepts and effectiveness of Application Profiling, Figure 8-6 on page 355 through Figure 8-10 on page 360 show several varying configurations and client mixes. Note that the application design and configuration is intended only to show the effect of Application Profiling on specific concurrency problems. Typical customer applications will generally not have the same concurrency issues as those shown here, and these results cannot be extrapolated onto any other application or data access pattern.

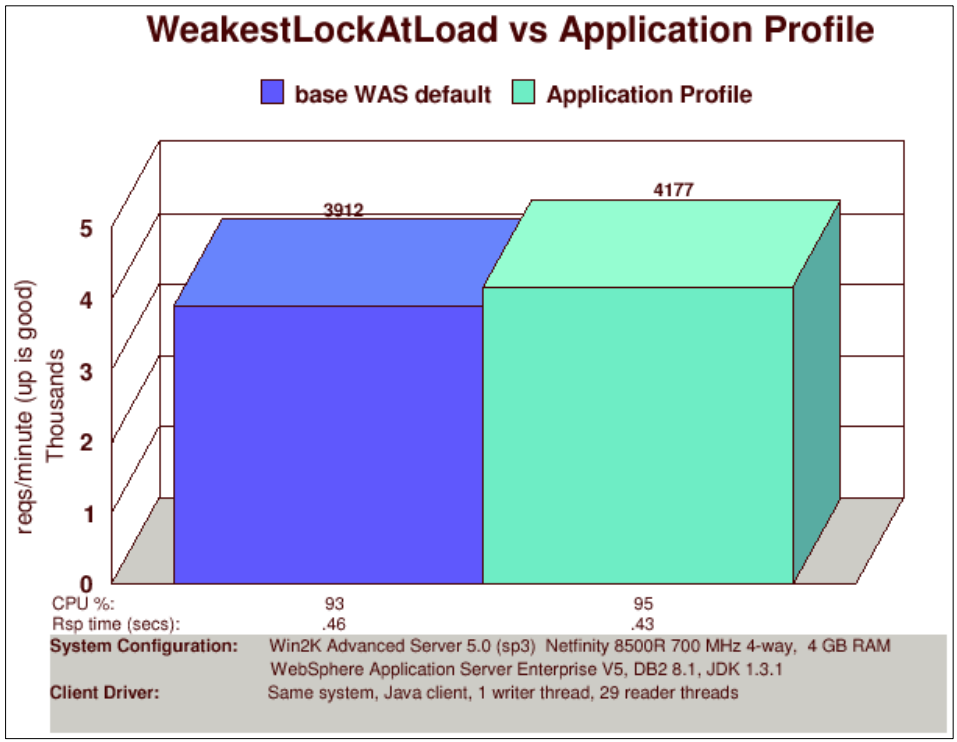


Figure 8-6 WeakestLockAtLoad versus Application Profile

In the example shown in Figure 8-6, read threads and update threads invoke findByPrimaryKey, then retrieve and iterate through a CMR collection of 50 beans. Read threads get a CMR field from each bean, and update threads store new values. For WebSphere Application Server, all bean accesses default to wsPessimisticUpdate-WeakestLockAtLoad. For Application Profiling, read threads use OptimisticRead, and update threads use PessimisticUpdate.

According to the chart, the difference can be small between the default wsPessimisticUpdate-WeakestLockAtLoad in WebSphere Application Server policy and Application Profiling defaults in WebSphere Application Server Enterprise. This is due mainly to the locks required by the workload and minimal contention. No selects are for update, and locks are escalated to write locks by the database on an as-needed basis. The default wsPessimisticUpdate-WeakestLockAtLoad Access Intent Policy is appropriate for certain data access patterns, and for this scenario Application Profiling provides minimal benefit.

Note: The default wsPessimisticUpdate-WeakestLockAtLoad Access Intent can pose problems for more than one update thread due to deadlocks and rollbacks. For more than one update thread, a different Access Intent such as PessimisticUpdate should be used, and Application Profiling can improve performance for that case as seen in Figure 8-7 and Figure 8-8.

While Figure 8-6 on page 355 shows little difference between the default Access Intent Policy wsPessimisticUpdate-WeakestLockAtLoad and a configuration with default Application Profiling, an environment where stronger locking is necessary shows how Application Profiling improves performance by optimizing the Access Intent to the client request. The remaining charts show how a stronger locking requirement poses more of a concurrency problem, and how Application Profiling can be used to overcome the locking contention by associating specific client requests with a particular Access Intent.

Figure 8-7 shows two examples of a workload where a single client does updates and multiple clients do reads. The update request must use PessimisticUpdate, rather than the default wsPessimisticUpdate-WeakestLockAtLoad shown in Figure 8-6 on page 355. All clients access the data using a findByPrimaryKey.

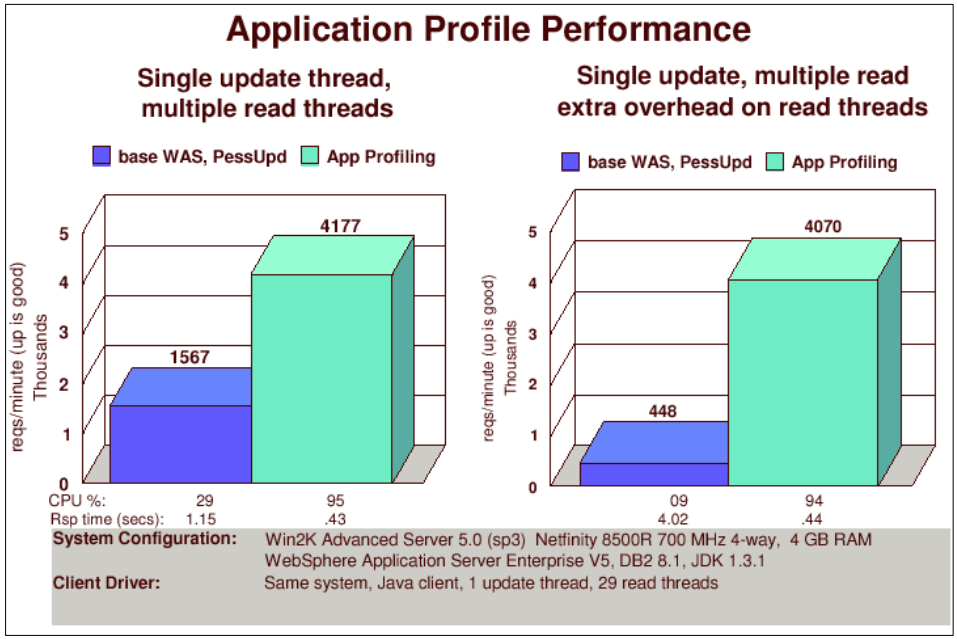


Figure 8-7 Pessimistic Update for all accesses versus Application Profiling

In the scenario used for Figure 8-7 on page 356 and Figure 8-8 on page 358, read threads and update threads do findByPrimaryKey, then retrieve and iterate through a CMR collection of 50 beans. Read threads get a CMR field from each bean, and update threads store new values. For the WebSphere Application Server V5 example, the beans are loaded using the PessimisticUpdate policy because only one Access Intent Policy can be used when the beans are loaded. For WebSphere Application Serve Enterprise V5 Application Profiling, read threads load beans using OptimisticRead, and update threads load beans using PessimisticUpdate. After the bean is loaded, all data accesses from all method invocations use the policy in effect when the bean was loaded.

Figure 8-7 on page 356 shows the performance differences obtained when Application Profiling tunes the application server to the workload. In the case of WebSphere Application Server V5, the read thread and the update threads both load the beans using the same PessimisticUpdate Access Intent Policy. Therefore, the update thread and the read threads all carry the same overhead when accessing the data.

For WebSphere Enterprise Application Profiling, transactions that load beans for read only can use a different Access Intent than transactions that load beans for update. In this case, the update thread continues to use PessimisticUpdate on data accesses after the beans are loaded, but the read threads use OptimisticRead, which eliminates the read thread locks that cause contention with all other threads. The lower CPU utilization indicates a bottleneck caused by database locking.

Database lock monitor tools (such as DB2 Snapshot) can be used to detect and verify locking behavior. Figure 8-8 on page 358 shows how additional business logic on threads increases concurrency issues (due to increased lock hold time), and again how Application Profiling can be used to overcome the locking contention by associating client requests with the optimal Access Intent. The chart shows an even larger delta in the PessimisticUpdate scenario when we add additional processing to the read thread (such as doing file I/O to build a report) while the locks are held. A comparison between this and the previous chart shows the Application Profiling scenario can withstand the extra lock time held on the read threads and maintain CPU utilization rather than serializing on the database.

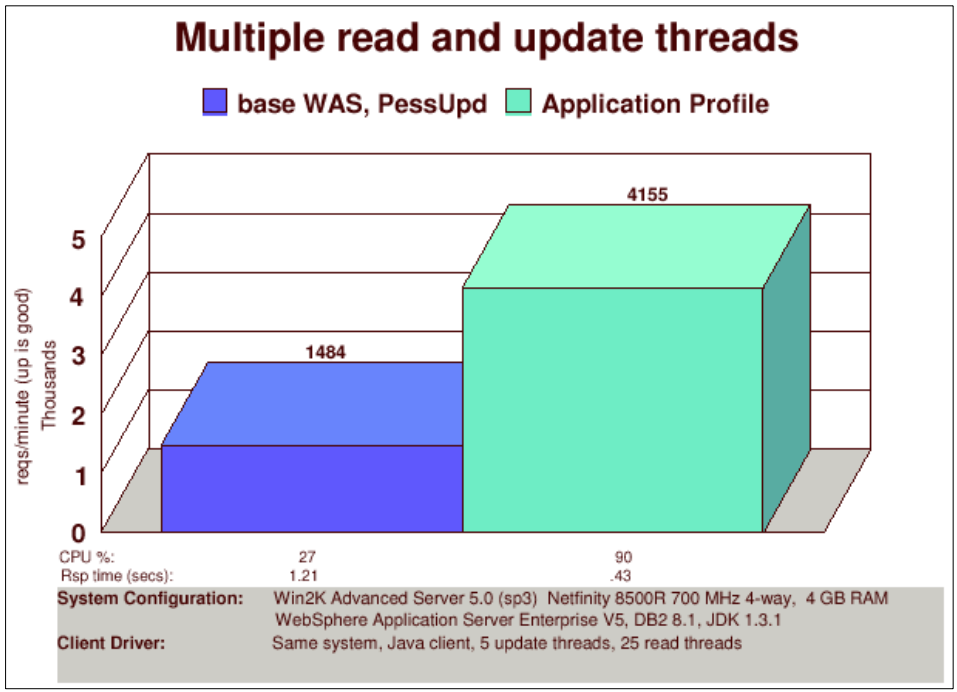


Figure 8-8 Pessimistic Update for all Accesses versus Application Profiling

Figure 8-8 shows more data similar to Figure 8-7 on page 356, but the scenario includes an increase in update threads.

Again, for WebSphere Application Server V5, all accesses to data using findByPrimaryKey use the PessimisticUpdate Access Intent because that is the Access Intent Policy used for loading the entity beans. Since incoming threads can be either for read or for update, all threads must use that update Access Intent.

Application Profiling enables multiple Access Intents per thread, and the Access Intents can be associated with and therefore optimized to the incoming request (read Access Intent for read requests, update Access Intent for update requests).

In summary, Application Profiling can make a significant performance improvement for specific data access patterns. In the cases shown here, when some incoming requests require stronger locking than others, Application Profiling can optimize the data access pattern to specific requests. Note that the scenario used for these charts is for demonstration purposes only, and that real-world scenarios with different data access patterns and locking requirements will likely see less performance benefit than that shown here.

Collection Increment Access Intent

Figure 8-9 shows the usage of Application Profiling to tune the collection increment. Collection Increment determines how many beans are activated with data from the persistence manager cache, for a collection returned by a finder method. Read threads obtain and iterate through a collection. For the WebSphere Application Server V5 example, all threads use the same collection increment because only one Access Intent Policy can be configured on any one load point (the method that drove the bean to load), while Application Profiling enables different collection increment values to be tuned for different client requests.

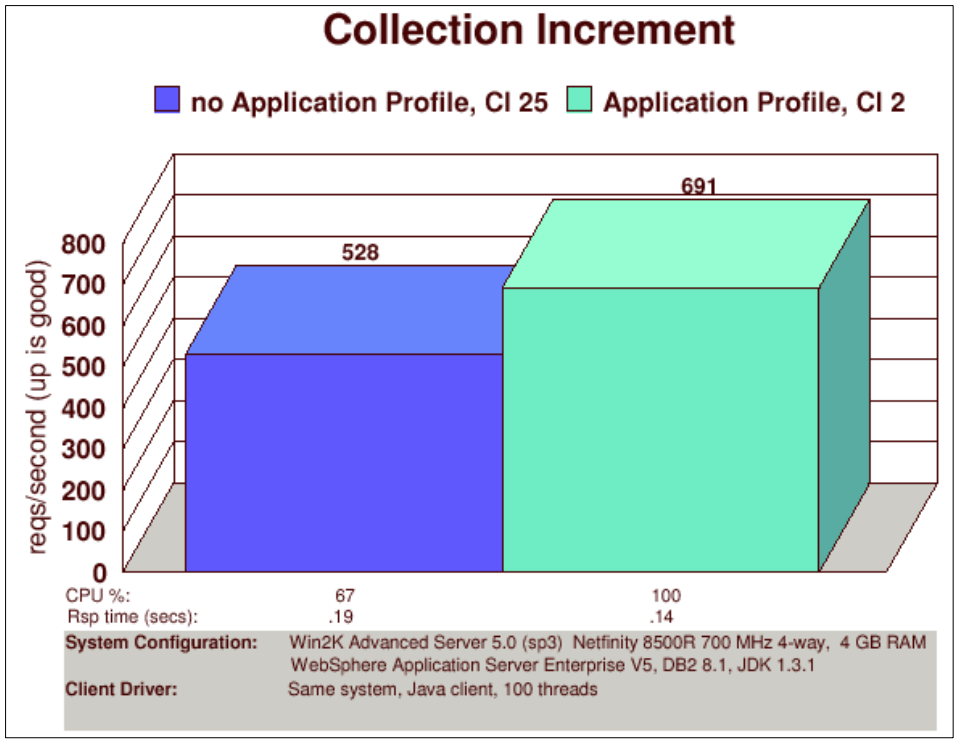


Figure 8-9 Application Profiling using Collection Increment

For Figure 8-9, the result shows a workload where a set of clients benefit from a reduced collection increment value because the search result is satisfied earlier in the iteration, and not as many beans must be activated with data from the persistence manager cache.

In this example, a session bean method invokes the remote interface on a custom finder method, then iterates through the returned result set collection. Under the covers, the application server EJB container activates and returns the

initial number (collection increment value) of EJBs. For the rest of the result set, the container gets the keys only and does not activate the beans. When the iterator in the session bean requires the next group of EJBs in the result set, the next group (again, collection increment value) of EJBs are activated and returned.

In the example in Figure 8-9 on page 359, the session bean logic generally finds the EJB it requires earlier in the result set and jumps out of the iteration operation. Setting the collection increment to a lower value avoids the additional overhead of activating additional beans.

Read ahead Access Intent

Read ahead is an Access Intent hint that specifies an association between CMP entity beans. Read ahead allows applications to minimize the number of database round trips by retrieving a transaction's working set of EJBs within one query. Read ahead caches the data for related objects, which ensures the data is present for the activation of the EJBs that are most likely needed next by the application.

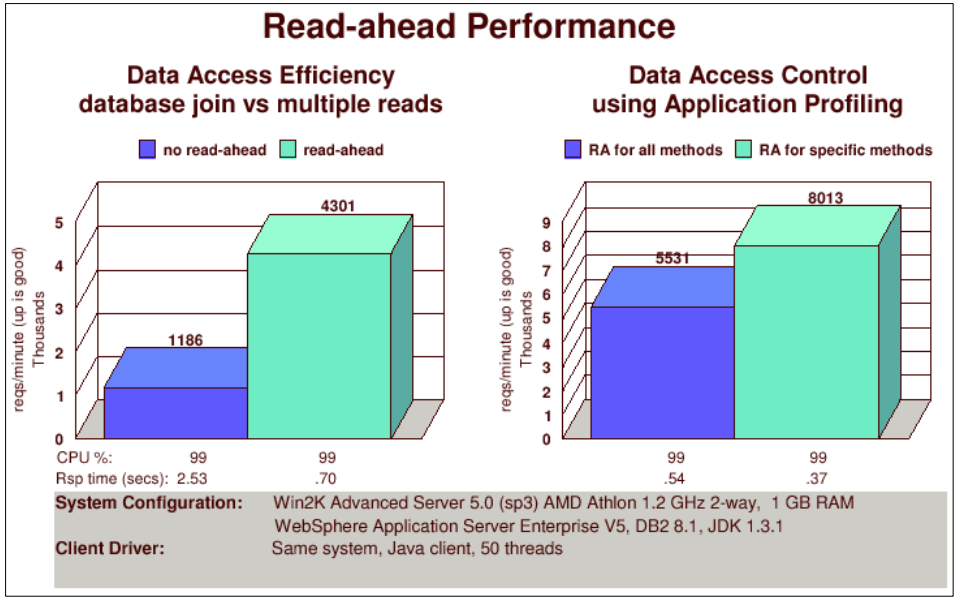


Figure 8-10 Read ahead performance

In Figure 8-10, the chart on the left shows how performance can be improved by reducing trips to the database. For this example, client threads make requests to a session bean that does a findByPrimaryKey on a CMP entity bean A. The session bean then uses a CMR field in bean A to get a collection from CMP bean

B. While the collection is being iterated, a third database access occurs while fields are retrieved from CMP bean C.

The comparison shows the difference in throughput when 50 client threads make requests to an application with no read ahead configured, versus with read ahead configured. The read ahead relationship is configured in bean A, and specifies that data should be read for the second and third beans B and C. When the findByPrimaryKey is invoked on bean A, a more complex query is issued that includes a database join, which caches the data for beans B and C. The chart on the left shows the difference for multiple trips to the database, versus a single trip to the database but with a more complex join query. More trips to the database adversely affects throughput, while the more efficient read ahead produces better throughput.

Read ahead can also have a negative impact on throughput, if the read ahead hint is not applied correctly. If read ahead causes a load of data that not all clients need, inefficiencies can occur.

The chart on the right side of Figure 8-10 on page 360 shows the difference Application Profiling makes when multiple Access Intents are specified for bean A. In this example, clients invoke two methods on a session bean. Both session bean methods invoke the same findByPrimaryKey on entity bean A. One session bean method requires data from three entity beans in the same transaction and can benefit from read ahead (the same A, B, and C transaction as in the previous example).

The other session bean method requires data from only the first entity bean; reading data from the second and third beans is not necessary. The client driver runs a mix of 50 request threads, with 25 threads accessing the session bean method that does the read-ahead, and the other 25 threads accessing the method that requires only one field from the first bean.

The chart shows the difference when both findByPrimaryKey invocations use the same Access Intent (Read Ahead) versus the Application Profiling capability to assign different Access Intents to the same bean. For WebSphere Application Server Enterprise V5, the method that needs only a single bean saves on database accesses. If read ahead is configured for bean A, both session bean operations get all the data on the findByPrimaryKey method even though one of them does not require data from the second and third beans. Read ahead is not always an optimization, but Application Profiling enables the customization to use read ahead only where it can best be used.

8.4 Assembly

The sample scenario for this chapter differs from the global scenario used in this book to fit the purpose of illustrating different configuration possibilities that could not be addressed in the global scenario.

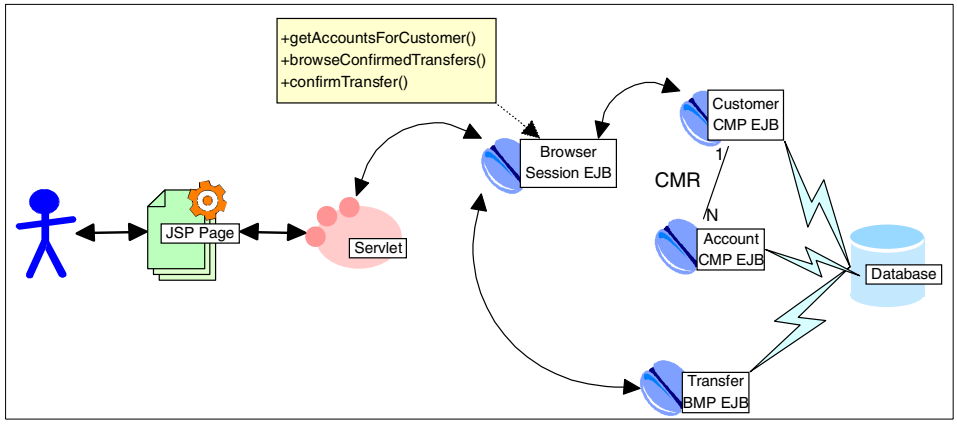


Figure 8-11 Sample scenario for Application Profiling

The sample scenario shown in Figure 8-11 represents a simple e-banking application. The application has Customer and Account CMP Entity EJBs with a relation of one Customer to many Accounts. It also has two Session EJBs:

- Customer Administrator: For administration purposes, such as Customer creation, modification and deletion.
- Account Browser: Intended to be used by customers connecting online to browse their accounts.

A user can browse his accounts, transfer money between accounts, and view all the transfers associated with a specific account. These business methods are implemented in the Browser session EJB. The applications works with three Entity EJBs: a Customer CMP, an Account CMP and a Transfer BMP. A one-to-many Container Managed Relationship (CMR) has been established between Customer and Account.

Until now, configuration issues for persistence of the Entity CMP EJBs resided solely on the Entity CMP EJB. So the EJB container had to access the persistence layer always in the most restricted way of all possible accesses defined for an Entity CMP EJB.

The most restricted access for Customer may be defined for creation, update, or deletion. Therefore, the Customer EJB, even if it was going to be accessed for

browsing purposes by the AccountBrowser, would access the persistence layer performing a lock as if it were updated by the CustomerAdministrator.

The AccountBrowser best approach should be accessing the Customer in an optimistic way with no prefetch and with read-ahead. We are not performing any data modification, so optimistic access is best. Only one Customer is being retrieved, so there is no need for prefetch. But we want to browse all of the Customer's related Accounts, so a read ahead specified over the relationship would fit best.

On the other hand, the CustomerAdministrator best approach should be accessing the Customer EJB in a pessimistic way with no prefetch and no read ahead.

8.4.1 Creating an Access Intent Policy

We will use the sample scenario application illustrated in Figure 8-11 on page 362 to show you how to use the Application Assembly Tool to create a custom Access Intent Policy.

The browseConfirmedTransfers business method retrieves a collection of Transfer objects related to a given account number. Associated transfers are expected to be an average of 10 per month.

You will create an Access Intent Policy with an optimistic concurrency control. The transaction isolation level is read committed, so no uncommitted transfers will be read. Data integrity will not be compromised if during the execution of browseConfirmedTransfers method transfers are deleted or created. You can take advantage of Profiling setting the resource manager prefetch increment to 10, as this is the average number of transfers to be returned by the method.

1. Launch the WebSphere Application Assembly Tool.

   Once the Application Assembly Tool starts, the window shown in Figure 8-12 on page 364 opens. Click the **Browse** button and select the sample scenario application file named eBankEAR.ear. Click **OK** when done.

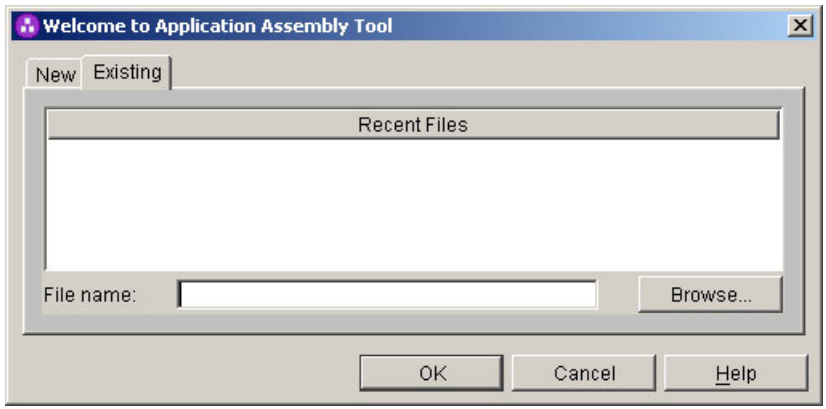


Figure 8-12 Application Assembly Tool: Welcome window

2. The Application Assembly Tool loads the enterprise application archive and shows a tree on the left pane with all the configurable options. Select **EJB Modules -> eBankEJB**. The tree in Figure 8-13 on page 365 shows options related to Application Profiling highlighted with a red square. Refer also to Figure 8-2 on page 346 to view the relationship between the entities these options configure.

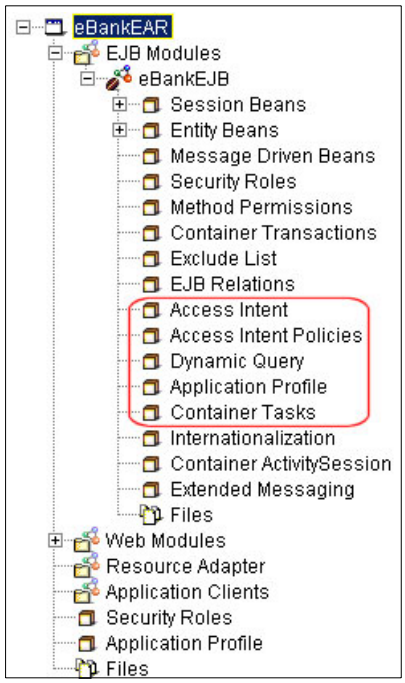


Figure 8-13 Tree of configurable options

3. On the left pane, select the **Access Intent Policies** node. Select **File -> New -> Selected Object** from the menu to create a new Access Intent Policy. The New Access Intent Policy window will open, as seen in Figure 8-14 on page 366.

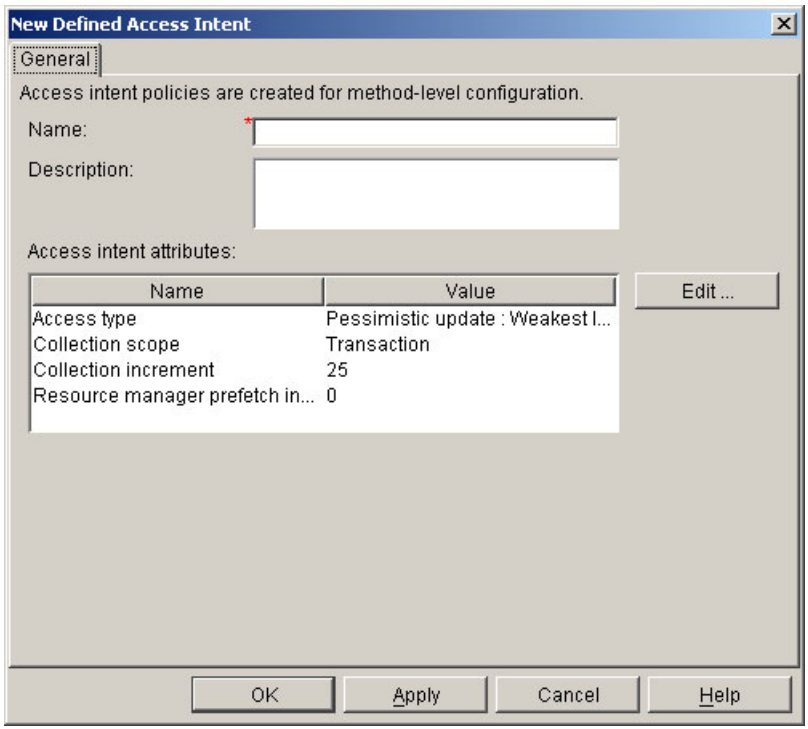


Figure 8-14 Application Assembly Tool: New Access Intent Policy window

4. Give the Access Intent Policy the following name: eBankFindTransferByAccountPolicy.
5. Click the **Edit** button. A list with the four configurable attributes appear. Select **Access type**. All the possible access types are present; choose **Optimistic Read**.
6. Click the **Edit** button again. Select **Resource manager prefetch**. Set the value to 10.

8.4.2 Creating an Access Intent

The next step is to associate the previously created Access Intent Policy with the findByAccount method on the Transfer bean. When this method is invoked on the Transfer bean to retrieve a collection of Transfer objects, the database will prefetch as many as 10 reducing the number of trips to the database.

This task can be performed both from the Application Assembly Tool and from WebSphere Studio IE.

Application Assembly Tool

The following steps apply to the Application Assembly Tool:

1. In the left pane, select the **Access Intent** node. Select **File -> New -> Selected Object**. The New Access Intent window will open, as seen in Figure 8-15.

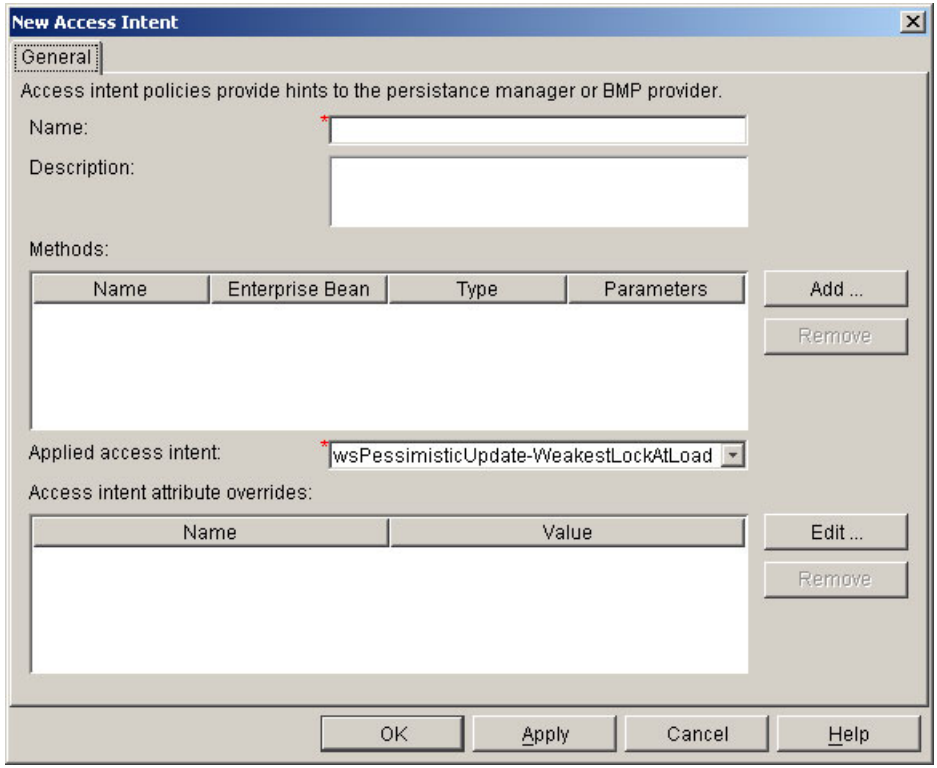


Figure 8-15 Application Assembly Tool: New Access Intent window

2. Give the Access Intent the following name: eBankFindByAccountIntent.
3. Click the **Add** button. The next window shows a tree with all the available entity beans and its methods. You can select a method to associate in the Access Intent, depending of the interface that exposes it. Expand the tree to see the following nodes:
   - eBankEJB.jar
   - Transfer
   - All methods

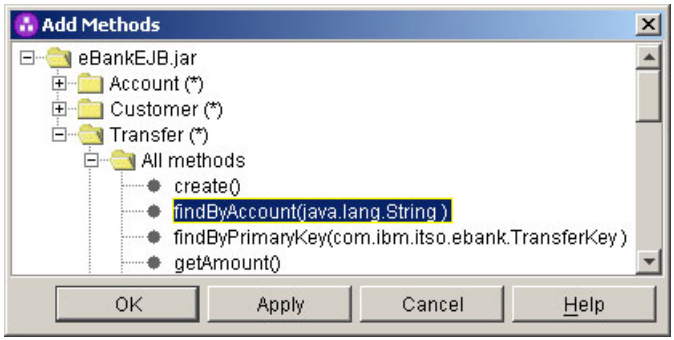


Figure 8-16 Application Assembly Tool: Add methods to an Access Intent

Select the method with the following signature: findByAccount(java.lang.String). Click **OK** to confirm.

4. The Application Assembly Tool shows the window seen in Figure 8-15 on page 367 with the data you just added. From the Applied Access Intent drop-down list, select **eBankFindTransferByAccountPolicy**, the Access Intent Policy we created in previous steps.

> **Note:** Clicking the **Edit** button lets you override Collection Scope, Collection Increment, Resource Manager Prefetch Increment, and Read Ahead Policy attributes. Given that the default policies cover all possible combinations of the remaining attribute Access type, you can use default policies and override them as needed instead of creating new policies.

5. Click **OK** to confirm the creation of the new Access Intent.

WebSphere Studio IE

To perform this operation from WebSphere Studio IE, you need to import the EAR file into the application workbench, as follows:

1. Launch WebSphere Studio IE.
2. Select **Window -> Open Perspective -> J2EE**.
3. From the top left pane, click the **J2EE Hierarchy** tab. A tree with all J2EE components currently loaded in the workbench appears, as seen in Figure 8-17 on page 369.

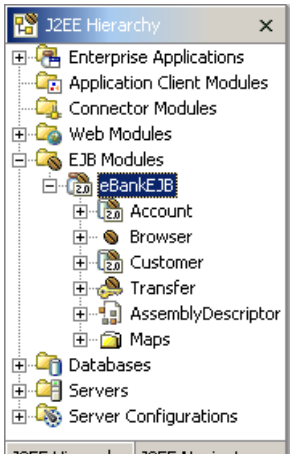


Figure 8-17 J2EE Hierarchy tree

Expand the tree by clicking on the plus sign on the left of the EJB Modules and eBankEJB nodes.

4. Right-click **eBankEJB** and from the pop-up menu, select **Open With -> Deployment Descriptor Editor**. The Deployment Descriptor will open in the top-right pane of the perspective.
5. Switch to the **Access** tab. Scroll down if necessary to view the Access Intent for Entities 2.x option, as seen highlighted in Figure 8-18 on page 370.

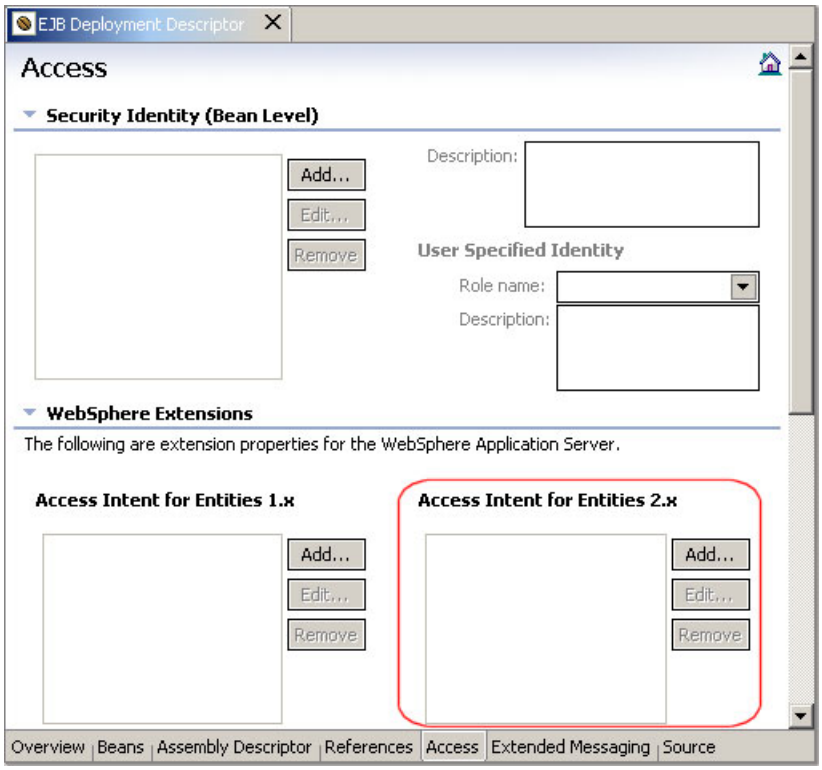


Figure 8-18 Access Intent for Entities 2.x

6. Click the **Add** button next to Access Intent for Entities 2.x. A wizard starts to help you configure your Access Intents.

   a. Give the following name to the new Access Intent: eBankFindByAccountIntent.

   b. Select the appropriate Access Intent Policy: **wsOptimisticRead**.

   > **Note:** You cannot configure Access Intents with custom policies from WebSphere Studio IE, since it only offers WebSphere Application Server capabilities. You cannot override attributes when creating an Access Intent from WebSphere Studio IE.

   Click **Next** to continue.

   c. The next window shows all Entity EJBs in the current EJB module. Check the **Transfer** bean. Click **Next** to continue.

d. The Transfer bean is shown on the next window. Expand it by clicking the plus symbol on the left. All the methods of the Transfer bean are shown categorized by the interface that exposes them. Notice the different symbols and their meanings, as seen in Figure 8-19.

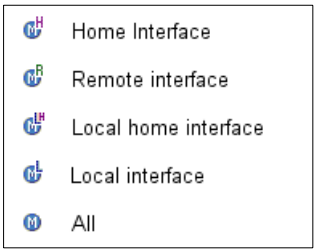


Figure 8-19 Icon legend

e. Scroll down and select all **findByAccount** methods, as seen in Figure 8-20.

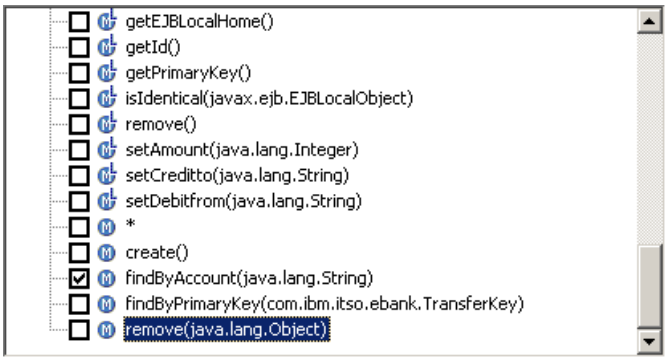


Figure 8-20 Select method

f. Click **Finish** to confirm the creation of the new Access Intent.

8.4.3 Creating an Application Profile

The getAccountsForCustomer business method implemented on the Browser EJB retrieves all the accounts associated with a given customer. Since there is a CMR specified between Customer and Account, we can take advantage of Profiling by reading ahead of time Account objects as soon as we retrieve a Customer. We will configure an optimistic read policy with read ahead specified on the CMR attribute of Customer.

First, you will create a task and associate it with the getAccountsForCustomer method. Then you will create an Application Profile and tie it to the previously

defined task. In the final step, you will set the appropriate Access Intent for the Customer bean in this profile.

The following steps apply to the Application Assembly Tool:

1. We are going to create an Application Profile to benefit from the read ahead hint when retrieving Customer data. So when we get the Customer data, we will also obtain its related Accounts. This technique requires a transaction context.

   Create a container transaction that starts when getAccountDataForCustomer is invoked on the Browser bean. Select **Container Transactions** on the left pane. From the menu, select **File -> New -> Selected Object**.

   a. Give the association the following name: RequiredTransactions.

   b. Click the **Add** button. A tree with all the beans and its methods is shown.

   c. Select the **Browser** bean and under All Methods select **getAccountDataForCustomer(String)**. Click **OK**.

   d. Select a **Transaction Attribute** of `Required`. Click **OK** to confirm the changes.

2. On the left pane, select **Container Tasks**. On the menu, select **File -> New -> Selected Object**. The window seen in Figure 8-21 on page 373 will open.

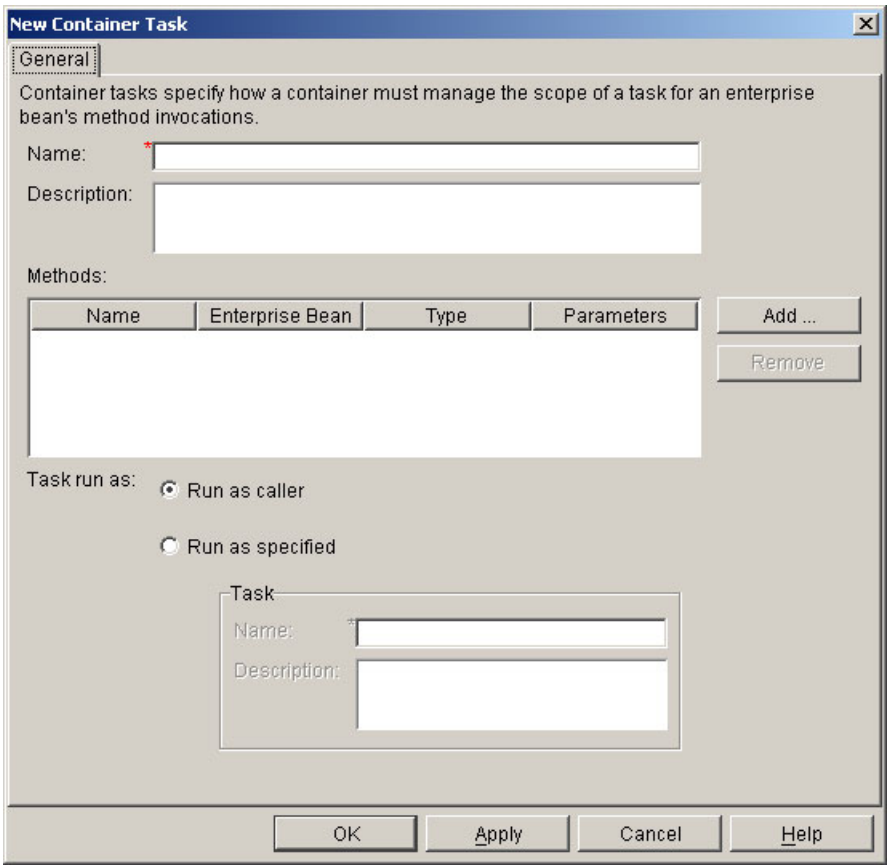


Figure 8-21 Application Assembly Tool: new task window

a. Give the task the following name: eBankGetAccountDataForCustomerRefTask. This will be the name of the association between the Entity EJB methods and the task name.

b. Click the **Add** button. The next window shows a tree with all the available entity and session beans and their corresponding methods categorized by the interface that exposes them. Expand the tree by selecting **eBankEJB.jar -> Browser -> All methods**.

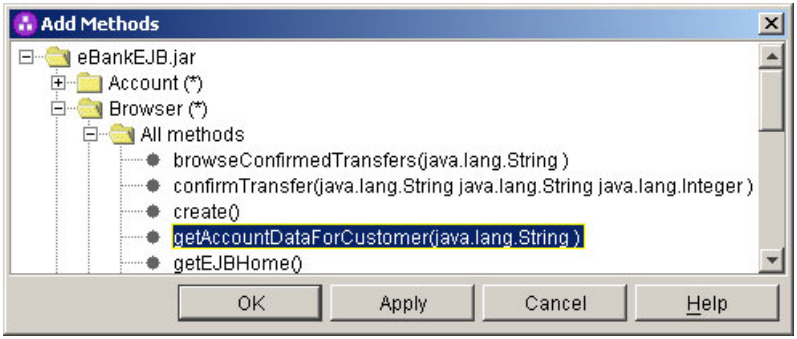


Figure 8-22 Application Assembly Tool: Adding methods in task creation

Select the **getAccountDataForCustomer(java.lang.String)** method. Click **OK** to confirm.

c. Select **Run as specified** and give the task the following name: eBankGetAccountDataForCustomerTask. This is the name of the task under which the Application Profile will be applied. If you select **Run as caller**, the Application Profile service will apply the profile associated to the caller's task.

d. Click **OK** to confirm the creation of the task.

3. On the left pane, select **Application Profile**. From the menu, select **File -> New -> Selected Object**. The window seen in Figure 8-23 on page 375 will open.

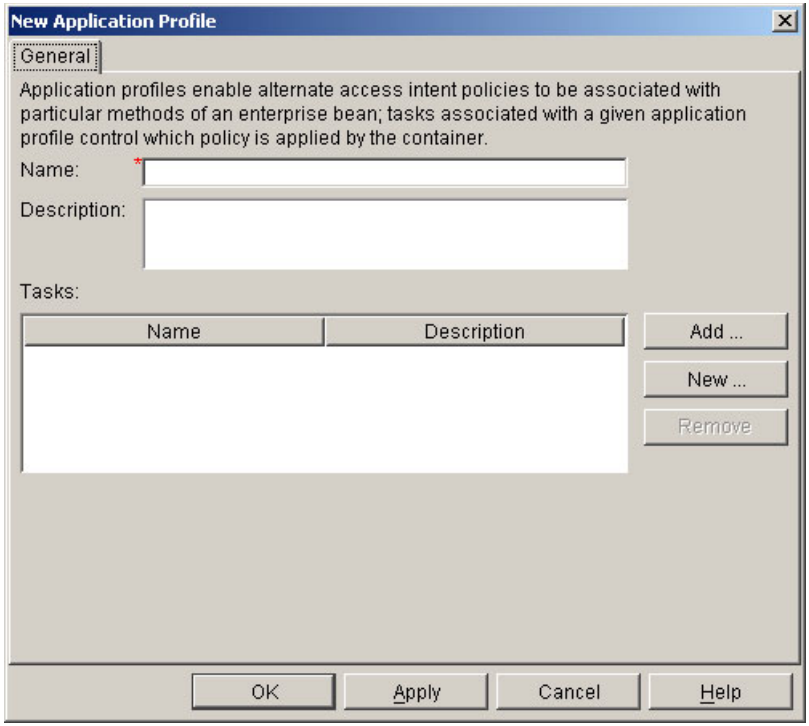


Figure 8-23 Application Assembly Tool: new Application Profile

a. Give the Application Profile the following name: eBankWebUserProfile.

b. Click the **Add** button. From the drop-down list, select the task you created in the previous step: **eBankGetAccountDataForCustomerTask**.

c. Click **OK** to confirm the creation of the new profile.

4. On the left pane, expand the tree by selecting **Application Profile -> eBankWebUserProfile -> Access Intent**. From the menu, select **File -> New -> Selected Object**. The window seen in Figure 8-24 on page 376 will open.

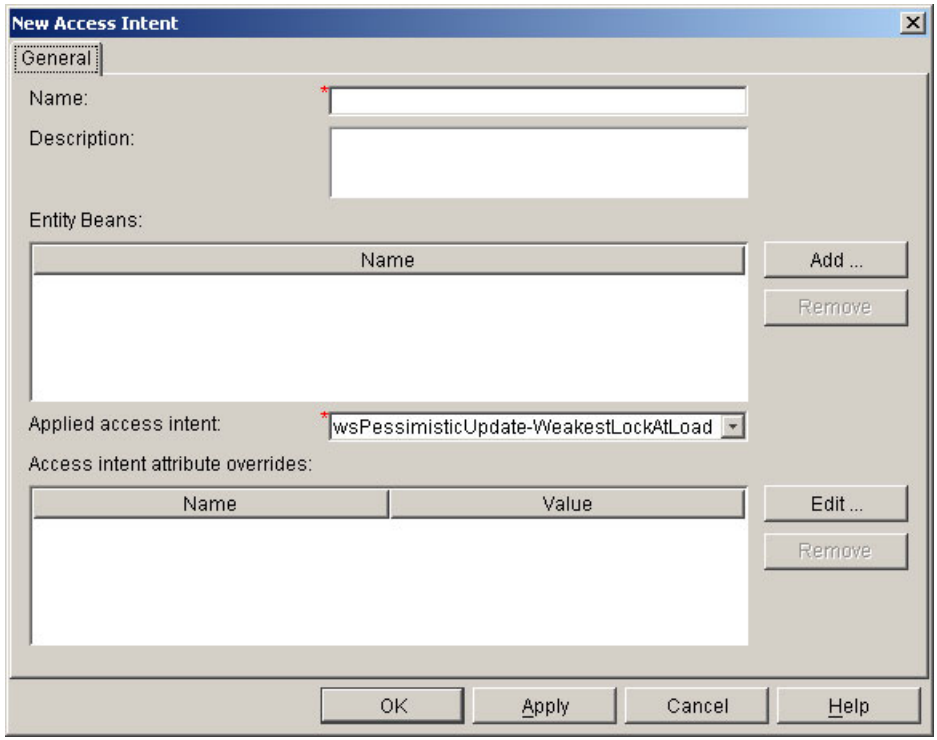


Figure 8-24 Application Assembly Tool: new Access Intent for an Application Profile

a. Give the new Access Intent the following name: eBankGetAccountDataForCustomerIntent.

b. Click the **Add** button. All available entity beans are shown. Select **Customer**. Click the **OK** button to confirm.

> **Note:** Notice the difference from the base Access Intent definition. In the base Access Intent definition, you associate a bean method with an Access Intent Policy. When creating an Access Intent under an Application Profile, you associate the whole bean to an Access Intent Policy. Base Access Intents are defined at method level. Application Profile Access Intents are defined at bean level.

c. From the Applied Access Intent drop-down list, select **wsOptimisticRead**. When browsing Account data for a Customer, we do not require any locks. The Access Intent is only for reading.

d. Click the **Edit** button next to the Access Intent attributes overrides. A menu with overridable attributes pops up. Select the **Read ahead** hint.

Important: When creating a Read ahead hint under an Access Intent (not an Application Profile) ,the Application Assembly Tool opens a window showing the following error:

```
EAAT0046I: The Access Intent attribute must be optimistic in order to
configure a read ahead hint.
```

The Application Assembly Tool will not let you select any CMR field, because it does not recognize the Access Intent you have already selected. On the other hand, when creating a read ahead hint under an Application Profile you are working with an association between two EJBs. The Access Intent chosen is applied on the EJB where the Access Intent is being configured and also on the related EJB. The Application Assembly Tool obliges you to explicitly set an Access Intent for this related EJB under the same Application Profile where the read ahead hint is being configured. The explicit Access Intent and the read ahead hint Access Intent must be of the same nature so that there is no conflict. Since a read ahead hint must be an optimistic read Access Intent, be sure to configure the CMR related EJB with an optimistic read Access Intent.

e. Close the error window.

f. Click **OK** to confirm the creation of the Access Intent without a read ahead hint configured.

g. Explicitly set an **wsOptimisticRead** Access Intent on Account EJB and name it eBankReadAheadCompatibleAccount.

h. At the top of the right pane, select the Access Intent you created on the Customer EJB: **eBankGetAccountDataForCustomerIntent**.

i. At the bottom of the right pane, click the **Edit** button under the Access Intent attributes overrides option. Select the **Read ahead** hint. Now the Application Assembly Tool shows a tree with available CMR fields, as seen in Figure 8-25.

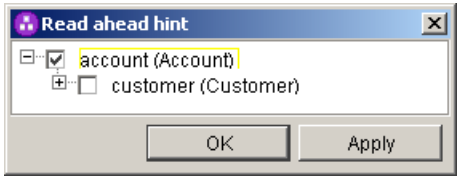


Figure 8-25 Application Assembly Tool: Access Intent with read ahead hint

j. Check **account**.

k. Click **OK** to confirm override the read ahead hint.

l. Click **Apply** to confirm changes in the Access Intent.

Note: Notice that once you assign an Access Intent to an entity bean under a profile, you cannot assign another Access Intent to the same bean under the same profile.

Whenever the Customer bean is accessed from the getAccountDataForCustomer method, this Access Intent will be applied. If we were planning to delete, create, or modify Customer objects through our application, a new profile should be created to accommodate new needs. When modifying or deleting Customer objects, locking would be useful. No read ahead would be needed, since you only need the Customer object, not its related data. We could set different collection increments when modifying just one Customer or applying a modification to a collection of Customer objects.

8.4.4 Creating a Dynamic Query Access Intent

This Access Intents are created under the Dynamic Query option in the Application Assembly Tool. Access Intents created under this option will be applied only if no other Access Intent is assigned to the associated bean and the bean is loaded as a result of a Dynamic Query.

Now you will create a Dynamic Query Access Intent for the Account bean. In order to create safe transfers, we will access the Account objects with a pessimistic concurrency control with read intent but able to escalate to an update. You will use the wsPessimisticRead-WeakestLockAtLoad policy. If the Account object is modified between the read and the update, an exception will be thrown.

1. On the left pane, select **Dynamic Query**. From the menu, select **File -> New -> Selected Object**. The window seen in Figure 8-26 on page 379 will open.

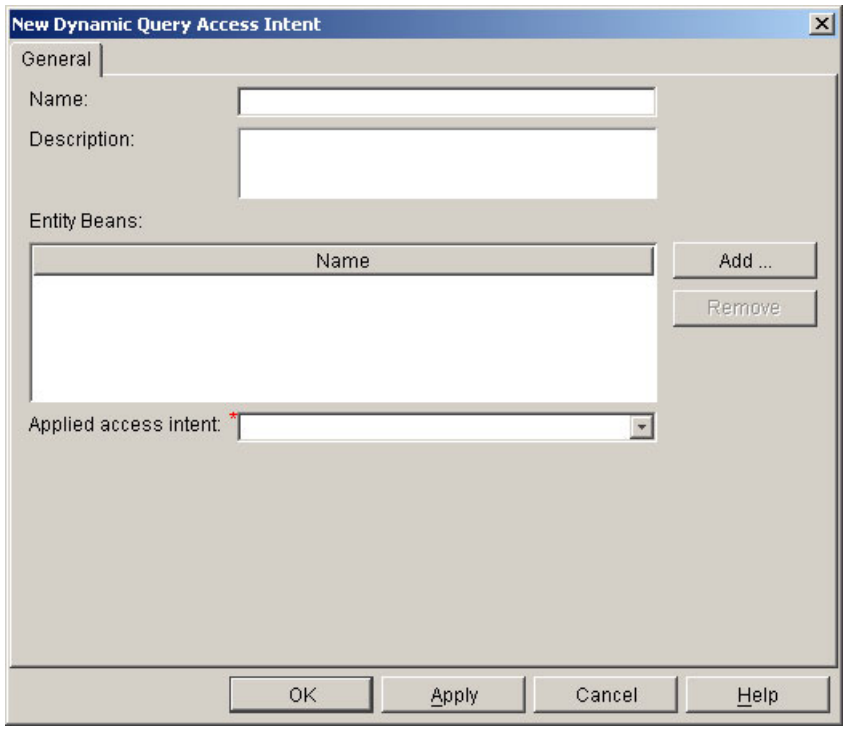


Figure 8-26 Application Assembly Tool: new Dynamic Query Access Intent

Note: Default Access Intent is defined at bean level. Notice that you cannot override default Access Intent Policies. If needed, a Dynamic Query Access Intent justifies the creation of a new Access Intent Policy.

a. Give the Access Intent the following name: eBankModifyAccountIntent.

b. Click the **Add** button. Select the **Account** entity bean.

c. From the Applied Access Intent drop-down list, select **wsPessimisticRead-WeakestLockAtLoan**.

d. Click **OK** to confirm the creation of the default Access Intent.

8.4.5 Application Profiling API

The Application Profiling service exposes an API available to EJBs through a JNDI lookup. This API enables you to programmatically set the task name.

To develop applications using this API, you need to include the appprofile-ee.jar in your project's classpath in WebSphere Studio IE.

1. In WebSphere Studio IE, select **Window -> Open Perspective -> Java**.
2. In the Package Explorer view, located on the left pane, right-click the Java Project or EJB Project where you want to use the Application Profiling service. From the pop-up menu, select **Properties**.
3. On the left pane of the Properties window, select **Java Build Path**.
4. On the right pane of the Properties window, select **Libraries**.
5. Click the **Add** variable button. A window opens showing all available variables. See Figure 8-27.

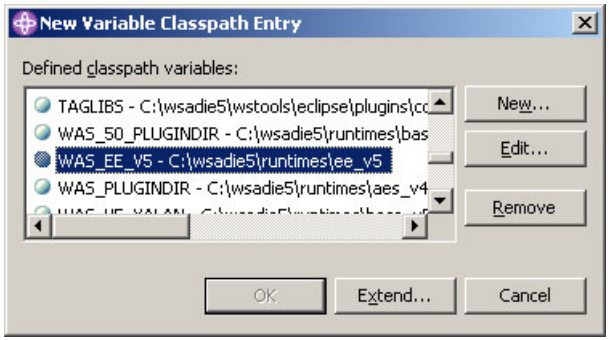


Figure 8-27 New Classpath variable

   a. Selectthe **WAS_EE_V5** variable. Click the **Extend** button to complete the path to the JAR file you need. A window opens and shows a directory tree.
   b. Navigate in the directory tree to the lib folder. In the lib folder, select **appprofile-ee.jar**. Click **OK** to add the new library.
6. Click **OK** to confirm the changes.

WebSphere Studio IE will rebuild your project and the Application Profiling service will be available to your code.

Use the code in Example 8-1 to programmatically set the task names.

Example 8-1 TaskNameManager API

```
InitialContext ic = new InitialContext();
// acquire access to the service
TaskNameManager tnManager =
   ic.lookup("java:comp/websphere/AppProfile/TaskNameManager");
try {
   tnManager.setTaskName("updateAccount");
```

```
} catch (IllegalTaskNameException e) {
   // task name reference not configured. Handle error.
}
/*
* reset the task name to undo any changes and recover whatever
* task name was current when this method was invoked.
*/
tnManager.resetTaskName();
```

Using this API in your enterprise applications will make them not portable to other application servers. Application Profiling is flexible enough and has a declarative approach. An enterprise application configured with Application Profiling is portable to any other application server, although no Access Intent or Application Profile will be applied since they are WebSphere Application Server proprietary features.

Testing the sample scenario

Refer to 8.5, “Problem determination and troubleshooting” on page 381 to enable Relational Resource Adapter (RRA) component traces. RRA component traces will print the queries used by the runtime to access the persistent storage. You can see how the Application Profiling hints are being used by the runtime to optimize the access.

8.5 Problem determination and troubleshooting

Enabling Relational Resource Adapter (RRA) component traces in WebSphere Application Server Enterprise gives you information on how the runtime is using the Application Profiling hints. To enable RRA component traces, follow these steps:

1. Launch the WebSphere Administrative Console, then log in.
2. Select **Servers -> Application Servers**.
3. On the right pane, click **server1**, which is the default instance.
4. From the Additional Properties table, select **Logging and Tracing**.
5. Select **Diagnostic Trace**. Two tabs show on the right pane: Configuration and Runtime. Their contents are the same, but the effect on the server configurations is different. Data stored on the Runtime tab will have effect only until the server instance is stopped. Changes are lost when the server instance is stopped. Data stored under the Configuration tab persists over system shutdowns. Click the **Runtime** tab.
6. In the Trace Specification text area type `RRA=all=enabled`.

7. Click **OK** to confirm the new settings. Since these changes are made in the Runtime tab, no additional step is required. Making changes in the Configuration tab requires you to click **Save**.

You have enabled traces so that you can see how EJBs are accessed. To compare the behavior of Application Profiling versus no Profiling, follow these steps:

1. Deploy the sample application with Application Profiling in your WebSphere Application Server Enterprise V5. The install instructions for the sample application, eBank, can be found in the additional materials available with this redbook (see Appendix C, “Additional material” on page 683).
2. Open your browser and type the following URL: `http://localhost:9080/eBankWeb/`.
3. When you are prompted, enter a valid Customer ID type `01`, then press Enter.
4. A list of accounts for the selected customer is presented. Select account number **001** and click the **Transfer money** button.
5. The selected account will be the account from where money will be transferred.
   a. Under credit, enter account number `003`.
   b. Under amount, type `150`.
   c. Click the **Confirm** button.
6. A summary of the operation is seen. Click **Back** to return to the Customer’s accounts view.
7. Select account number **001**. Click the **Show movements** button.
8. A list of historic related transfers is seen. The first balance value is the original balance of the account. Adding or substracting the value in the amount column gives you the next balance. The last value in bold letters, above the Back button, is the current account balance. Close the browser.
9. Go to <*WebSphere_root*>/logs/server1. Open the trace file.

If you compare traces from the sample application using the Application Profiling service and not using it, you will notice some differences. When retrieving the data for the specified customer, the Application Profiling service uses the read ahead hint. Compare the traces shown in Example 8-2 and Example 8-3 on page 383.

Example 8-2 No Application Profiling

```
[7/22/03 12:15:58:211 CEST] 5e03f599 WSRdbConnecti d getConnection
                                com.ibm.ws.rsadapter.cci.WSRdbConnectionFactoryImpl@4bd03591
                                com.ibm.ws.rsadapter.cci.WSRdbConnectionSpecImpl@d23690ae
```

```
  UserName     = null
  Password     = null
  Catalog      = null
  IsReadOnly   = null
  TypeMap      = null
  AccessIntent = (pessimistic update-weakestLockAtLoad)(collections: transaction/25) (resource
manager prefetch: 0) (AccessIntentImpl@d23690ae)

SELECT T1.ID, T1.NAME FROM ITSO.CUSTOMER  T1 WHERE T1.ID = ?

SELECT T1.ID, T1.BALANCE, T1.CUSTOMER_ID FROM ITSO.ACCOUNT  T1 WHERE T1.CUSTOMER_ID = ?
```

Example 8-2 on page 382 shows that the default Access Intent, weakestLockAtLoad, is being applied. Customer and Account data is retrieved in two steps. In addition, no prefetch is used, so Account rows need several trips to the database to be loaded.

Example 8-3 With Application Profiling: eBankWebUserProfile is being applied.

```
[7/23/03 13:33:05:460 CEST] 5b69d867 WSRdbConnecti d getConnection
                                com.ibm.ws.rsadapter.cci.WSRdbConnectionFactoryImpl@5cb7d842
                                com.ibm.ws.rsadapter.cci.WSRdbConnectionSpecImpl@be890c5f
  UserName     = null
  Password     = null
  Catalog      = null
  IsReadOnly   = null
  TypeMap      = null
  AccessIntent = (optimistic read)(collections: transaction/25) (resource manager prefetch:
5)(( account (ReadAheadItemImpl@78e1858)) (ReadAheadHintImpl@1))) (AccessIntentImpl@be890c5f)

SELECT A2.ID, A2.NAME, A1.ID, A1.BALANCE, A1.CUSTOMER_ID FROM ITSO.CUSTOMER  A2 LEFT OUTER JOIN
ITSO.ACCOUNT  A1 ON A2.ID = A1.CUSTOMER_ID WHERE A2.ID = ?
```

Example 8-3 shows the result of applying eBankWebUserProfile that you configured in the sample in 8.4.3, “Creating an Application Profile” on page 371. An optimistic read approach is used. Customer and Account data is retrieved in a single step. The use of a prefetch increment of 5 ensures that at least five rows will be loaded from the database in just one trip.

If you are using the Application Profiling API to dynamically set the task name at runtime, you should be able to manage the IllegalTaskNameException as seen in Example 8-1 on page 380.

9

Transactional Services

The transactional support provided by J2EE architecture can be too constrained for some application models. WebSphere Enterprise extends the architecture with new features for transactions. The possible transactional scenarios are:

- A transaction involving a single one-phase commit resource.

  This model is supported by J2EE as a Local Transaction.

- A transaction involving a single two-phase commit resource.

  This model is supported under J2EE as a Local Transaction.

- A transaction involving several one-phase commit resources.

  This model is not supported by the J2EE architecture. WebSphere Applications Server Enterprise V5 provides the ActivitySession service to support this model.

- A transaction involving several two-phase commit resources.

  This model is supported under J2EE as a global transaction.

- A transaction involving several two-phase commit resources and just a single one-phase commit resource.

  This model is not supported by the J2EE architecture. WebSphere Application Server Enterprise V5 provides the Last Participant Support to cover this model.

ActivitySession service and Last Participant Support are independent services.

This chapter first introduces all transaction-related concepts needed to understand Last Participant Support and ActivitySession services. Later, it separately explains how these services work, including configuration issues, runtime management and troubleshooting.

9.1 Transactions overview

In this book we use the following terms:

- Resource Manager Local Transaction (RMLT)
- Local Transaction Containment (LTC)
- Global transaction

Resource Manager Local Transaction (RMLT)

An RMLT is a resource adapter's view of a local transaction. It represents the unit of recovery on a single connection that is managed by the resource manager. The resource manager offers javax.resource.cci.LocalTransaction and java.sql.Connection interfaces to enable a bean or the container to request that the resource adapter commit or roll back its RMLT.

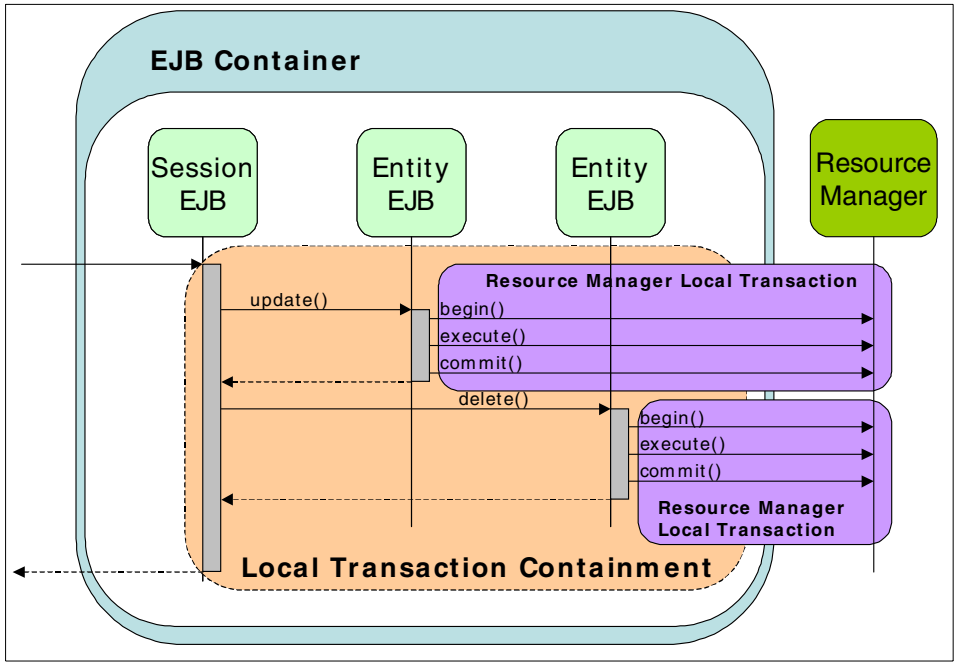


Figure 9-1 Resource Manager Local Transaction sample

The diagram in Figure 9-1 on page 386 illustrates two RMLTs invoked within the same LTC. Both RMLTs involve the same resource manager. The Entity EJBs that trigger the RMLTs are BMP Entity EJBs, so explicit invocation of begin and commit methods are needed.

Local Transaction Containment (LTC)

A LTC is a bounded *unit of work scope* where zero, one, or more Resource Manager Local Transactions (RMLT) may be accessed. The LTC defines the boundary at which all RMLTs must be complete. Any incomplete RMLT may be resolved according to the resolution policy specified at deployment time. An LTC context is always established by the container in the absence of a global transaction. An LTC is local to a bean instance. LTCs are not shared across beans even if those beans are managed by the same container. The J2EE specification defines the LTC on the bean method level.

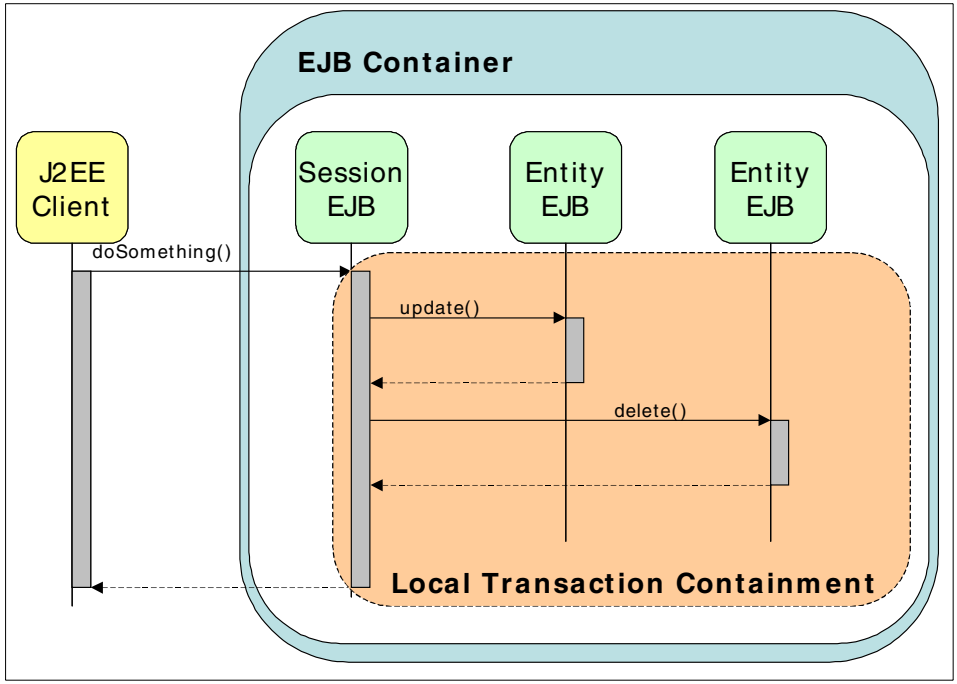


Figure 9-2 Local Transaction Containment sample

The diagram in Figure 9-2 illustrates an LTC where a Session EJB starts a Container Managed Transaction on request of a J2EE client call. Two different Entity EJBs are involved in the LTC.

Global transaction

Global transactions are supported by resource managers capable of dealing with the two-phase commit protocol. These resource managers implement the javax.transaction.xa.XAResource interface. The application server uses a Transaction Manager to coordinate the transaction. The Transaction Manager is external to any of the resource managers and it is provided by the application server.

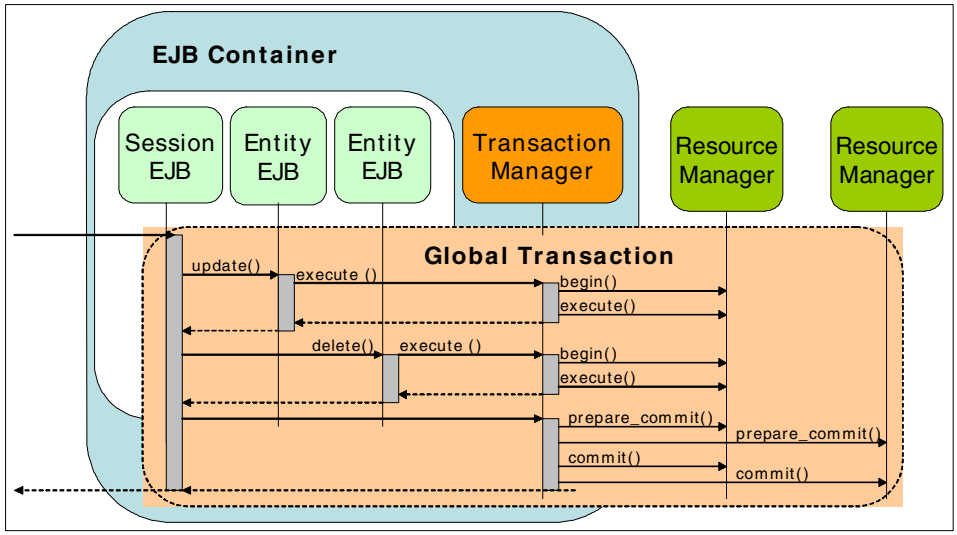


Figure 9-3 Global transaction

Figure 9-3 shows a transaction where two Container Managed (CMP) entity beans are backed up by two different XA resource managers. A global transaction context is needed. The Transaction Manager coordinates both resource managers using the two-phase commit protocol. There is no RMLT in a global transaction. From the resource manager's point of view, the transaction cannot be local, since it is externally managed by the Transaction Manager.

The application server will provide the following functions:

- Inform the transaction manager when a transaction begins.
- Perform the work of the transaction.
- Tell the transaction manager to commit the transaction.

The transaction manager uses the XAResource interface to coordinate the two-phase commit process across multiple resource managers. Two-phase commit, as seen in Figure 9-4 on page 389, works as follows:

1. In phase 1, the transaction manager asks all resource managers to prepare to commit their work. If a resource manager can commit its work, it replies

affirmatively, and hardens its recoverable data to the permanent storage. A negative reply reports an inability to commit for any reason.

2. In phase 2, the transaction manager directs all resource managers either to commit or roll back the work done on behalf of the global transaction, based on the replies from phase 1.

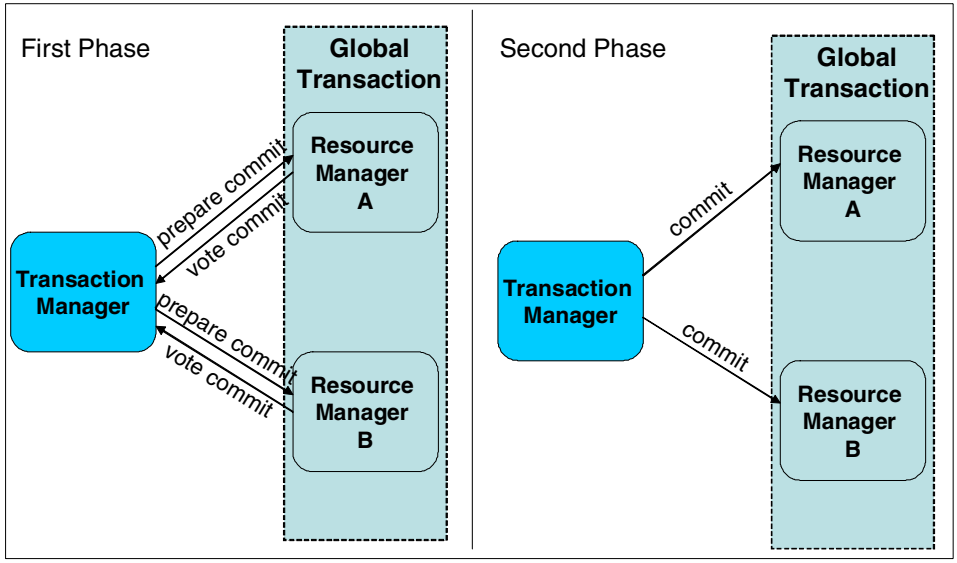


Figure 9-4 Two-phase commit

9.2 Last Participant Support

Last Participant Support allows the single one-phase commit resource to be involved in a global transaction where multiple two-phase commit resources execute. Multiple interactions may occur involving the one-phase commit resource in the same transaction, but only one such resource may be involved.

It is possible for a resource adapter that does not implement the XAResource interface to participate in a global transaction using Last Participant Support. This allows the use of a single one-phase commit resource in a global transaction, along with any number of two-phase commit resources. At transaction commit, the two-phase commit resources will first be prepared. If this is successful, the one-phase commit resource will be called to commit, followed by the call to commit for two-phase commit resources. See Figure 9-5 on page 390.

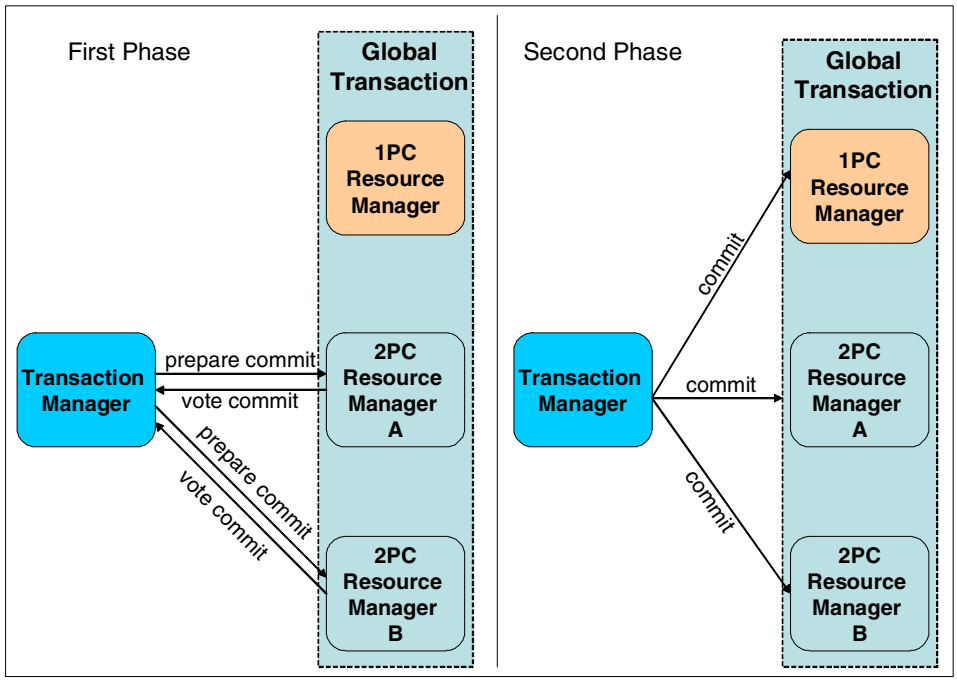


Figure 9-5 Last Participant Support

Last Participant Support introduces the hazard of a mixed resolution of the global transaction if the one-phase commit resource succeeds and one or more of the two-phase commit resources fail during the commit phase. The one-phase commit resource cannot be recovered.

Last Participant Support cannot be considered a substitute for the two-phase commit protocol. Applications that use Last Participant Support must be structured to handle the hazard mentioned before.

9.2.1 Configuration

Configuration of an Enterprise Application for Last Participant Support can be achieved using the Application Assembly Tool or using the Administrative Console. In both cases, the configuration tasks are reduced to a check box that enables or disables the acceptance of a heuristic hazard.

Application Assembly Tool

To enable Last Participant Support from the Application Assembly Tool, follow these steps:

1. Open the sample enterprise application archive ACompany.ear in the Application Assembly Tool.
2. On the left pane, select the **ACompany** node.
3. On the right pane, select **WAS Enterprise** tab.
4. On the WAS Enterprise tab, select **Accept heuristic hazard**. See Figure 9-6.

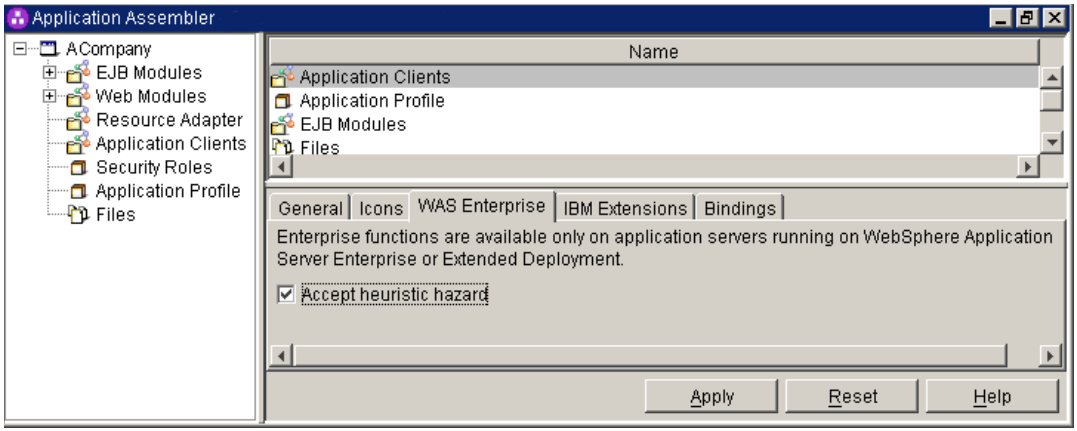


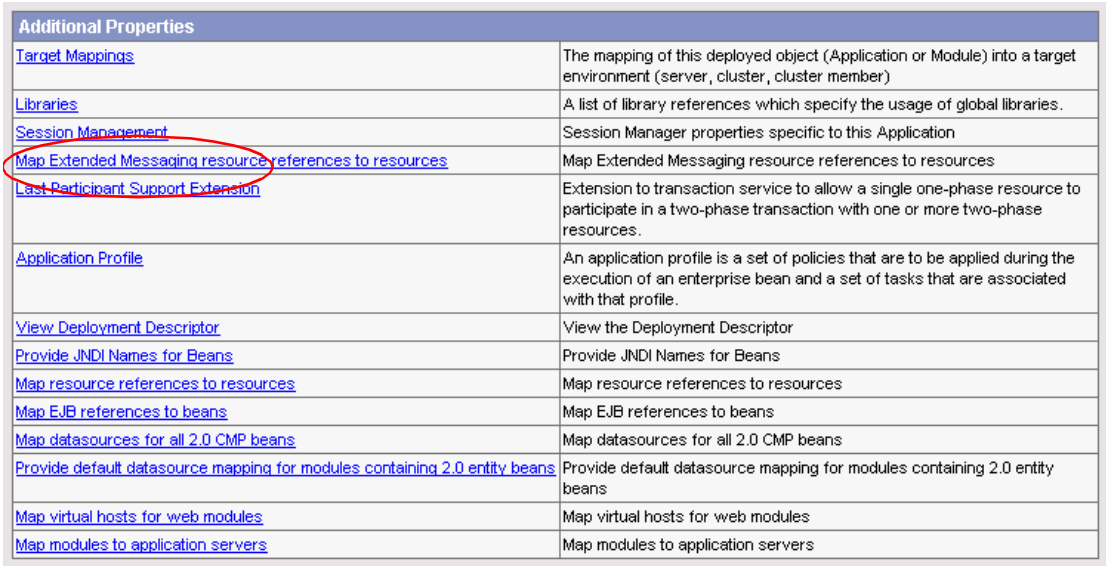
Figure 9-6 Application Assembly Tool: Last Participant Support enablement

5. Click **Apply** to make the configuration update.
6. Save and close the EAR file.

Administrative Console

To enable Last Participant Support from the Administrative Console, follow these steps:

1. Launch the Administrative Console and log in.
2. Select **Applications -> Enterprise Applications**.
3. On the right pane, select the name of your application: **ACompany**.
4. Under the Configuration tab, scroll down if necessary until you can see the Additional Properties table. Under Additional Properties, select **Last Participant Support** extension. See Figure 9-7 on page 392.



Additional Properties	
Target Mappings	The mapping of this deployed object (Application or Module) into a target environment (server, cluster, cluster member)
Libraries	A list of library references which specify the usage of global libraries.
Session Management	Session Manager properties specific to this Application
Map Extended Messaging resource references to resources	Map Extended Messaging resource references to resources
Last Participant Support Extension	Extension to transaction service to allow a single one-phase resource to participate in a two-phase transaction with one or more two-phase resources.
Application Profile	An application profile is a set of policies that are to be applied during the execution of an enterprise bean and a set of tasks that are associated with that profile.
View Deployment Descriptor	View the Deployment Descriptor
Provide JNDI Names for Beans	Provide JNDI Names for Beans
Map resource references to resources	Map resource references to resources
Map EJB references to beans	Map EJB references to beans
Map datasources for all 2.0 CMP beans	Map datasources for all 2.0 CMP beans
Provide default datasource mapping for modules containing 2.0 entity beans	Provide default datasource mapping for modules containing 2.0 entity beans
Map virtual hosts for web modules	Map virtual hosts for web modules
Map modules to application servers	Map modules to application servers

Figure 9-7 Administrative Console - Last Participant Support extension

5. Check **Accept Heuristic Hazard**.
6. Click **OK** to confirm the configuration change.
7. Save the configuration for WebSphere.
8. Restart the application for the changes to take effect.

9.2.2 Troubleshooting

Problems under the Last Participant Support extension can be derived from:

1. Mixing multiple single one-phase commit resources with one or more two-phase commit resources under the same transaction.

   This is not a supported scenario. Last Participant Support only accepts a single one-phase commit resource.

2. Mixing just a single one-phase commit resource with one or more two-phase commit resources under the same transaction without enabling Last Participant Support.
3. Mix of the previous two problems.

Tracing

When you work on a scenario similar to the one explained in the previous section under the second (2) point, default traces are enough to diagnose the problem.

You can view traces in the file <WAS_INSTALL>/logs/<servicemen>/SystemOut.log or using the Administrative Console. The Administrative Console not only shows traces but gives a complete diagnosis, and sometimes a solution, for the referred problem.

If you try to run the sample application eBankLPS without Last Participant Support enabled, the runtime environment will throw an exception if the application has a heuristic hazard and you have not chosen to accept it.

Look at the bottom of the Administrative Console to check for errors. Clicking the link of new errors will show a detailed description of the exceptions thrown as seen in Figure 9-8. Three exceptions are thrown related to this misuse of a single one-phase commit resource manager with one or more two-phase resource managers without enabling Last Participant Support.

Timestamp	Message Originator	Message
Thu Apr 17 15:18:37 EDT 2003	com.ibm.ejs.container.util.ExceptionUtil	CNTR0020E: Non-application exception occurred
Thu Apr 17 15:18:36 EDT 2003	com.ibm.ejs.j2c.LocalTransactionWrapper	J2CA0030E: Method enlist caught java.lang.Illega
Thu Apr 17 15:18:36 EDT 2003	com.ibm.ejs.jts.jta.TransactionImpl	WTRN0063E: An illegal attempt to enlist a one ph

Figure 9-8 Administrative Console exceptions

The first exception is triggered by the container. In the list shown in Figure 9-8, you can see time stamps and realize that exceptions are ordered first to last from bottom to top. Click in the first exception to see the details as seen in Figure 9-9.

General Properties		
Message	WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable resources has occurred.	Message text as received from the server runtime
Message type	Error	Type of message
Explanation	The transaction service has refused an attempt to enlist a one phase capable resource with a transaction already involving other two phase capable resources, as the application does not accept the heuristic risk that this would involve.	Explanation
User action	Ensure that one and two phase capable resources are not involved in the same transaction, or reconfigure the application to accept the heuristic risk that this would involve.	Recommendation
Message Originator	com.ibm.ejs.jts.jta.TransactionImpl	Originator of the event
Source object type	RasLoggingService	Type of the source object
Timestamp	Thu Apr 17 15:18:36 EDT 2003	Time when the event was fired
Thread Id	7fb2240f	Java runtime thread ID where the event was encountered
Node name	mka0knmy	Node which fired the event
Server name	server1	Server which fired the event

Figure 9-9 Administrative Console: error details

The Explanation row of the General Properties table has enough information to recognize the problem. The User Action row suggests corrective actions.

Heuristic reporting

Using Last Participant Support introduces a hazard of a mixed output if one or more two-phase resources fail to commit in the commit phase. At this moment, the *prepare commit* has been issued and the one-phase commit resource has been committed. No rollback can be performed on the one-phase commit resource. The output is mixed, as there will be committed and uncommitted resources.

A more unusual situation is the heuristic condition that happens when the system crashes when resolving the one-phase commit resource. In this case WebSphere Application Server will be unable to determine how the one-phase commit resource was resolved. The two-phase commit resources will be rolled back. When restarting the application server, it will inform you that such a heuristic condition has been reached, as seen in Example 9-1.

Example 9-1 Heuristic condition

```
[7/14/03 17:23:19:659 CEST] 6deed3fc TrLog         E WTRN0061W: A heuristic condition may have
occurred for transaction 000000010214b4786c9ae46a3c7e78287e72bc22039d18461f8a
```

It can be helpful when using Last Participant Support, as well as with ActivitySession service, to enable heuristic reporting. Heuristic reporting provides tracking of one-phase commit resources so that you can be sure if these resources have been correctly resolved or a mixed output state has been reached.

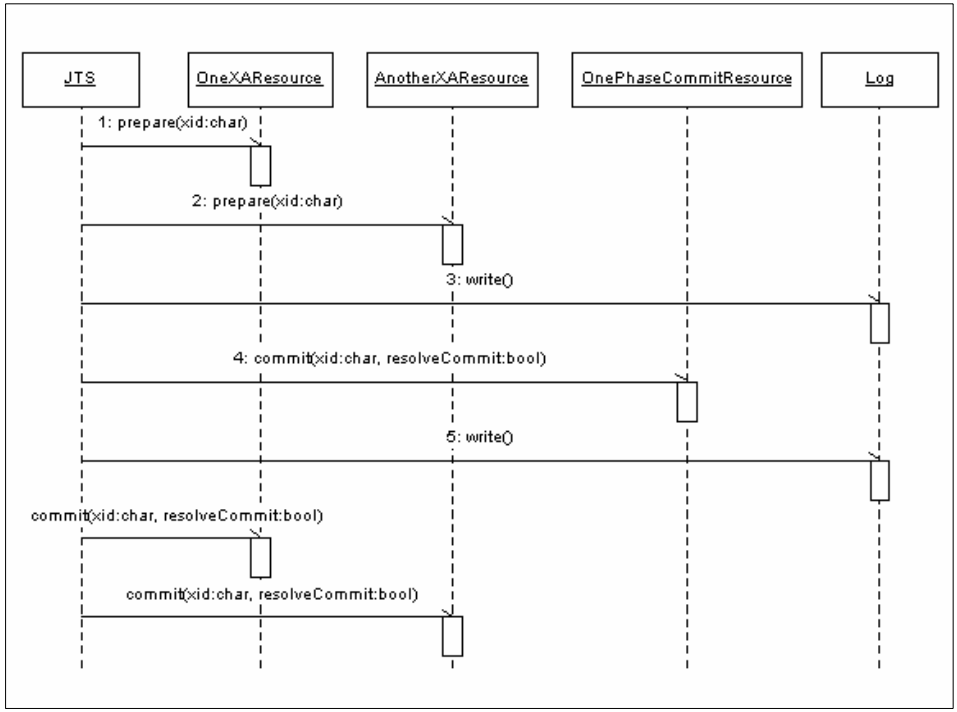


Figure 9-10 Heuristic reporting sequence diagram

Figure 9-10 shows how heuristic reporting works. The Java Transaction Service (JTS) coordinates the global transaction across resources. In this sample, there are two XA capable resources (two-phase commit) and a one-phase commit resource. The JTS starts with the prepare phase, as also seen in Figure 9-4 on page 389. In this phase, the JTS asks the XA resources for their disposition to commit. The second phase consists of committing the resources. The first to be committed is the one-phase resource. If heuristic reporting is enabled, a line is written to the log before and after committing the one-phase resource. If eventually any of the XA resources cannot commit, it will provoke a rollback in all the two-phase commit resources. The heuristic reporting will inform you whether the one-phase commit resource succeeded in committing, provoking a mixed outcome, or was also rolled back. Using heuristic reporting, you will be able to recover from inconsistent states induced by mixed outcome.

To configure heuristic reporting, open the Administrative Console:

1. Select **Servers -> Application Servers**.
2. In the right pane, under the Configuration tab, scroll down through the Additional Properties table to select **Transaction service**.

3. Select the **enableLoggingForHeuristicReporting** check box.

9.3 ActivitySession

EJBs cannot extend a Resource Manager Local Transaction (RMLT) beyond the boundary of an EJB method invocation. There is a need to extend support for ACID transactions. A J2EE application accessing one or more EJBs backed by non-transactional one-phase commit resources is not capable of coordinating these resources.

ActivitySession provides both a way to extend transaction boundaries beyond method invocation and to specify EJB activation and passivation by means of these new boundaries. ActivitySession becomes a new entity in the transactional scenario defining new boundaries for a unit of work.

ActivitySession can be associated with an HTTP session, so not only can the unit of work boundaries be defined by the client but the EJB life cycle can be scoped to that client.

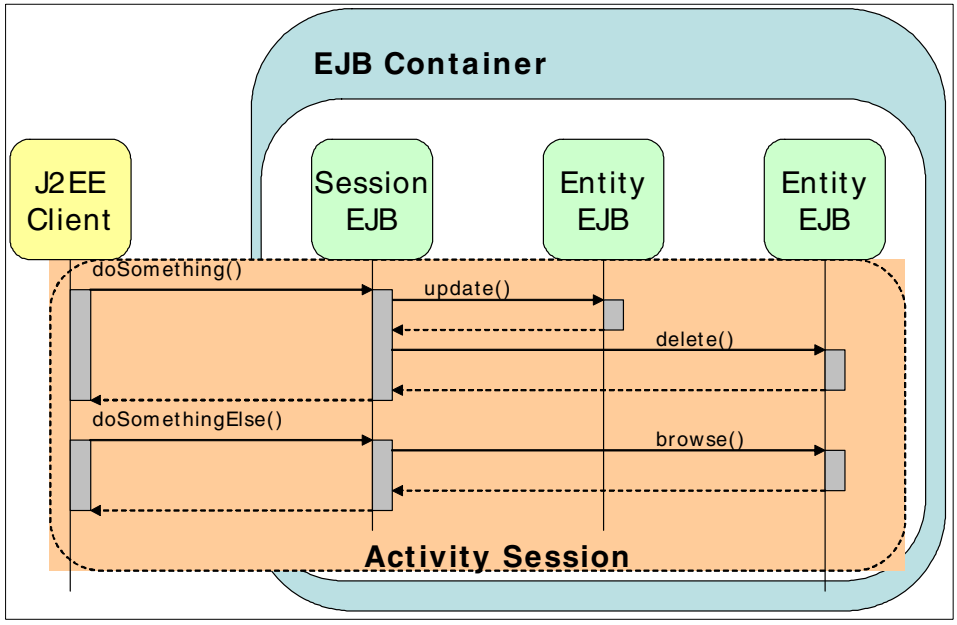


Figure 9-11 ActivitySession

Figure 9-11 shows how ActivitySession can extend the LTC. In this particular scenario, the ActivitySession is scoped to the client life cycle. Entity EJBs could be activated at the beginning of the ActivitySession and passivated at the end,

reducing the overhead of activating and passivating the EJBs on each method call.

There are a few usage rules that have to do with RMLTs, LTCs, global transactions and ActivitySessions:

- Nested ActivitySessions are not supported.
- ActivitySession boundaries cannot overlap.
- ActivitySessions may encapsulate one or more global transactions.
- Global transactions within an ActivitySession are independent of each other.
- It is not possible for an LTC to coexist with a global transaction.
- No one can wrap an ActivitySession; ActivitySessions exist on their own.

9.3.1 Extended Local Transaction

ActivitySession service extends the J2EE LTC beyond the method invocation providing Extended Local Transactions. Extended Local Transactions are offered in WebSphere Enterprise V5 by means of the ActivitySession as LTCs.

There are two patterns of LTC usage:

- Containment

  RMLTs within the LTC are started and completed by the application. RMLTs are said to be *contained* by the LTC. RMLTs that are not completed by the application by the end of the LTC boundary are cleaned up by the container in a direction (commit or rollback) determined by the unresolved action policy. This is a bean-managed or programmatic approach.

- Enlistment

  RMLTs are started by the container when the application first uses a connection. RMLTs are said to be *enlisted* in the LTC. RMLTs are completed by the container at the end of the LTC. This is a container-managed or declarative approach.

The ActivitySession service captures the commit operations of each local and global transaction and upholds them. At the end of the ActivitySession, the container will effectively commit or roll back each individual transaction.

The ActivitySession is not a substitute for the two-phase commit protocol. An ActivitySession may result in a mixed outcome, if any single-phase resource successfully gets committed before another resource fails to commit. In this case, the ActivitySession service will allow the programmer to retrieve the list of resources that were committed and those whose state is uncertain.

Configuration

Configuration of ActivitySession service requires two steps that resemble the transactional configuration specified by the Java Transaction Architecture (JTA):

1. Configure the ActivitySession component level properties.

   Components that support ActivitySession are:

   – EJB Components: Session and entity beans
   – Web Components: Servlets

2. Configure the Container ActivitySessions

   This step is only needed if the application is using an enlistment pattern. This is done in a manner very similar to the definition of Container Transactions in J2EE. ActivitySession boundaries are defined by specifying the ActivitySession support given by each EJB method.

ActivitySession component level properties

Some attributes have been added to the IBM extensions to manage LTCs:

- Boundary

  Identifies the containment boundary at which all contained RMLTs must be completed. Possible values are:

  – Bean method

    RMLTs must be resolved within the scope of the bean method where they were started. It is the default value.

  – ActivitySession

    If an ActivitySession context is present, RMLTs must be resolved within the ActivitySession scope. If no ActivitySession context is present the policy will be the same as if BeanMethod value was assigned.

Restriction: A value of ActivitySession is not valid for Stateless Session Beans.

- Resolution control

  Assigns the responsibility of initiating and ending RMLTs to a component. Possible values are:

  – Application

    The application code is responsible for explicitly starting RMLTs and completing them before reaching the LTC boundary. An enlistment pattern is used.

If connections are used without calling begin()::LocalTransaction or setAutoCommit(false)::Connection, then the connections will be auto-committed by the resource adapter or the underlying resource manager. This execution model may introduce the need for an application to be notified when the LTC boundary is ending in order to fulfill its responsibility to complete RMLTs it started. A synchronization interface is exposed to implement this need.

Any incomplete RMLTs by the end of the LTC boundary will be cleaned up by the container according to the value of the unresolved action attribute.

– Container

The container is responsible for both starting RMLTs and completing them within the LTC boundary. Container begins an RMLT when a connection is first used within the LTC scope and automatically completes it at the end of the LTC scope. The boundary may be either an ActivitySession or a bean method.

- Unresolved action

  Indicates the action the container will request any RMLT to take in case it is unresolved at the end of the LTC boundary.

Important: This attribute should only be applied when *resolution control* has a value of *application*. Unresolved action does not make sense in a context where it is the container that already decides what action to request on the resource managers with incomplete RMLTs. Application Assembly Tool will not check that you apply this constraint.

– Commit

Pending RMLTs will be instructed to commit by the container.

– Rollback

Pending RMLTs will be instructed to roll back by the container. This is the default value.

These parameters can be configured using the Application Assembly Tool or in WebSphere Studio IE. LTCs can be configured in EJBs as well as in Servlets.

- WebSphere Studio IE

  To configure LTCs from WebSphere Studio IE, follow these steps:

  – For EJBs:

    i. Open the EJB Deployment Descriptor.

    ii. Switch to the Beans tab.

iii. Select the EJB to configure.

iv. Scroll down the right pane to access the WebSphere Extensions.

v. Under WebSphere Extensions the three attributes (Boundary, Resolver and Unresolved action) appear below the Local Transaction 2.0 title.

– For Servlets:

i. Open the Web Deployment Descriptor.

ii. Switch to the Servlets tab.

iii. Select the Servlet to configure.

iv. Scroll down the right pane to access WebSphere Extensions.

v. The three attributes (Boundary, Resolver and Unresolved action) appear at the bottom.

Important: In the Servlet configuration these attributes only have meaning if ActivitySession is to be used. The Boundary attribute should be always set to ActivitySession since the Bean Method value does not apply. The Boundary attribute value should be inferred from the usage or not of the ActivitySession.

- Application Assembly Tool

  To configure LTCs from the Application Assembly Tool, follow these steps:

  – For EJBs:

  i. On the left pane, expand the tree by selecting **<enterprise_application> -> EJB Modules -> <ejb_module>** and then selecting either **Session Beans** or **Entity Beans**.

  ii. Select the Session or entity bean you want to configure.

  iii. On the right pane, switch to the IBM Extensions tab.

  iv. The three properties to be configured appear under the Local Transactions title.

  – For Servlets:

  i. On the left pane, expand the tree by selecting **<enterprise_application> -> Web Modules -> <web_module> -> Web Components**.

  ii. Select the Servlet you want to configure.

Important: In the Application Assembly Tool, the ActivitySession service configuration attributes are in two different tabs: WAS Enterprise and IBM Extensions. The name of one of the attributes is also changed. The Resolver attribute appears as ActivitySession control kind.

iii. Open the IBM Extensions tab to access the Unresolved action attribute.

iv. Switch to the WAS Enterprise tab to access the Resolver attribute. The Resolver attribute appears named as ActivitySession control kind. When set to None, no ActivitySession is used. If Application or Container values are applied, then Boundary is set to ActivitySession, since no other value may apply to Servlets.

Container ActivitySession

After configuring how the ActivitySession service is going to behave in each component, you should configure what support is expected by each method. A method may be associated with the following ActivitySession attribute values:

- Never

  An ActivitySession context is forbidden. Any violation causes a RemoteException to be thrown.

- Supports

  If an ActivitySession context is received it will be used. If no ActivitySession is present at invocation, none will be used.

- Not Supported

  No ActivitySession is needed. If one is received, it will be suspended. On method return, the ActivitySession will be resumed.

- Requires New

  If an ActivitySession context is received, it is suspended and a new ActivitySession is created. If no ActivitySession is present, a new one is created.

- Required

  If an ActivitySession context is received it will be used. If no ActivitySession is active when a method is called, a new ActivitySession is created.

- Mandatory

  An ActivitySession must be present at method invocation. Otherwise, an ActivitySessionRequiredException is thrown.

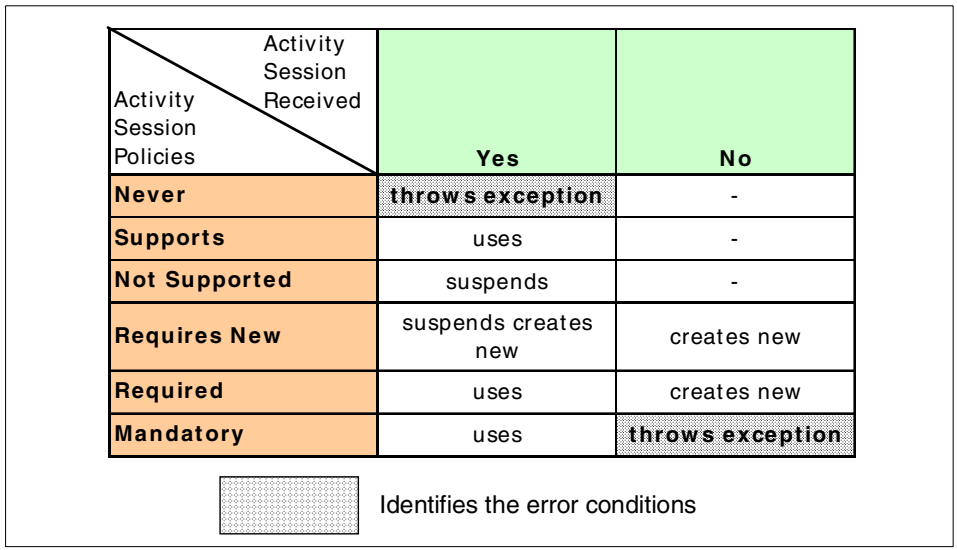


Figure 9-12 ActivitySession attribute values

Figure 9-12 shows the possible ActivitySession attribute values and their effect when an ActivitySession is received and when it is not.

Container ActivitySessions can only be configured in the Application Assembly Tool. There is no tool support for this in WebSphere Studio IE. Container ActivitySession configuration groups methods from different components and assigns this group a support level for the ActivitySession.To create a container ActivitySession, follow these steps:

1. In the Application Assembly Tool, expand the tree on the left pane by selecting **<enterprise_application> -> EJB Modules -> <ejb_module> -> Container ActivitySession**.
2. From the menu, select **File -> New -> Selected Object**. You will see the window shown in Figure 9-13 on page 403.

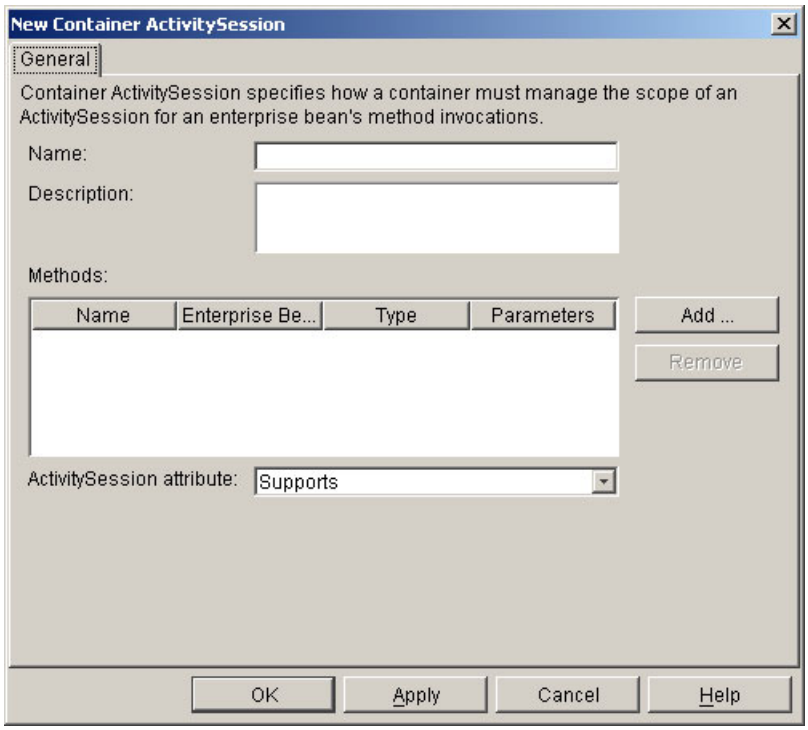


Figure 9-13 Application Assembly Tool - New Container ActivitySession

You must group methods with the same ActivitySession attribute under the same Container ActivitySession definition. Nevertheless, you can have different Container ActivitySessions with the same ActivitySession attribute value but with different naming. The name can add semantics to the Container ActivitySession, so you can identify the purpose of the associated methods.

3. Give a name to the Container ActivitySession.
4. Click the **Add** button. A window will open to browse the methods of the EJBs.
5. Select the methods you want to add and click **Apply** for each one.
6. Once you have finished adding methods, click **OK** to confirm the changes.
7. From the drop-down list, choose the ActivitySession attribute value required for these methods.
8. Click **OK** to create the new Container ActivitySession.

ActivitySession service API

If the Resolver is set to Application, a containment pattern is being used and coding using the ActivitySession service API is needed for the application. You can achieve the same goals using ActivitySession programmatically using the API instead of configuring Container ActivitySessions.

To access the ActivitySession service API, you need to add the activitySession.jar to your project's build path. Follow these steps:

1. In WebSphere Studio IE, select **Window -> Open Perspective -> Java**.
2. In the Package Explorer view, located in the left pane, right-click the Web Project or EJB Project where you want to use the ActivitySession service. From the pop-up menu, select **Properties**.
3. On the left pane of the Properties window, select **Java Build Path**.
4. On the right pane of the Properties window, select **Libraries**.
5. Click the **Add** variable button. A window opens showing all available variables as seen in Figure 9-14.

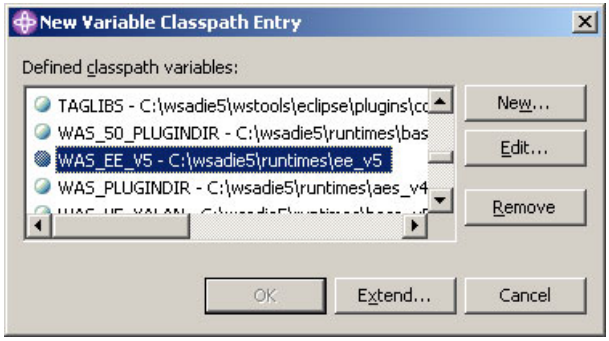


Figure 9-14 Build path variable

   a. Select the **WAS_EE_V5** variable. Click the **Extend** button to complete the path to the JAR file you need. A window opens and shows a directory tree.
   b. Navigate in the directory tree to the lib folder. Select **activitySession.jar**. Click **OK** to add the new library.
6. Click **OK** to confirm the changes.

WebSphere Studio IE will rebuild your project and the ActivitySession service will be available to your code.

The ActivitySession service is exposed to application programmers through the com.ibm.websphere.ActivitySession.UserActivitySession interface. The methods exposed by this interface are shown in Example 9-2 on page 405.

Example 9-2 UserActivitySession

```
public interface UserActivitySession {

   public static final int EndModeCheckPoint = 0;
   public static final int EndModeReset = 1;
   public static final int StatusSessionActive = 0;
   public static final int StatusSessionCompleting = 1;
   public static final int StatusSessionCompleted = 2;
   public static final int StatusNoSession = 3;
   public static final int StatusUnknown = 4;

   public abstract void beginSession();
   public abstract void endSession(int i);
   public abstract void resetSession();
   public abstract void checkpointSession();
   public abstract int getStatus();
   public abstract String getSessionName();
   public abstract void setSessionTimeout(int i);
   public abstract int getSessionTimeout();
   public abstract void setResetOnly();
}
```

Important: Throws clauses have been removed from the code in Example 9-2 for a clearer view.

An ActivitySession is started with a beginSession method call and ended with a endSession method call.

The UserActivitySession lets you checkpoint work to commit changes made up to a specific point without ending the ActivitySession.

The resetSession method call causes the ActivitySession service to roll back all changes to the last checkpoint.

You can programmatically set a timeout for the ActivitySession through the setTimeOut method call. The time is specified in seconds. It can also be set through the Administrative Console for each server instance, as seen in 9.4, “Runtime” on page 410.

The implementation of this interface can only be accessed through a JNDI lookup. The UserActivitySession implementation is bound to the java:comp/websphere/UserActivitySession name. Example 9-3 shows how to use the UserActivitySession.

Example 9-3 UserActivitySession usage

```
//perform lookup
```

```
InitialContext ic= new InitialContex();
UserActivitySession uas=
  (UserActivitySession)ic.lookup("java:comp/websphere/UserActivitySession");
uas.beginSession();

//do some work
myBean.doSomeWork();

//mid checkpoint
uas.checkpointSession();

//do some more work
myOtherBean.doSomeWork();

//end the session
uas.endSession(UserActivitySession.EndModeCheckPoint);
```

Important: Necessary try/catch statements have been removed for clarity. For a detailed description of exceptions needed to be caught, refer to "Exception handling" on page 411.

A Web application can use both transactions and ActivitySessions. Any transactions started within the scope of an ActivitySession must be ended by the Web component that started them and within the same request dispatch.

Important: Actually WebSphere Studio IE V5.0.1 has a bug that inhibits the use of the ActivitySession service under WebSphere Test Environment EE. To correct this configuration problem, edit the file named <wsad_ie_install>/runtimes/ee_v5/properties/implfactory.properties. Search for the key `com.ibm.websphere.csi.ContainerExtensionFactory`. Assign it the following value: `com.ibm.ejs.csi.ContainerExtensionFactoryPMEImpl`.

9.3.2 Extended EJB life cycle

Without ActivitySession service support, there were only two different ways to keep an EJB within scope between multiple method calls:

- Use of commit option A.

  This option prevents the EJB from being workload managed and disallows sharing the database tables with other applications.

- Start a global transaction.

  Global transactions need external resources (not only from the resource manager) to coordinate the different RMLTs. A transaction service such as the JTS must be used. This carries a performance weight.

ActivitySession service lets you extend the life cycle of an EJB without the performance drawback of a global transaction and avoiding the restrictions of using commit option A.

An EJB activation can be scoped to the ActivitySession, rather than being scoped to the transaction boundaries, offering additional control on the timing for EJB activation and passivation, with potential performance benefits.

This feature can highly improve performance by avoiding unnecessary passivations. EJBs remain active and ready.

Configuration

New EJB Bean cache policies have been added to existing IBM extensions. The Bean cache policies are governed by two attributes:

- Activate At

  Defines when an EJB is activated and placed in cache as well as when it is removed from cache and passivated.

  - Once

    The EJB is activated and put in cache the first time it is accessed. Removal and passivation depend on the container.

  - Transaction

    Activation and passivation as well as cache life time are governed by transaction boundaries. The EJB is activated and put in cache when the transaction begins and passivated and removed from cache when the transaction ends.

  - ActivitySession

    Activation and passivation as well as cache life time are governed by ActivitySession boundaries. This is a new policy.

- Load At

  Specifies when the container should synchronize the bean with the persistence layer. This implies whether the container has exclusive or shared access to the database.

  - Activation

    The EJB is loaded from the persistent storage when it is activated regardless of the Activate At policy. The container will use exclusive access to the persistence layer.

  - Transaction

    The EJB is loaded at the start of the transaction. The container will use shared access to the persistence layer.

The Bean cache policy governs which commit option will be used for the specific bean where it is configured.

Load At \ Activate At	Once	Transaction	Activity Session
Activation	A	Unsupported	Unsupported
Transaction	B	C	C+

Figure 9-15 Bean cache policies and commit options

Figure 9-15 shows which commit options will be used depending on the Load At and Activate At values of the Bean cache policy.

According to the EJB 2.1 specification, supported commit options are A, B and C. Commit option C+ is an IBM extension to support an EJB life cycle bounded to the Activity Session boundaries. With commit option C+, the EJB is activated only once per ActivitySession, but data may be loaded or stored multiple times within that session at checkpoints.

- Option A

  When the transaction ends, the container caches the instance and it remains valid across transactions. The container invokes ejbStore to synchronize the instance. The instance remains both ready and valid, so the container will not load it again from the persistent storage when the next transaction begins. Exclusive access to the instance is ensured by the container.

- Option B

  The container invokes ejbStore, caches the instance but marks it as not valid. When a new transaction begins, the container invokes ejbLoad to synchronize the bean, since the container does not ensure exclusive access.

- Option C

  The container does not cache the instance. At the end of the transaction, the container invokes ejbStore and ejbPassivate and returns the instance to the pool. At the beginning of the next transaction, ejbActivate and ejbLoad are invoked on the retrieved instance to synchronize the state of the EJB.

- Option C+

  This option is different from the others because instead of referring to transaction boundaries, it refers to ActivitySession boundaries. An ActivitySession may contain multiple transactions, global or local. With commit option C+, an EJB is stored at the transaction end and loaded at the beginning of the next transaction. No activation or passivation is needed. The same object is reused between transactions.

Figure 9-16 summarizes these properties. The Write column specifies whether the EJB is written to persistent storage at the transaction end. The Ready column specifies if the instance remains ready for the next transaction. If ready, the instance is not returned to the pool. The Valid column specifies if the instance remains valid for the next transaction. If valid, the instance does not need to be synchronized with the persistent storage at the beginning of the next transaction.

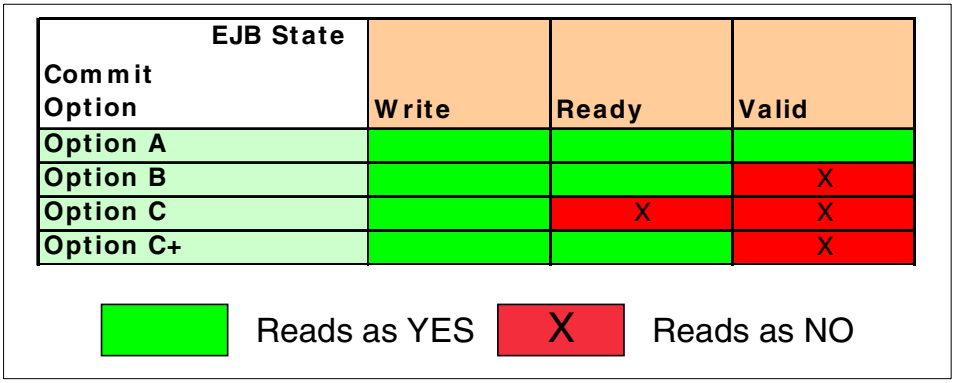


Figure 9-16 Commit options

The EJB life cycle is influenced by the commit options. Depending on the commit option used by the container, some EJB callback methods may be obviated to achieve better performance.

Figure 9-17 shows which methods get invoked in a scenario where an Entity EJB is invoked in two consecutive transactions. The figure illustrates whether the selected method is invoked or not at the end of the first transaction and at the beginning of the second transaction.

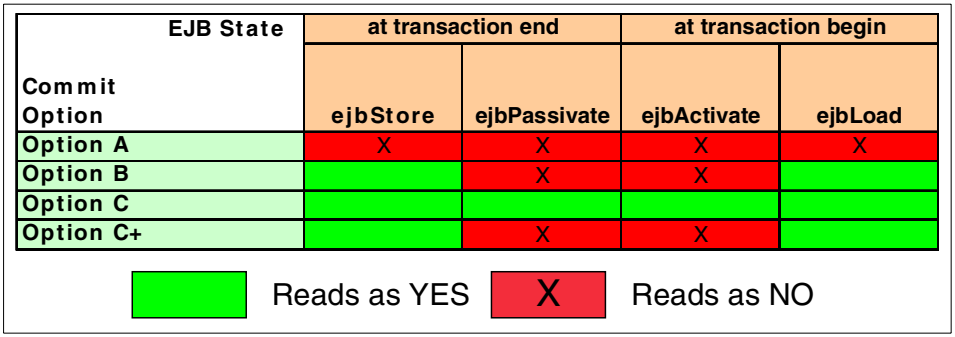


Figure 9-17 EJB callback methods invocation depending on commit options

Important: Both Figure 9-16 and in Figure 9-17 show different views of the same data. Notice that Option C+ in an ActivitySession context behaves similar to Option B in a transaction context. Option C+ is the only possible option when using ActivitySession service for scoped activation.

9.3.3 Usage scenarios

Using the ActivitySession service in association with a Web container lets you extend and define transactions of EJBs during the client's life cycle.

Coordination of one-phase commit resources is not supported by the J2EE specification. ActivitySession adds more flexibility to the constrained J2EE model, enabling the coordination of one-phase commit resources under a unit of work.

Client scoped life cycle

ActivitySessions used within a Web container are automatically associated to the HTTP Session. An EJB can be activated and passivated in ActivitySession boundaries, as was seen in 9.3.2, "Extended EJB life cycle" on page 406. Therefore, it is possible to span the life of an EJB and map it to the client's life cycle.

Client side demarcation

The association between the ActivitySession and the HTTP Session also rewards you with the possibility of client-side defined transactions. ActivitySession lets you coordinate a number of one-phase commit resources within the boundaries of a single unit of work. You can span this unit of work through multiple client invocations.

Coordination of one-phase commit resources

ActivitySession service allows the coordination of multiple one-phase commit resources within the boundaries of a single unit of work. It is not a substitute for the two-commit phase protocol and introduces a heuristic hazard.

9.4 Runtime

Although ActivitySession service usage can lead to a mixed outcome when coordinating several one-phase commit resources, there is no need to accept the heuristic hazard.

9.4.1 Enable the ActivitySession service

The only runtime configuration needed is the enablement of the ActivitySession service on the server you want to deploy your application to. To enable the ActivitySession service, follow these steps:

1. In the left pane of the Administrative Console, select **Servers -> Application Servers**.
2. In the right pane, select the server where you want to enable the ActivitySession service. The default server name is server1.
3. Select the **Configuration** tab, if not already selected.
4. Scroll down the Additional Properties table and click in **ActivitySession Service**.
5. Select the **Startup** check box and specify a default timeout value. The default value is 5 seconds. This means that if the ActivitySession is not resolved after five seconds, it will be rolled back.

9.4.2 Troubleshooting

This section describes how to use WebSphere tracing facilities to recover from errors derived from mixed outcomes. It also shows how to deal with exceptions when using a bean-managed ActivitySession.

Exception handling

When using a bean-managed ActivitySession, several exceptions must be taken into account, depending on the invoked method:

- begin()::UserActivitySession
  - ActivitySessionAlreadyActiveException

    You tried to start an ActivitySession while another one was already present. ActivitySession boundaries cannot overlap with each other.
  - SystemException

    Unknown error starting a new ActivitySession.
  - NotSupportedException

    ActivitySession is not supported by a nested call within this ActivitySession context.
  - TransactionPendingException

    You tried to start a new ActivitySession while a transaction was still pending. ActivitySessions and transactions cannot overlap their boundaries.

- endSession()::UserActivitySession
  - MixedOutcomeException

    A heuristic condition has been reached. Some of the resources were able to commit, whereas others were rolled back.
  - NotOriginatorException

    You tried to end the ActivitySession from a component other than the one that started the ActivitySession. ActivitySessions must be started and resolved within the same component.
  - SystemException

    Unknown error when resolving the ActivitySession.
  - NotSupportedException

    ActivitySession is not supported by a nested call within this ActivitySession context.
  - ActivitySessionResetException

    Unknown error when trying to reset the current ActivitySession.
  - NoActivitySessionException

    You tried to end an ActivitySession context but no ActivitySession context was present.
  - ContextPendingException
  - ActivitySessionPendingException

Heuristic reporting

You can also use the heuristic reporting facility when using the ActivitySession service. Refer to “Heuristic reporting” on page 394.

9.5 JTA extensions

Programming Model Extensions have also been introduced to JTA. These extensions are available both to Web and J2EE containers. Two new mechanisms have been added:

- Callback registration

  Provides a mechanism for applications to be notified when a transaction completes.
- Transaction identity

  Provides a mechanism to determine the transaction identity.

For more information on JTA Extensions, refer to the WebSphere InfoCenter.

9.6 Samples

Several sample applications are provided for illustration:

- eBankBase

  This is the base sample. Contains two modules:

  – eBankWebBase: a Web module.
  – eBankEJBBase: an EJB module.

  The base sample is a simple banking application where you can select a customer ID, browse accounts, and perform transfers.

- eBankLPS

  This sample extends the base sample by simulating a CICS back end with a one-phase commit data source. It is a typical scenario where data is not only backed locally to the application but also propagated to legacy systems that do not provide two-phase commit connectors.

- eBankLPSFail

  This sample uses a custom one-phase commit resource that sleeps the thread at commit. If you "kill" the WebSphere Application Server process at this time, you will meet a heuristic condition. Check SystemOut.log for the precise time of killing the process, as shown in Example 9-4. Also check the server.pid file to see which server process ID you have to kill.

Example 9-4 SystemOut ready to kill WAS

```
[7/14/03 17:22:12:372 CEST] 2225d3b9 SystemOut     O HEY!!! kill WAS now!!!
[7/14/03 17:22:12:592 CEST] 2225d3b9 SystemOut     O XAResource received end
with TMSUCCESS
[7/14/03 17:22:12:592 CEST] 2225d3b9 SystemOut     O In Commit -- CRASH NOW!
```

- eBankLifeAS

  This sample illustrates the EJB extended life cycle, and bounds an ActivitySession to an HTTP Session. The Customer entity bean has been configured to be activated and passivated at ActivitySession boundaries. You can compare activation and passivation events on the Customer bean with those in the base sample. In the base sample, shown in Example 9-5, Customer is activated and passivated at method boundaries.

Example 9-5 Sample without extended life cycle

```
O -BEGIN-Logon
O -END-Logon
```

```
 -BEGIN-getAccountDataForCustomer
O Customer [null] activated!
O Customer [01] loaded!
O -END-getAccountDataForCustomer
O Customer [01] stored!
O Customer [01] passivated!
I SRVE0180I: [eBankWebAS] [/eBankWebAS] [Servlet.LOG]: /accountList.jsp: init

 -BEGIN-browseConfirmedTransfers
O -END-browseConfirmedTransfers
I SRVE0180I: [eBankWebAS] [/eBankWebAS] [Servlet.LOG]:
/browseConfirmedTransfers.jsp: init

 -BEGIN-getAccountDataForCustomer
O Customer [null] activated!
O Customer [01] loaded!
O -END-getAccountDataForCustomer
O Customer [01] stored!
O Customer [01] passivated!

 -BEGIN-confirmTransfer
O -END-confirmTransfer
I SRVE0180I: [eBankWebAS] [/eBankWebAS] [Servlet.LOG]: /confirmTransfer.jsp:
init

 -BEGIN-getAccountDataForCustomer
O Customer [null] activated!
O Customer [01] loaded!
O -END-getAccountDataForCustomer
O Customer [01] stored!
O Customer [01] passivated!

 -BEGIN-Logoff
O -END-Logoff
```

In the sample shown in Example 9-6, Customer is activated at logon and passivated at logoff.

Example 9-6 Sample with extended EJB life cycle

```
O -BEGIN-Logon
O -END-Logon

O -BEGIN-getAccountDataForCustomer
O Customer [null] activated!
O Customer [02] loaded!
O -END-getAccountDataForCustomer
O Customer [02] stored!
```

```
I SRVE0180I: [eBankWebAS] [/eBankWebAS] [Servlet.LOG]: /accountList.jsp: init

O -BEGIN-browseConfirmedTransfers
O -END-browseConfirmedTransfers
I SRVE0180I: [eBankWebAS] [/eBankWebAS] [Servlet.LOG]:
/browseConfirmedTransfers.jsp: init

O -BEGIN-getAccountDataForCustomer
O Customer [02] loaded!
O -END-getAccountDataForCustomer
O Customer [02] stored!

O -BEGIN-getAccountDataForCustomer
O Customer [02] loaded!
O -END-getAccountDataForCustomer
O Customer [02] stored!

O Customer [02] passivated!
O -BEGIN-Logoff
O -END-Logoff
```

Apart from the provided EAR files, some other files are provided to help you create the testing environment:

- ebank.creation.sql

  Contains SQL statements to create necessary tables and data structures needed for the two-phase commit resource.

- backend.creation.sql

  Contains SQL statements to create necessary tables and data structures needed for the one-phase commit resource that will simulate a CICS server. Transfer data will be duplicated in this database.

- customer.txt

  Sample data for the customer table.

- account.txt

  Sample data for the account table.

To create a test environment and install all the samples, follow these steps:

1. From the DB2 control center, create a database called EBANK using the wizard. Use the default values.
2. From the DB2 control center, create a database called BACKEND using the wizard. Use the default values.
3. Copy all the files listed above to a specific directory in your file system, <install_files_dir>.

4. Open a DB2 command window and move to the directory named <install_files_dir>.
5. Type the following command: **`db2 -tvf ebank.creation.sql`**
6. Type the following command: **`db2 -tvf backend.creation.sql`**

Tip: At this point ,all the databases, tables, and sample data has been created and imported. Check that you have two databases: EBANK and BACKEND. Check the database EBANK. There should be three tables: CUSTOMER, ACCOUNT and TRANSFER. Check that CUSTOMER and ACCOUNT contain data. Check that the BACKEND database contains one table: TRANSFER. Note that all tables are under the ITSO schema.

7. Open the Administrative Console and create an authentication alias to connect to the databases:
   a. In the Administrative Console, select **Security -> JAAS Configuration -> J2C Authentication Data**. Click **New**.
   b. Set the J2C Authentication Data Alias to ebankDbUser.
   c. Set the J2C Authentication Data User ID to the User ID with which you access DB2.
   d. Set the J2C Authentication Data Password to the password with which you access DB2.
   e. Click **OK** to accept the changes. Then click **Save** to prepare for saving the changes. Finally, click the **Save** button to confirm changes and save them to the configuration files.
8. Open the Administrative Console and create the two data sources: an XA data source to connect to EBANK and a one-phase commit data source to connect to BACKEND.
   a. To create the two-phase commit data source, do the following:
      i. In the Administrative Console, select **Resources -> JDBC Providers**.
      ii. In the right pane, click **New**.
      iii. From the JDBC providers combo box, select **DB2 JDBC Provider (XA)** and click **OK**.
      iv. Give this provider the following name: DB2eBankXAProvider. Click **OK** to accept the changes.
      v. Select the recently created provider: **DB2eBankXAProvider**.
      vi. Scroll down to Additional Properties and select **Data Sources** and click **New**.

vii. Set the Data Source Name to ebank.

viii. Set the Data Source JNDI Name to jdbc/ebank.

ix. Set the Component-managed Authentication Alias to ebankDbUser.

x. Set the Container-managed Authentication Alias to ebankDbUser.

xi. Click **OK** to accept changes. Then click **Save** to prepare for saving the changes. Finally, click the **Save** button to confirm the changes and save them to the configuration files.

b. To create the one-phase commit data source:

i. In the Administrative Console, select **Resources -> JDBC Providers**.

ii. In the right pane, click **New**.

iii. From the JDBC providers combo box, select **DB2 JDBC Provider** and click **OK**.

iv. Give this provider the following name: DB2eBankNonXAProvider. Click **OK** to accept the changes.

v. Select the recently created provider **DB2eBankNonXAProvider**.

vi. Scroll down to Additional Properties and select **Data Sources** and click **New**.

vii. Set the Data Source Name to backend.

viii. Set the Data Source JNDI Name to jdbc/backend.

ix. Set the Component-managed Authentication Alias to ebankDbUser.

x. Set the Container-managed Authentication Alias to ebankDbUser.

xi. Click **OK** to accept the changes. Then click **Save** to prepare for saving the changes. Finally, click the **Save** button to confirm changes and save them to the configuration files.

9. Install the provided Enterprise Applications in WebSphere Application Server. You do not have to modify any of the default settings already provided in the sample applications.

10. Test the installation. Open a Web browser and type the following URL: `http://<host>:<port>/<application_name>`.

    If you are running a local WebSphere Application Server installation with default settings the URL will be `http://localhost:9080/<application_name>`.

    Possible values for `<application_name>` are eBankBase, eBankLPS, eBankLPSFail, or eBankLifeAS.

    Once running the application, possible customer IDs are 01, 02 or 03. Possible account numbers are 001, 002, 003, 004, 005, 006.

10

Business Rule Beans

This chapter explains how to use the Business Rule Beans in building enterprise applications. It includes the following topics:

- Overview of the Business Rule Beans framework
- Design considerations for using Business Rule Beans in enterprise applications
- Implementation of business rules
- Testing of business rules
- Preparation for business rules deployment

The second part of this chapter explores the runtime aspects of Business Rule Beans:

- Deployment
- Troubleshooting
- Performance considerations
- Security considerations

10.1 Planning

The Business Rule Beans framework extends the scope of the WebSphere Application Server Enterprise to support business applications that externalize their business rules. Business Rule Beans are designed to remove the volatile and rapidly changing components of a system, for example the business rules, from the application to allow changes to be made to the application without touching the applications' core components, such as user interfaces, database interfaces, and business object structure.

Rule externalization is accomplished by extending the application analysis and design processes to identify the points of variability in application behavior. Business Rule Beans are implemented as standard Java components (Java Beans or Enterprise Java Beans) that are managed by the WebSphere environment (much like an EJB component). To access the Business Rule Bean, a program simply needs to implement a trigger point; this trigger point interfaces with the Business Rule Bean framework to execute the business rule that is encapsulated within a Business Rule Bean. Programming a unique new rule implementation in Java is usually a simple process, made easier by the set of predefined rule implementors that can be used to create a customized business rule.

Business Rule Beans

A business rule is a statement that defines or constrains some aspect of a business by asserting control over some behavior of that business. A business rule officiates over frequently changing business practices, and can come from government regulations, company practices, customer status, or other external factors such as adapting business processes to react to competitive pressure. These types of rules can change periodically, and do not require a rebuild of all applications that support the business process. By externalizing rule processing, the rule can change without affecting the business process. At its simplest level, a business rule is little more than a well-placed if/then statement that compares a variable against a determined value, and then issues a command when they match.

Why use Business Rule Beans?

Business Rule Beans provide a framework and a tool that facilitates externalizing business rules from the core of the business applications. Externalizing the rapidly changing laws that governs the business process allows decoupling of the rules from the process application. So at any point in time, when performing any change to the business process rules, one does not need to rebuild the whole business application. Moreover, this separation leads to reduced maintenance and testing costs of the application code, since any rule change would require only maintaining and testing the rule itself, and not the whole business

application. Rule maintenance is not limited to modifying existing business rules; it also incorporates the introduction of new rules. Moreover, changes to business rules can be made ahead of time and scheduled to take effect automatically at a specific time.

The Business Rule Beans framework provides a Rule Management tool that has a very simple administrative GUI. It can be used by business analysts who are not IT professionals to configure existing rules, by changing variable business rule data. Let's say for example that an insurance company grants a 10% discount to customers who are more than 60 years old. At some point in time, the company's policy may change the discount percentage. This can be incorporated in the system in a very simple manner. The business analyst needs to use the Rule Management Application and change only the value of the discount percentage.

Business Rule Beans framework not only decouples the business code from the application code, but also decouples user roles within an organization. In any organization applying the Business Rule Beans framework, the following three roles need to exist:

- Rule-based Application Component Developer: Provides EJBs and servlets that call out to business rules when appropriate. The component developer is conceptually unaware of the specific implementation that is going to be associated with the rule.
- Rule Implementation Developer: Creates the Java classes that provide the concrete implementation of the business rules.
- Domain Expert (Business Analyst): Determines the values for the business rules, such as the start and end dates for using the rule, the rule's initialization parameters, and the business intent.

This kind of working arrangement is advantageous because these individuals are able to work in parallel. The rule implementation developer may create a rule without needing to know the values that are to be contained inside it, and the analyst can modify it knowing the rule name and the folder it is located in, using the Rule Management Application.

Business Rules can be utilized in virtually every application domain. Some obvious applications include personalization of Web sites, where you could configure rules to adapt the behavior of the site depending on the particular client who is accessing it. You can determine the user classification based on previous access duration to the Web site, and then determine the Web site content based on the user classification. The insurance and banking industries, which are characterized by complex and mutating business requirements, can also make broad use of the rules. For example, the car insurance rate can be based on a customer's age, miles driven per month, or number of accidents within a certain

durations. In banking applications, service charges can be determined based on a customer's classification as determined by the amount of investment by that customer.

The following is a detailed client scenario for using Business Rule Beans. Consider a Web banking site that has a business model that places customers in one of a number of categories, depending on their average daily balance, their average daily activity, and their exception history. Depending on these items, a customer could be in the Risk, General, Gold, or Silver category. The rules that determine this categorization are in a state of nearly continuous adjustment, as the economy changes, government regulations change, the financial affairs of the bank's customers change, and as competitive pressure is brought to bear on the bank. The category into which a customer is placed controls what fees they are required to pay, how many free checks they can write in a month, and the interest rate they earn on their balance. This system could easily be implemented using the Business Rule Beans framework to manage and control both the categorization and the decisions that are contingent upon categories. A business analyst could relatively easily learn to manage the rules that govern this categorization, and as they change, could modify them so that developer skills can be directed toward more productive activities. A default administrative view is provided, or this can be done even more effectively through custom applications that are written to the Java/J2EE APIs that the Business Rule Beans framework supports, thereby providing full access to the rules that are used.

This bank likes to periodically motivate its customers to try to get into the next higher category. To do that, they run month-long promotions where customers who maintain some average daily balance (lower than would normally be required for the next category) are automatically placed into the higher category for a trial period of some number of additional months. After that period, if they've met the criteria for that category, they get to stay. After the month-long promotion, the rules go back to their normal condition. Since the rules are implemented using the Business Rule Beans framework, they can easily be overridden for a given period of time and rendered ineffective at a later point of time. At that time, the original rules come back into effect. The override rules remain in the system, to be used again when it is time for the next promotional activity. Since the rule data has been externalized, the override rules can simply be copied from the base rules, and such values as the minimum average daily balance for the category can be easily and quickly changed to reflect the new temporary business requirements.

10.2 Design

WebSphere Application Server Enterprise V5 provides a framework for Business Rule Beans. In this section, we study this framework and how the components of this framework interact with each other.

10.2.1 Business Rule Beans framework

The Business Rule Beans framework facilitates the externalizing of changeable business logic by providing the following components:

- Business Rule Beans EJBs, which provide a runtime environment for the framework.

  During the development phase, they persist business rule configurations. At runtime, Business Rule Beans EJBs are responsible for finding and firing Business Rule Beans rules. Business Rule Beans EJBs must be deployed with the application that uses them.

- Business Rule Beans Trigger Point framework, which provides an interface to access Business Rule Beans rules from the rule client (application code that uses a Business Rule Beans rule).

  Trigger points are pieces of application code that make a call to a rule. They are placed in EJBs, servlets, or Java classes that depend on business logic implemented in Business Rule Beans rules. The rule client creates an instance of a TriggerPoint object and calls one of its trigger methods, which looks up Business Rule Beans rules by names defined in the Rule Management Application. Once the rule is found, the TriggerPoint object invokes the rule by calling the fire method on the RuleImplementor (a Java class that implements a business rule).

- Business Rule Beans Rule Implementors, which are Java classes that provide implementation for common business logic.

  These rule implementors are ready for use by rule clients through the TriggerPoint objects described above.

- A starter set library of generic rules, for example GreaterThan, IsNull, and Range written in Java, that can be used for building customized rules.

- An interactive Rule Management Application to create, retrieve, update, and delete rules.

  It is a stand-alone tool for maintaining the rules. You can use it to configure the rule name, implementation class, start and end date, initialization parameters, etc.

- A batch eXtended Markup Language (XML) based rule loader/unloader to make rules portable.

  This is the Business Rule Beans Rule Importer and Exporter toll, which allows you to import and export Business Rule Beans rule configuration as XML files. It can be used to move Business Rule Beans rule configuration between servers.

10.2.2 Architecture

The described components are used for both during the development phase and at runtime. At runtime, the rule client invokes Business Rule Beans rules using the TriggerPoint object. Each Business Rule Beans rule configured in the Rule Management Application is represented as an EJB at runtime. After the Business Rule Beans rule is located, the framework fires the actual implementation and returns results to the rule client. Figure 10-1 shows the interaction of Business Rule Beans components.

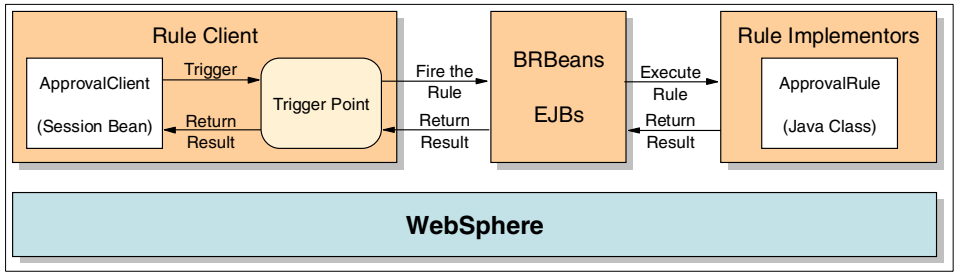


Figure 10-1 Runtime interaction of Business Rule Beans components

The runtime code that is used to find and trigger rules is made up of two parts:

1. The part that runs on the client, that is the rule client component. It can be any EJB, servlet, or Java class in your enterprise application that is dependent on the rule business logic. You can place the TriggerPoint in any of these components to use your Business Rule Beans rule. The client code is used to do the following:
   - Find a specified rule.
   - Determine where the rule should be triggered. Refer to 10.8.2, “Rule firing location” on page 453 for more details about rule firing locations.
   - Call the fire method on all the rules
   - Combine the results from the rules.

2. The part that runs on the server consists of the EJBs used to represent rules and rule folders. These EJBs do the following:
   - Provide for business rule persistence.
   - Provide query functions that the client part of the runtime can use to find rules to be triggered.

The objects used to implement a business rule contain methods and attributes used by Business Rule Beans runtime and/or its administrative component. An externalized business rule is implemented as a pair of objects:

- Rule: This is an Entity EJB that stores all the persistent data for the business rule. This is the object that the trigger point framework code performs a query to find the Rule object(s) representing the business rules to be triggered. Once the Rules are found, the framework code determines where the Rule is to be invoked, either local to the trigger point or remotely on the application server. It then invokes the fire method on either the Rule EJB itself or on the local copy of the EJB to perform the function of the business rule.
- RuleImplementor: The class name of the business rule's RuleImplementor is stored persistently in the Rule. The RuleImplementor is a transient object (not managed by the application server) that the Rule instantiates and then uses to do the actual work. When the fire() method is called on the Rule object, the Rule object combines its persistent set of values with the parameters it received on invocation to create the parameter list for the parameter list. The actual execution of the RuleImplementor algorithm can take place either remotely or locally.

The Business Rule Beans framework uses a database to persist Business Rule Beans rule data. You can create a separate database to maintain rule data or create tables in the main database used by the application. All EJBs accessed in the same transaction must specify the same isolation level. Using a different database for business rules removes this restriction, because application EJBs and rule EJBs are accessed in different transactions. Having separate databases also follows the concept of externalizing the business rules from the enterprise application. Moreover, having the rules data separate from the application data facilitates the reuse of the existing business rules with other applications. In our sample application, we used a separate database to maintain business rule data.

There are two kinds of rules in WebSphere. The difference between the two is mainly in the return value.

- A classifier is capable of accepting a set of input values, and returning a set of values that enumerates what something with those characteristics is. These rules are very useful for such things as controlling process flows, implementing systems that are event driven, and other more advanced system architectures. A classifier rule returns a String that represents the

classification. Typically, the values that the String can assume belong to a limited set (Gold, Silver, Platinum for customer status, for instance).

- A non-classifier or base rule is the most generic kind of rule. It processes the input parameters and returns an array of objects. For more details about the different types of business rules, refer to the WebSphere Application Server Enterprise V5 InfoCenter.

10.3 Development

In order to develop an enterprise application the uses Business Rule Beans, three activities take place during the development process (see Figure 10-2):

- Rule implementors are created. This step is optional. Application developers can use Business Rule Beans rule implementors provided with the framework.
- Trigger points are placed in the rule client.
- Business rules are configured using the Rule Management Application. The tool uses Business Rule Beans EJBs to save Business Rule Beans rule configuration information. This condition implies that an application server must be started prior to using the tool.

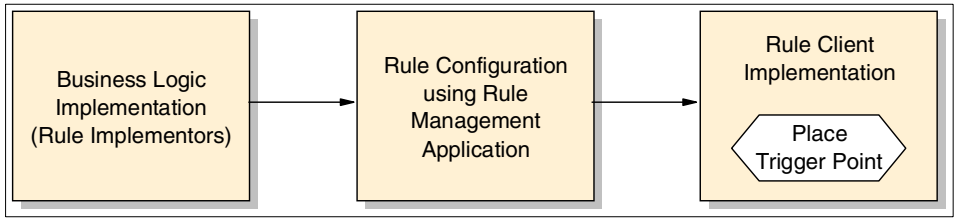


Figure 10-2 Business Rule Beans components during the development phase

10.3.1 Development environment setup

In developing Business Rule Beans based applications, you need to start by setting up WebSphere Studio IE to develop and run applications using Business Rule Beans. This section assumes that DB2 is the DBMS used.

Create a project containing Business Rule Beans code

The following are the steps for setting up the environment for developing Business Rule Beans:

1. From the J2EE perspective main menu, select **File ->New -> EJB Project**.
   a. Select **Create 2.0 EJB Project**.

b. Click **Next**.

c. Specify a project name, `ACompanyBrbEJB`. Select **Use default** for the EJB project.

d. Specify an EAR project name, `ACompanyBrbEAR`. Select **Use default** for the EAR project.

e. Click **Finish**.

2. Now that the project is created, you need to add the necessary JAR files for Business Rule Beans support to the project Java build path. Right-click the new **ACompanyBrbEJB** project and select **Properties** from the menu.

   a. In the properties for ACompanyBrbEJB window, select **Java Build Path**. Click the **Libraries** tab. Click the **Add Variable** button, and select **WAS_V5_XERCES**. Click **OK**.

   b. Click the **Add Variable** button and select **WAS_EE_V5**. Click **Extend**. In the variable Extension window, expand the **lib** directory, and select **brbClient.jar**, **brbServer.jar**, **brbRuleMgmtApp.jar**, and **distexcep.jar**. Click **OK**.

   c. Click **OK**.

3. Add the BRBeansDB2 JAR file to the ACompanyBrbEJB project.

   a. Select **File -> Import**, and select **EJB JAR file**. Click **Next**.

   b. Browse and locate the **BRBeansDB2.jar** file to import. This file can be found in <*WebSphere_Studio_IE_root*>\runtimes\ee_v5\BRBeans\BRBeansDB2.jar.

   c. Choose to import to an existing EJB project, and type in the EJB project name `AcompanyBrbEJB`.

   d. Click **Finish**.

4. Generate the Business Rule Beans code for deployment.

   a. In the J2EE perspective, J2EE Hierarchy view, expand **EJB Modules**, right-click **AcompanyBrbEJB:BRBeansDB2** and select **Generate -> Deploy and RMI Code**.

   b. Make sure three EJBs are selected: **Rule**, **RuleFolder**, and **RuleHelper**, then click **Finish**.

Creating the Business Rule Beans database

This section walks you through the process of creating the Business Rule Beans database. For running Business Rule Beans two tables are needed: Rule and RuleFolder, which correspond to the Entity EJBs.

1. In the J2EE perspective, J2EE Hierarchy view, under EJB Modules, select **ACompanyBrbEJB:BRBeansDB2**. Right-click and select **Generate -> Schema DDL**. This generates the Table.ddl.

   Note: The database script can be found under the database directory under the name of brbDB.ddl.

2. Open a DB2 command window by selecting **Start -> Programs -> IBM DB2 -> Command Window**. Enter the command **`db2 create database brb`**. This will create a database called brb to be used by the Business Rule Beans.
3. Connect to the database and grant access to the application database user:

   ```
   db2 connect to brb
   db2 grant dbadm on database to user dbuser
   db2 disconnect current
   ```

4. Connect to brb database that you have just created using the command:

   ```
   db2 connect to brb user <db_username> using <db_password>.
   ```

5. Change directory to <*WebSphere_Studio_IE_root*>\Workspace\ACompanyBrbEJB\ejbModule\META-INF, where <*WebSphere_Studio_IE_root*>\Workspace is the directory containing your WebSphere Studio IE workspace, so this may be a different directory on your machine.
6. Enter the command **`db2 -tf Table.ddl`**. You should now have tables for the Rule and RuleFolder Entity EJBs.

Configure Rule Management Application

The provided Rule Management Application is used to create and configure rules. In order to be able to run the Rule Management Application from WebSphere Studio IE, you need to do the following:

1. Go to the Java perspective, and select the Package Explorer view. Expand the Business Rule Beans EJB project **ACompanyBrbEJB**, and expand the JAR file **WAS_EE_V5/lib/brbRuleMgmtApp.jar**.
2. Expand the package **com.ibm.ws.brb.rm.ui**, and select the **RuleManagement.class**.
3. From the main menu of WebSphere Studio IE, select **Run -> Run...**, the Launch Configurations window will open. Select **Java Application** and click the **New** button at the bottom of the window. This should add a new Java application to the list named RuleManagement as shown in Figure 10-3 on page 429. Verify that in the Main tab, the Project is ACompanyBrbEJB, and the Main class is com.ibm.ws.brb.rm.ui.RuleManagement.

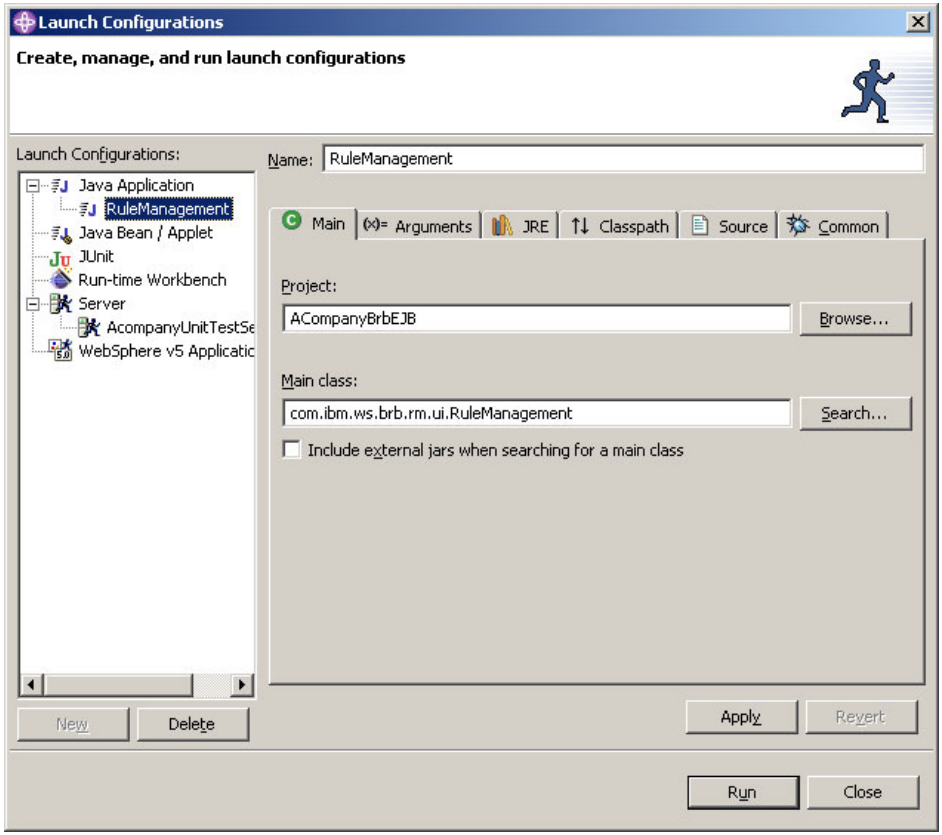


Figure 10-3 Java application configuration for Rule Management Application

4. Select the **Arguments** tab, and specify the following for the VM arguments: "-DbrbPropertiesFile=<*WebSphere_Studio_IE_root*>\runtimes\ee_v5\bin\brbeansDefaultProperties", where <*WebSphere_Studio_IE_root*> is the name of the directory where WebSphere Studio Application Developer Integration Edition is installed on your machine. If there are spaces in the path name, you should put double quotes (") around the entire property as shown here.

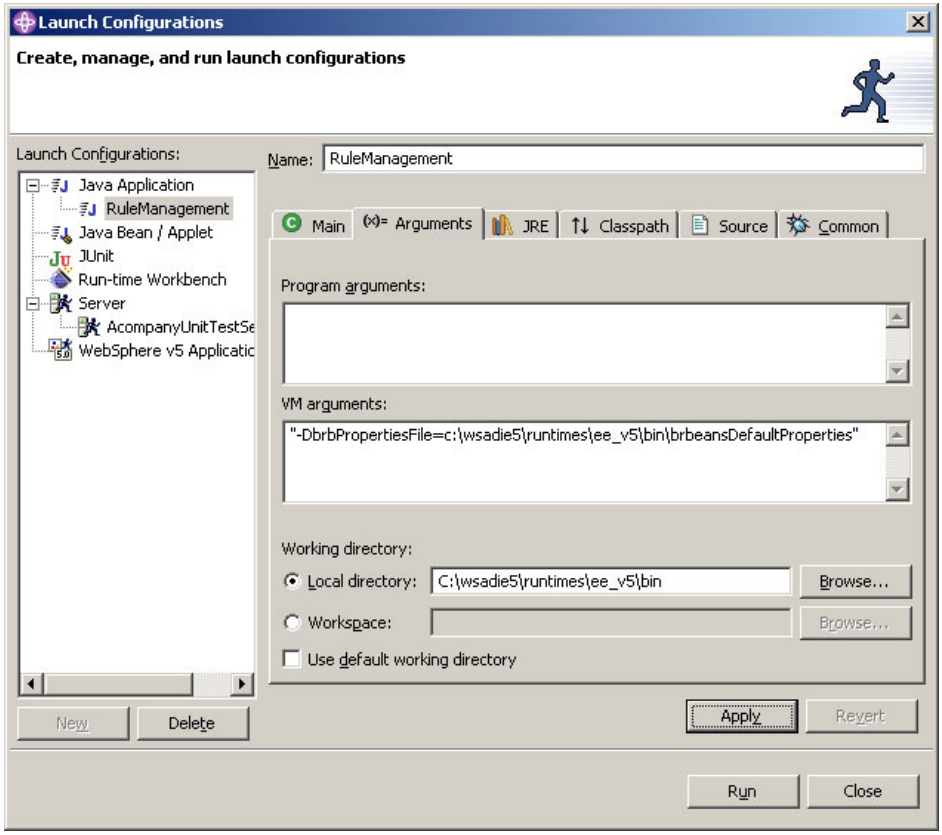


Figure 10-4 Rule Management Application arguments

Important: The brbeansDefaultProperties file contains WebSphere server host and port information, as well as the JNDI names of Business Rule Beans EJBs. If you are accessing the WebSphere server remotely, modify the host and port information before using the Rule Management Application. Note also that you can make a copy of the original brbeansDefaultProperties file, and give it a name of your choice, such as brbeansMyAppProperties, do the necessary changes to that file, and use it in your application's configuration instead of the default one. Example 10-1 shows the contents of the brbDefaultProperties file.

Example 10-1 brbDefaultProperties file

```
host=localhost
port=2809
RuleJndi=brbeans/application/Rule
```

```
RuleFolderJndi=brbeans/application/RuleFolder
RuleHelperJndi=brbeans/application/RuleHelper
```

5. Under Working directory, uncheck **Use default working directory**, and select **Local directory**. For Local directory, specify <*WebSphere_Studio_IE_root*>\runtimes\ee_v5\bin.
6. Click the **JRE** tab. Select the **WebSphere v5 EE JRE**.

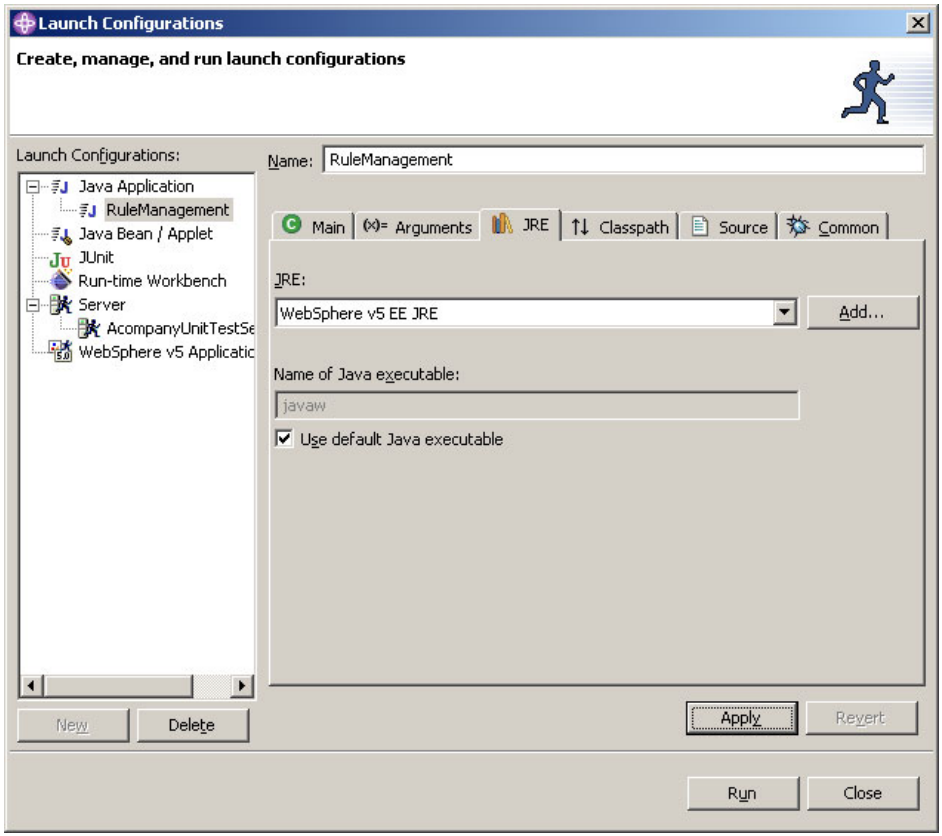


Figure 10-5 Rule Management Application JRE

7. Click the **Classpath** tab. Uncheck **Use default class path** at the bottom of the tab. This allows you to add entries to the classpath.
   a. Click **Add External JARs** button. Go to the <*WebSphere_Studio_IE_root*>\runtimes\ee_v5\lib directory and select the file **namingclient.jar**. Click **Open**. Namingclient.jar should be added to the classpath.

b. Click the **Advanced** button. Select **Add External Folder** and click **OK**. Select the <*WebSphere_Studio_IE_root*>\runtimes\ee_v5\properties directory. Click **OK**. The properties directory should be added to the classpath.

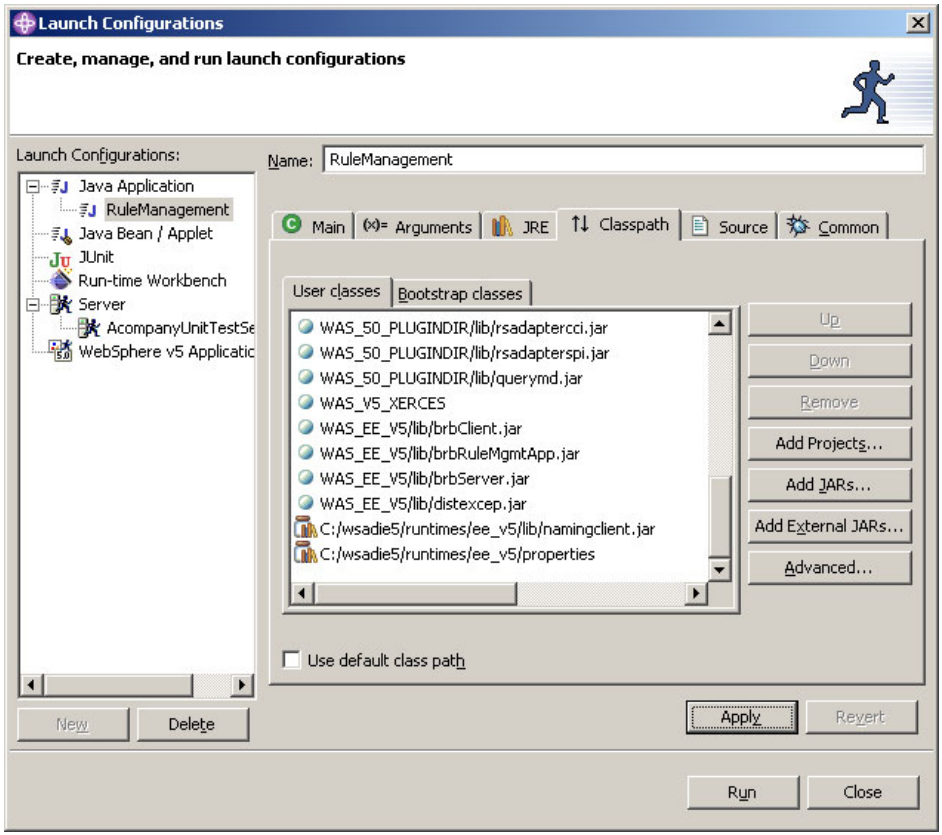


Figure 10-6 Rule Management Application classpath

8. Click **Apply** to save the changes to the RuleManagement Java application. Click **Close**.

Configure application server

Now that you have performed the Rule Management Application setup, you need to start the application server before running the Rule Management Application with ACompanyBrbEJB project. The following steps show how to set up and start the ACompanyUnitTestServer to be able to run the Rule Management application, and then run the application.

1. Switch to the Server perspective. In the Server Configuration view, expand **Servers**, and double-click **ACompanyUnitTestServer**. In the editor, select the **Environment** tab, click the **Add** button in the System Properties section. Enter brbPropertiesFile as the name, and for the value enter <*WebSphere_Studio_IE_root*>\runtimes\ee_v5\bin\brbeansDefaultProperties where <*WebSphere_Studio_IE_root*> is the name of the directory where WebSphere Studio Application Developer Integration Edition is installed on your machine.
2. Create a data source on the application server to be used by the Business Rule Beans and the Rule Management Application.
   c. Select the **Security** tab at the bottom of the view. In the JAAS Authentication Entries section click the **Add** button. In the JAAS Authentication Entry window, type `ACompanyBrbAlias` for Alias, and then type your DB2 user ID and password in the User ID and Password fields. Click **OK**.
   d. In the editor for the ACompanyUnitTestServer, select the **Data Source** tab at the bottom of the view.
      i. Under Server Settings, in the JDBC provider list select **DB2 JDBC Provider (XA)** whose implementation class is COM.ibm.db2.jdbc.DB2XADataSource. Click the **Add** button to the right of Data source defined in the JDBC provider select above.
      ii. In the Create a Data Source window select **DB2 JDBC Provider** and select **Version 5.0 data source**. Click **Next**. Specify `BRBeansDS` for the Name. Specify `jdbc/BRBeansDS` for the JNDI name. For Container-managed authentication alias, select **ACompanyBrbAlias**. Click **Next**.
      iii. Select the **databaseName** property. For the value of this property specify the name of your database, `BRB`. Click **Finish**.
   e. Save the file ACompanyUnitTestServer and close it.
3. Specify to use the data source.
   a. In the J2EE perspective, J2EE Navigator view, expand ACompanyBrbEJB to ejbModule/META-INF/ejb-jar.xml. Right-click ejb-jar.xml and select **Open with -> Deployment Descriptor Editor**.
   b. Select the **Overview** tab. Scroll down to the WebSphere Bindings section and find JNDI - CMP Factory Connection Binding. For the JNDI name, specify `eis/jdbc/BRBeansDS_CMP`. Note that this is the JNDI name of the J2C connection factory that was automatically created when you created the 5.0 data source.
   c. Save and close the file.

4. In order to be able to start the Rule Management Application with the security enabled, you need to grant the RuleManager and RuleUser roles to everyone.
   a. In the J2EE perspective, Hierarchy view, expand Enterprise Applications and select **ACompanyBrbEAR**, right-click and select **Open with -> Deployment Descriptor Editor**.
   b. Select the **Security** tab as shown in Figure 10-7. Click the **Gather** button, and three security roles will show up in the list. Select each of the RuleManager and the RuleUser roles and in the WebSphere Bindings section, select the **Everyone** check box.

      Note: Granting the roles RuleManager and RuleUser is done here only for development purposes. In a runtime environment, the roles are mapped to the users and groups according to the security policy.

   c. Save and close.

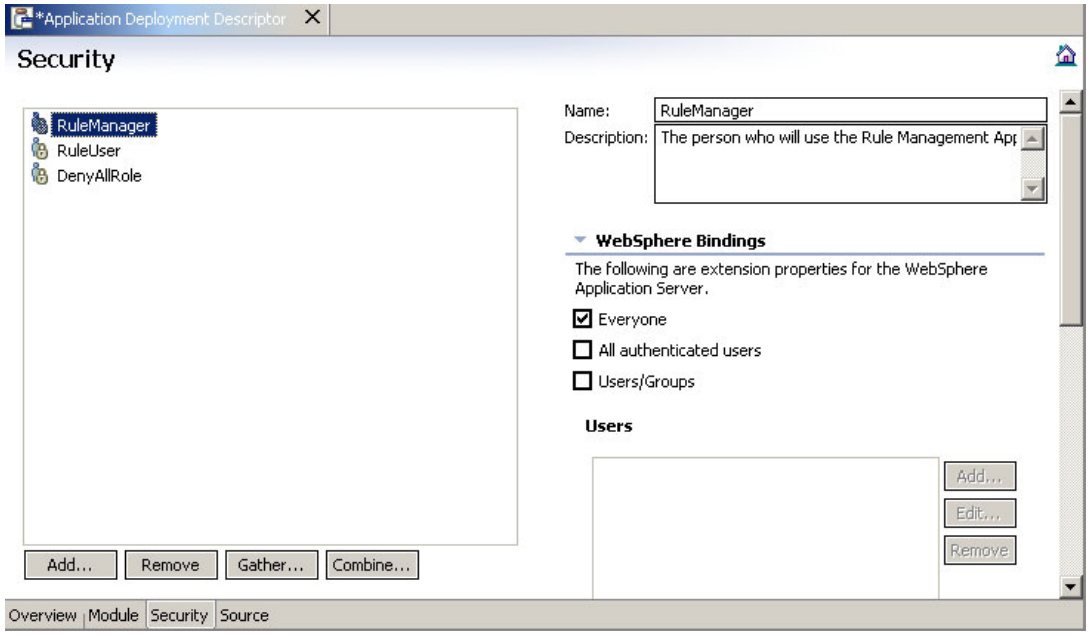


Figure 10-7 Business Rule Beans security roles

5. Add the ACompanyBrb project to the sample application server. In the Server perspective, Server Configuration view, under Servers, right-click **ACompanyUnitTestServer**, and select **Add -> ACompanyBrbEAR**.

10.3.2 Creating the rule implementor

Rule implementors are Java classes that encapsulate business logic. These classes must implement the RuleImplemetor interface, which has three methods:

- init()

  This method provides the initial state to the rule implementor. The method is called by the framework when a Business Rule Beans rule is first fired, just like a servlet's init method that is invoked the first time a servlet is run. Rule implementor initialization parameters are configured using the Rule Management Application. For more details, see 10.3.3, "Creating and configuring the rule" on page 436.

- fire():

  This method implements business logic of the rule and returns results of an algorithm to the caller. The TriggerPoint object invokes the fire method from the rule client, passing parameters expected by the rule implementation.

- getDescription():

  This method returns a string containing a description of the RuleImplementor.

Example 10-2 Signatures of RuleImplementor methods

```
public Object fire(TriggerPoint arg0, Object arg1,
                IRuleCopy arg2, Object[] arg3)
                   throws BusinessRuleBeansException
public String getDescription()
public void init(Object[] arg0, String[] arg1,
          String arg2, IRuleCopy arg3)
             throws BusinessRuleBeansException
```

Example 10-2 shows the signatures of the methods that a RuleImplementor class needs to implement. In order to create the RuleImplementor:

1. In the J2EE perspective, J2EE Navigator view, select and expand the **ACompanyBrbEJB**. Select the **ejbModule** and right-click. Create a new package named, for example, `com.acompany.brb.rules`.
2. Create a new class named, for example, `ApprovalRule`, in the package com.acompany.brb.rules. Let the class implement the RuleImplementor interface.
3. Implement the init method. The init method needs to pick up the rule initialization parameters. In this sample application, the rule parameter that is needed for initialization is the total amount of the purchase order, above which an approval is required, which is called in our sample approvalLimit.
4. Implement the fire method. The fire method as mentioned earlier contains the rule business logic. In case of this sample application, the fire method

compares the total purchase order amount with the approval limit amount. If it is greater than the limit, then an approval is required; otherwise no approval is needed.

For more details about the methods implementation, have a look at the sample code provided in Appendix B, “Sample scenario” on page 665.

10.3.3 Creating and configuring the rule

The Rule Management Application is used to configure and maintain Business Rule Beans rules. The tool gives the ability to perform the following tasks:

- Create a new Business Rule Beans rule definition, specifying:
  - Business Rule Beans rule folder and name (used to look up the Business Rule Beans rule by the TriggerPoint object)
  - Business Rule Beans rule start and end date.
  - Business Rule Beans rule type: Classifier, non-classifier, or classified.
  - Business Rule Beans rule implementor Java class.
  - Initialization parameters for the rule implementor.
  - Dependent Business Rule Beans rules.
- Modify a Business Rule Beans rule definition.
- Export or import Business Rule Beans rule definitions, which are created in XML format.
- Quick copy action, which allows you to replace existing Business Rule Beans rules with a new one on a specified date. The Quick Copy function creates a copy of the existing rule and allows you to change the Business Rule Beans rule start date and the initialization parameters. Once the copy of the Business Rule Beans rule is created, the end date of the original rule is set to the start date of the new rule.

To order to create and configure rules during the development process using WebSphere Studio IE, you need to do the following:

1. Start the server. In the Servers view, select the **Servers** tab at the bottom. Right-click the Server Instance **ACompanyUnitTestServer** and select **Start**. This should switch you to the Console view automatically. If not, select the **Console** view tab at the bottom. Wait for the message `Server server1 open for e-business`. The test environment should now be started.
2. Switch to the Java perspective. Click the **Run** button. The Rule Management Application GUI should come up. Note that when you want to run the Rule Management Application again, you can simply go to the Java perspective,

select **Run -> Run...** from the main menu, select **RuleManagement**, and click **Run**.

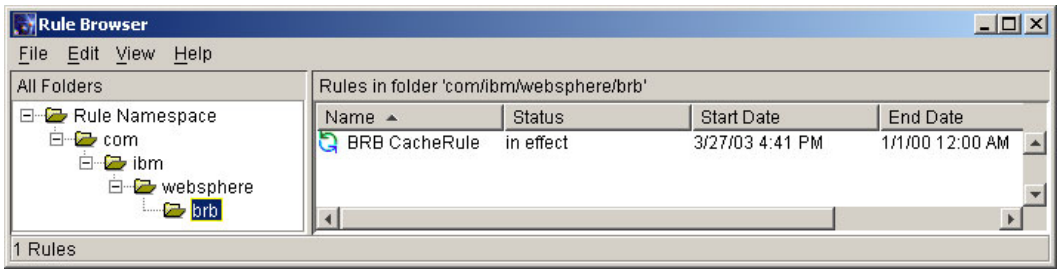


Figure 10-8 Rule Management Application GUI

Expanding the Rule name space as shown in Figure 10-8, you would find that there is an already existing rule shipped with Business Rule Beans. This is the Business Rule Beans CacheRule. For more details about that rule, refer to 10.8, “Performance considerations” on page 451.

3. Create a new folder structure for the new rule to be created. It is recommended that you follow the Java package naming convention when creating folders for your rules. For our sample application, the folder structure is com/acompany/brb/rules. To create a rule folder, select the folder where you want the new folder to be nested. From the main menu, select **File -> New -> Folder**. A new folder appears in the folder hierarchy in edit mode. Enter a folder name and press the Enter key.
4. Create a new rule ApprovalRule. In the rule browser window, select the folder where you want the new rule to be created. From the main menu, select **File -> New -> Rule**.
   a. In the New Rule properties window, in the General tab enter the general information about the rule, such as its name, start date, classification,etc., as shown in Figure 10-9 on page 438.

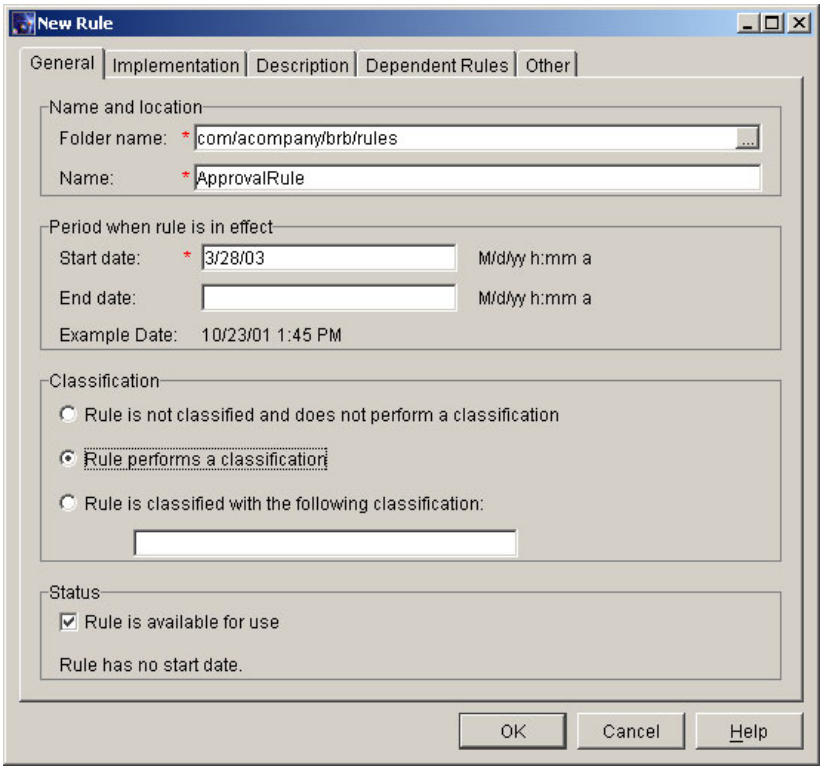


Figure 10-9 New Rule creation

b. Use the Implementation tab to define the manner in which the rule is implemented. You should specify the fully qualified name of the Rule Implementor class, the rule firing location, and the rule initialization parameters. To specify the rule initialization parameters, select the **Add** button in the Initialization parameter section. Enter the Description, Type, and Value as shown in Figure 10-11 on page 439.

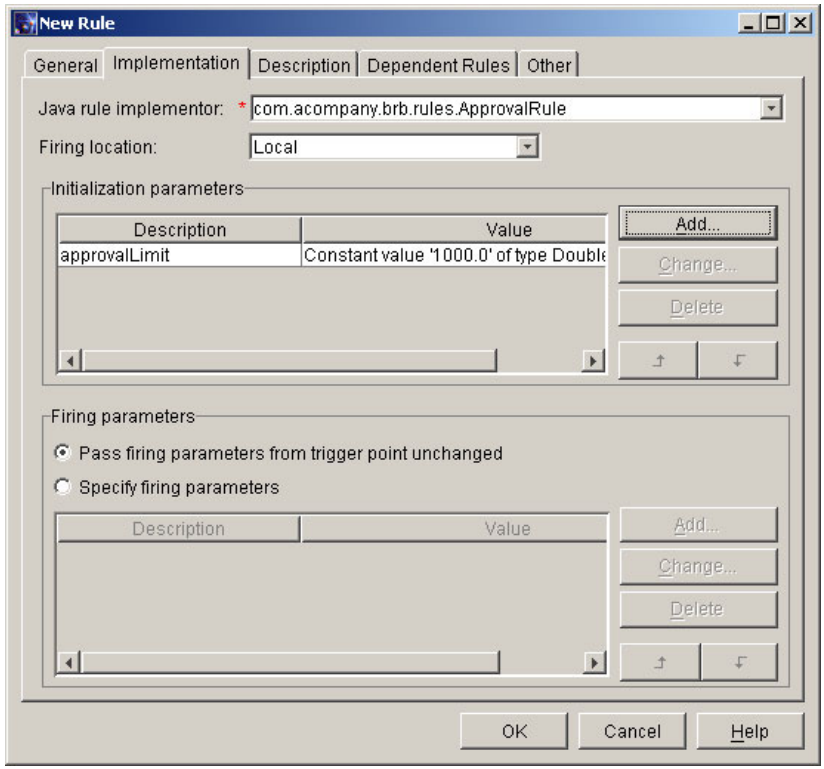


Figure 10-10 New Rule Implementation information

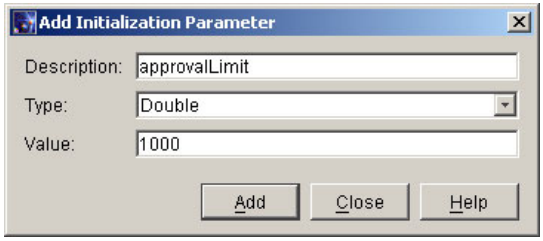


Figure 10-11 Rule initialization parameters

c. Click **OK**.

The Description tab is used to define the purpose and intent of the rule. The Dependent Rules tab can be used to specify the rules that the newly created rule will depend on. Finally the Other tab is used to establish precedence, and enter information that is relevant to you but does not fit into any other category.

Once a business rule is created among the properties, they describe its state. This determines whether a rule can be used by the rule client at runtime or not. Table 10-1 shows the possible states of a business rule, and what each means. Rule states can be modified using the Rule Management Application.

Table 10-1 Business rules states

State	Description
In effect	The rule is available for use. A rule must be in this state in order to be fired from the rule client.
Schedules	The rule start date is set to some date in the future.
Expired	The rule is no longer active, since the rule end date is in the past.
Invalid	The rule configuration has errors.
Unavailable	The rule is not ready for use.

10.3.4 Creating the rule client

In our sample application, the rule client is a session bean that is used by the Place Order process to check whether an approval is needed for the purchase. To create the rule client session bean and to place a trigger point, do the following:

1. Switch to the J2EE perspective, and select the J2EE Navigator view. Select and expand the **ACompanyBrbEJB** project. Select **ejbModules**, and create a new package named `com.acompany.brb.ejbs`.
2. Create a new session bean named `ApprovalClient`. Open the ApprovalClientBean.java for editing. Add the statement `import com.ibm.websphere.brb.TriggerPoint`.
3. Implement the isApproveRequired method as in Example 10-3.

Example 10-3 Implementation of isApproveRequired - TriggerPoint placement

```
public boolean isApproveRequired (Double total){
   boolean result = false;
   try {
      //Create new trigger point
      TriggerPoint tp = new TriggerPoint();

      //Disable caching - for demonstration purpose
      tp.disableCaching();

      //The rule expects the total purchase order amount
      Object[] firingParams = { total };
      String ruleName = "com/acompany/brb/rules/ApprovalRule";
```

```
      //Call the rule
      Object resultObject = tp.triggerClassifier(
         null, //Target Object-not required
         firingParams, //Rule firing parameters
         ruleName); //The name of the rule to fire

      if (resultObject!=null) {
         //Rule successfully called
         Object[] resultArray = (Object[])resultObject;
         result = ((Boolean) resultArray[0]).booleanValue();
      }
   } catch(Exception e) {
      e.printStackTrace();
   }
   return result;
}
```

4. Promote the method to remote interface.
5. Select **ACompanyBrbEJB**, right-click and select **Generate ->Deploy and RMIC Code**.

10.3.5 Integrating the sample application with the business rule

As shown in Figure 10-12 on page 442, we need to integrate our rule with the sample application by making the Place Order process, and in particular the isApproveRequired subprocess, invoke the ApprovalClient EJB. When the isApproveRequired process invokes the isApproveRequired method of the ApprovalClient, it passes the purchase order total amount as an argument. The ApprovalClient will trigger the ApprovalRule, which is already initialized with an approval limit of $1000.

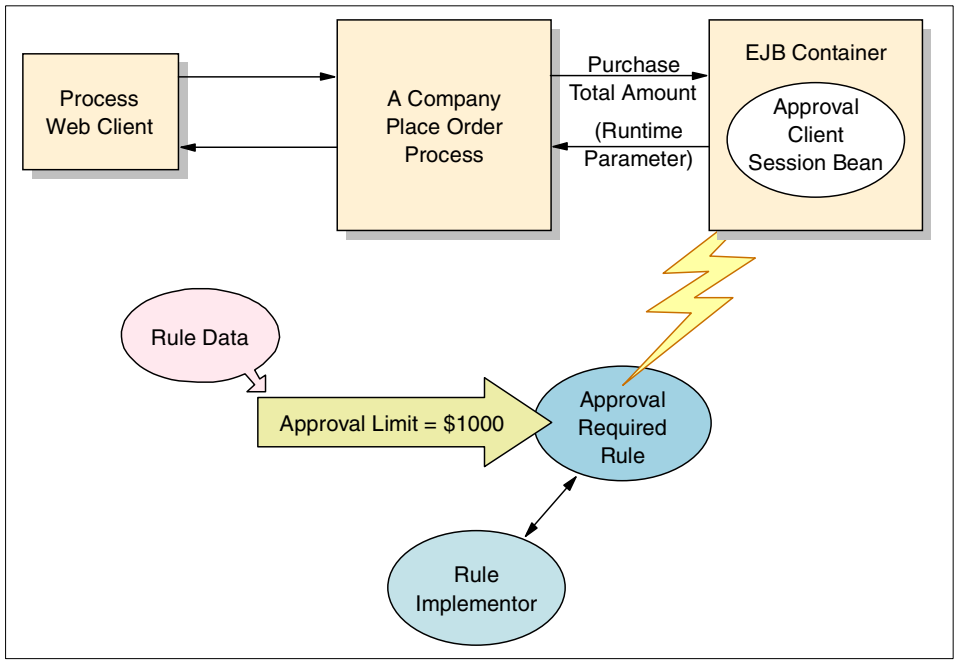


Figure 10-12 Sample application with Business Rule Beans

1. In the J2EE perspective, J2EE Navigator view, expand ACompanyServices and select **PO.process**.

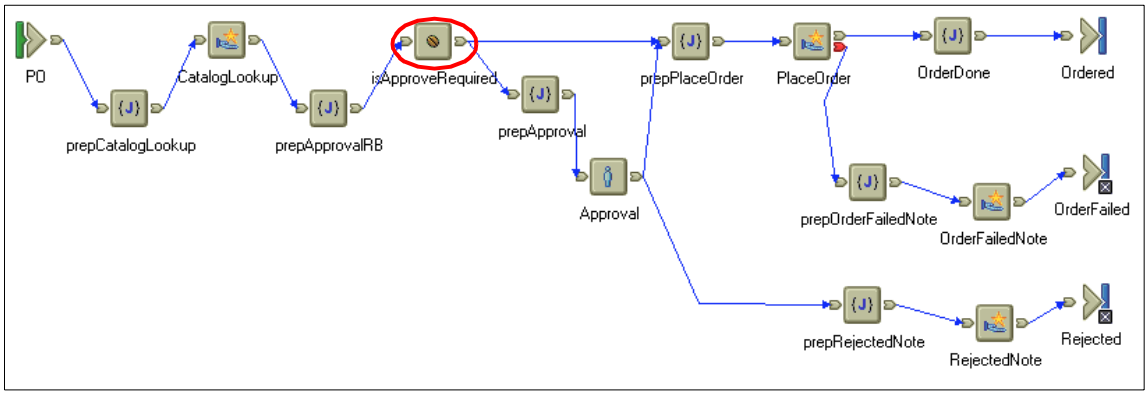


Figure 10-13 Place Order Process

2. Select the **isApproveRequired** process, and right-click and select **Properties**. Select the **Implementation** option.

a. Click the **Browse** button and select the interface representing the rule client, in this case **com.acompany.brb.ejbs.ApprovalClient**.

b. Select the method that you need the process to invoke. In this case it is **isApproveRequired** method.

c. Specify the JNDI name ejb/ApprovalClient.This is the JNDI name for the EJB specified in the EJB description.

d. Click **OK**.

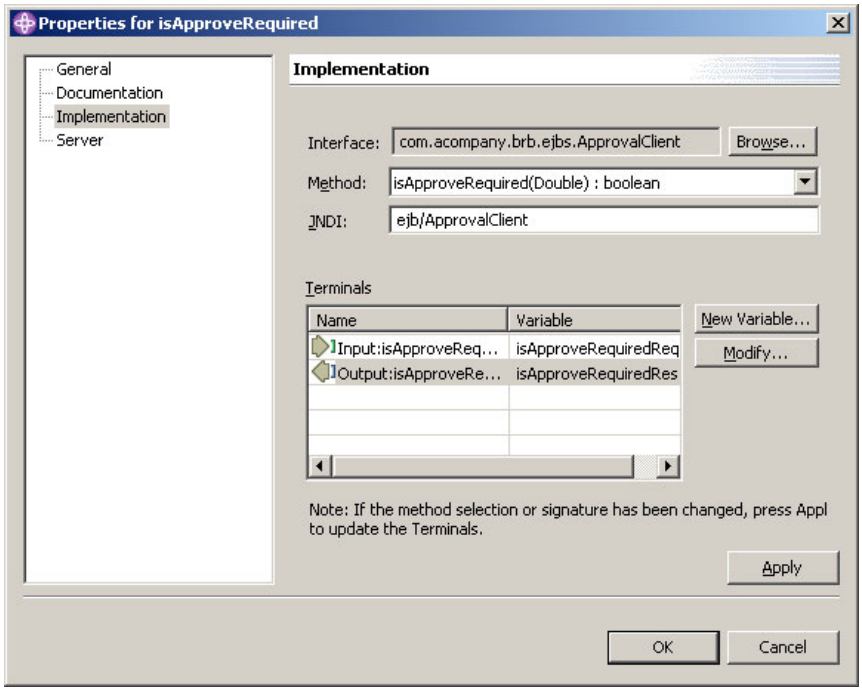


Figure 10-14 Properties for isApproveRequired process

3. Save and close the PO.process.

4. In the J2EE perspective, J2EE Hierarchy view, select **EJB Modules -> ACompanyProcess EJBs**, right-click **Generate Deploy code**. Make sure **POPortTypeService** is selected, and click **Finish**.

5. Switch to the Server perspective, and select **ACompanyUnitTestServer** from the Server tab. Right-click and select **Deploy process**.

10.4 Unit test environment

In this section, we test our business rule and business rule client using the Universal Test Client. Then we test our sample application after the integration of the application with the Business Rule Beans.

10.4.1 Rule unit testing

In order to use the Universal Test to test the rule itself without the sample application, do the following:

1. In the Server perspective, Servers view, select the **Servers** tab. Select **ACompanyUnitTestServer**, right-click and select **Start**.
2. After the server starts successfully, in the same view, select **ACompanyUnitTestServer**, right-click and select **Run universal test client**. IBM Universal Test Client starts, showing its home page.
3. From the home page, select **JNDI Explorer** This opens the JNDI Explorer as shown in Figure 10-15.

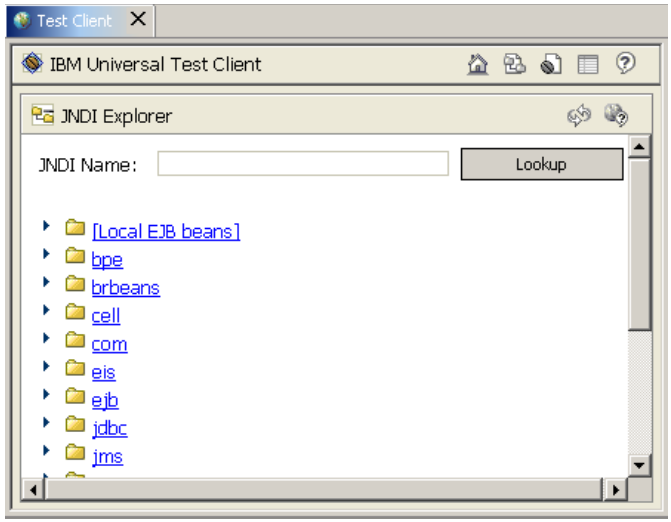


Figure 10-15 Universal Test Client - JNDI Explorer

4. Expand the **ejb** node, and look for the ApprovalClient session bean. Note that the path you have to follow to look for your bean is based on the JNDI name that you have assigned to your session bean. In the sample scenario case, the JNDI name for the bean is ejb/ApprovalClient. That is why you will find the bean directly under the ejb node.

5. Select the **ApprovalClient** entry. This opens the window in Figure 10-16. In the References section on the left, under the EJB References, you will find the ApprovalClient bean. Expand the **ApprovalClient** node; you will find the ApprovalClientHome node, which you will also need to expand. Select **ApprovalClient create()**. Click the **Invoke** button in the Parameters window on the right. Click the **Work with Object** button.

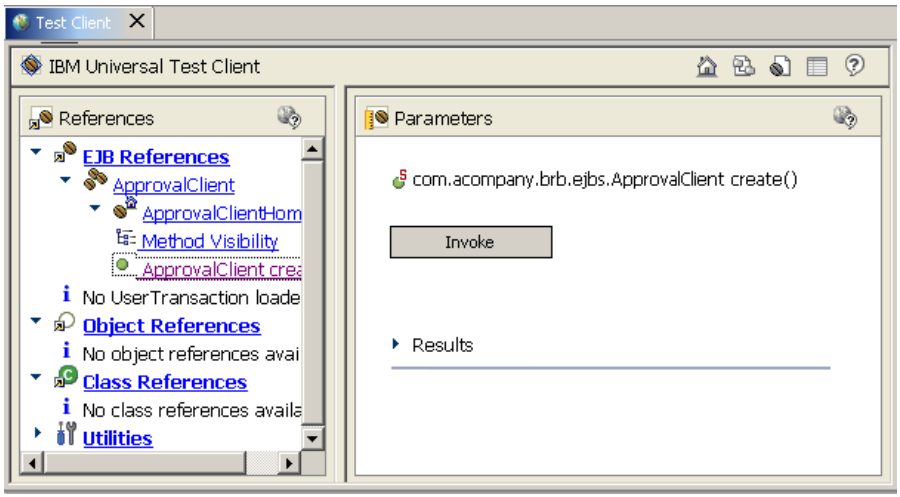


Figure 10-16 Universal Test Client - EJB References

6. Expand the newly added **ApprovalClient 1** bean in the References section. Select the boolean **isApproveRequired(Double)**, which will allow you to invoke the method by entering a value in the Parameters section, and click the **Invoke** button. This will call the isApproveRequired method in the ApprovalClient session bean, which in turn will trigger the ApprovalRule rule, and will return the result whether an approval is required for the test value you've entered.

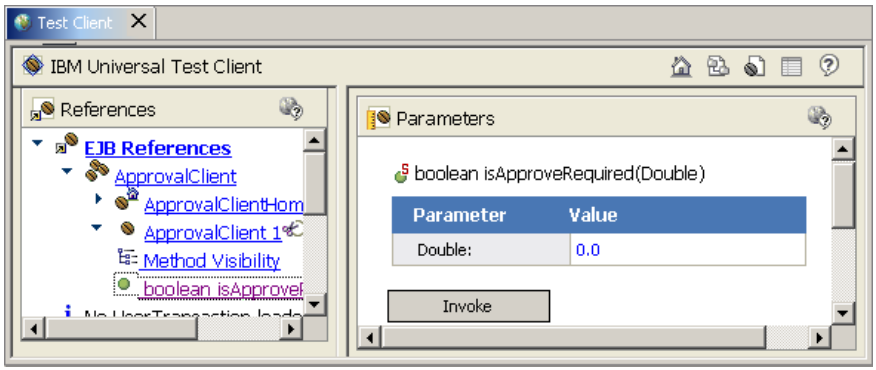


Figure 10-17 Invoking isApproveRequired method

10.4.2 Testing the sample application with the rule beans enabled

This section demonstrates how to test the sample application after the integration of Business Rule Beans. We are assuming the ACompanyUnitTestServer is already started.

1. From the Server perspective, Servers tab, select **ACompanyUnitTestServer**, right-click and select **Run Process WebClient**.
2. You will be requested to enter the administrative user name and password that you have used when you enabled security. Once you do this, the work item manager will open, as shown in Figure 10-18.

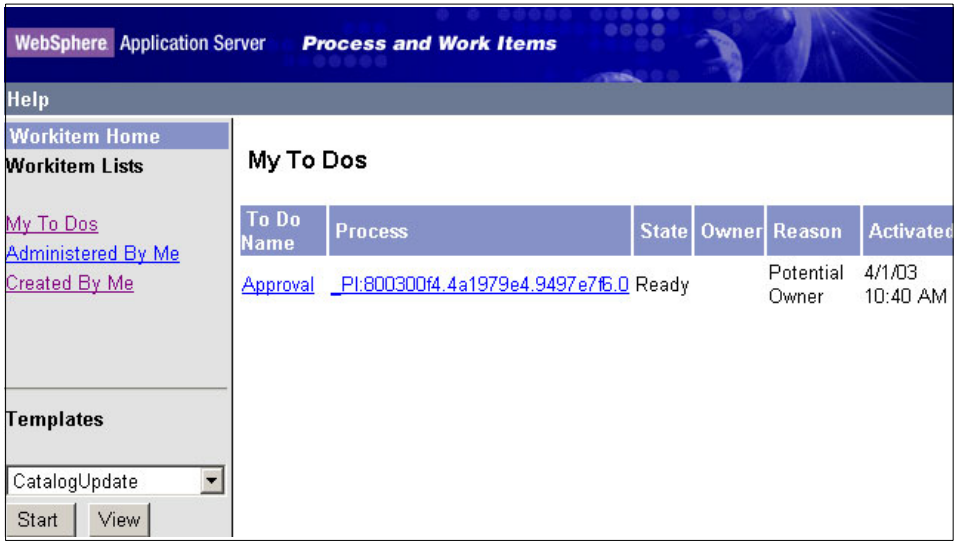


Figure 10-18 Work Item Manager

e. From the Templates combo box on the left, select **PO** and click the **Start** button.

3. Specify the process input message by entering an item ID and the desired quantity, and click the **Start Process** button. This will invoke the PlaceOrder process, which will use the ApprovalRule to decide whether an approval is needed for that order.

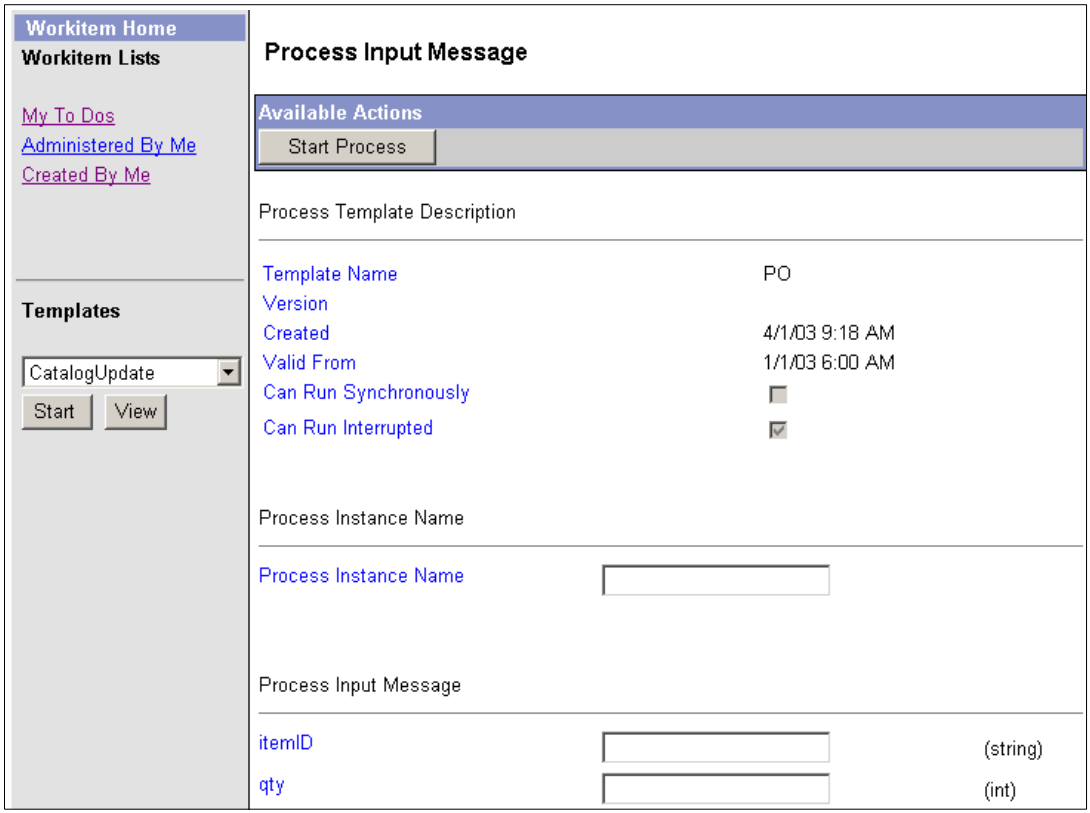


Figure 10-19 PO Process Input Message

10.5 Assembly

After successful completion of testing the sample application integrated with Business Rule Beans, you can import the application's EAR files into the Application Assembly Tool. There are no specific functions for Business Rule Beans in the Application Assembly Tool, but the deployment team can use it to modify JNDI names, data source names, or any references or bindings. Once the

necessary changes are done to the EAR file, it can be exported again from AAT and redeployed in WebSphere.

10.6 Deployment

Business Rule Beans should be deployed as a part of the application EAR file. We completed this step while setting up Business Rule Beans development environment in “Create a project containing Business Rule Beans code” on page 426. As mentioned before, the Business Rule Beans framework uses a database to persist Business Rule Beans rule data. In our case, this is the brb database created in “Creating the Business Rule Beans database” on page 427.

Before starting with the installation of the application, make sure that the ACompanyServer is running. This is the application server on which our sample application is installed. For more details, refer to Appendix B, “Sample scenario” on page 665. Start and log in to the Administrative Console, and perform the following steps:

1. Change the scope to the ACompanyServer in order to create the data source under the server. By default the scope is set to node.
2. Select **DB2 JDBC Provider (XA)**. Select **Data Sources** at the bottom of the window.
3. Click **New** to create a new data source, then provide the following information:
   - Name: `BRBeansDS`
   - JNDI name: `jdbc/BRBeansDS`
   - Check **Use this Data Source in container managed persistence (CMP)**
   - Component-managed Authentication Alias: `dbuser`
   - Container-managed Authentication Alias: `dbuser`

   Click **OK**.

   Note: The chosen alias is the same alias created in Appendix B, “Sample scenario” on page 665 for the sample application. If you want, you may create a separate alias to be used with your Business Rule Beans.

4. Select the newly created **BRBeansDS** entry, then select **Custom Properties** at the bottom of the page.
5. Select the **databaseName** entry and change the value to `brb`.
6. Save the configuration for WebSphere.
7. The next step is to install the ACompanyBrb application. Select **Applications -> Install New Application**.

8. Browse for the ACompanyBrb.ear file, then click **Next**.
9. Navigate through the installation steps. Make sure that you check the **Deploy EJBs** box, and provide the database type DB2UDB_V81 and database schema name EJB.

   Map both Rule and RuleFolder CMP 1.x beans to the BRBeansDS data source.

   Also make sure that your application is installed on the ACompanyServer application server.
10. Once the installation is done, save the configuration.
11. Open the file <*WebSphere_root*>\bin\brbeansDefaultProperties for editing. Check that the proper JNDI names are specified for the Business Rule Beans EJBs (Rule, RuleFolder, and RuleHelper). Check that the proper host name and port numbers are specified. In our sample application, we need to change the port number from 2809 (server1 port number) to 2810 (ACompanyServer port number). Save the changes to the file and close.
12. Select **Servers -> Application Servers** then select the **ACompanyServer** in order to assign the brbPropertiesFile to the server properties. In the ACompanyServer additional properties section, select **Process Definition**, which is used to define the command line information necessary to start/initialize a process.
13. In the Process Definition additional properties, select **Java Virtual Machine -> Custom Properties**.
14. In the custom properties window, click the **New** button to create a new property. Enter the following values:
    - Name: brbPropertiesFile
    - Value: <*WebSphere_root*>\bin\brbeansDefaultProperties
15. Click **OK** and save the WebSphere configuration.
16. Restart ACompanyServer.

10.6.1 Running the Rule Management Application

The Rule Management Application shipped with Business Rule Beans runs outside of any container. Hence the JNDI names need to be specified when this tool is run. The script for this tool requires that a properties file name be passed as a command-line parameter. This name is then specified as the value for the brbPropertiesFile property when the tool is run. To start the application:

1. Make sure that ACompanyServer is started.

Note: If you have created a new application server for the sample, then the port number of the JNDI directory services is different from the default. Open the brbeansDefaultProperties file in a text editor and change the port to the correct number. In our case it is 2810.

2. Open a command prompt and change the current directory to <*WebSphere_root*>\bin.
3. Issue the command **`rulemgmt brbeansDefaultProperties`**. This will start the Rule Management Application. For more details about using this tool, refer to 10.3.3, “Creating and configuring the rule” on page 436.

10.7 Problem determination and troubleshooting

A few common problems running applications using Business Rule Beans are listed below:

- When an application attempts to reference Business Rule Beans EJBs, the code will first look for the brbPropertiesFile Java property. If this property is specified, then the names listed in the file are used to find the EJBs. If the property is not specified, then Business Rule Beans attempts to use the EJB references specified in the container. If the application is not running in a J2EE container, and if the brbPropertiesFile property is not specified, then there is no way to resolve the EJB references. Example 10-4 shows the exception message and stacktrace if such a condition occurs. This is solved by specifying the brbPropertiesFile property.

Example 10-4 Sample trace showing error with the brbPropertiesFile

```
Message:
   The jndi name for the Rule EJB was not found. The possible cause: The
application client is not in a container (ex: client container, servlet, EJB),
a properties file was not specified with the java -DbrbPropertiesFile option,
and the default Rule jndi name of com/ibm/ws/brb/Rule was not found.
Stack trace:
com.ibm.websphere.brb.BusinessRuleBeansException
   at com.ibm.ws.brb.Helper.getRuleHome(Helper.java:778)
   at com.ibm.ws.brb.LocalRuleManager.<init>(LocalRuleManager.java:98)
   at com.ibm.websphere.brb.TriggerPoint.<init>(TriggerPoint.java:251)
   at com.ibm.websphere.brb.TriggerPoint.<init>(TriggerPoint.java:188)
   at
com.acompany.brb.ejbs.ApprovalClientBean.isApproveRequired(ApprovalClientBean.j
ava:48)
```

- The host name and port number used to access the name server can also be set in this properties file.
  - You might get a NameNotFound exception for your Business Rule Beans even though your JNDI names are specified correctly. This might be due to a mismatch between the port number specified in the brbproperties files and the server port number. By default in the brbeansPropertiesFile the port number is set to server1 port number, which is 2809. In our sample application, we have created a new application server, ACompanyServer, whose port number is 2810. Thus you need to modify the brbeansPropertiesFile with the correct server port number.
  - If the server name and the port number are not specified, then the name server used by the container in which the application is running is used. If the application is not running in a container, then `localhost` is used for the host name, and 900 is used for the port number.
- As discussed before, during runtime the trigger point object fetches the rule required, then it fires it. You might notice that the fetching process is successful (no errors or exceptions are thrown), but the rule doesn't perform the desired task. This might be because the rule is not in effect. You may check the rule status using the Rule Management Application. The rule status might be expired or scheduled. The rule status must be in effect for the rule to function properly.

10.8 Performance considerations

Every business rule is represented as an EJB at runtime. As a result, whenever a rule is triggered, a query is performed to find the EJBs representing the rules to be triggered, and a remote method call is performed on the EJB to actually trigger the rule. Since both of these actions can be costly from a performance perspective, this section discusses the techniques that can be used to improve performance.

10.8.1 Client-side caching

The Business Rule Beans framework incorporates a cache on the client side. The scope of this cache is limited to the JVM in which the client is running, so that any trigger calls performed in a particular JVM will use the same cache. The Business Rule Beans cache caches the results of all queries performed to find a set of rules to be triggered. The next time a trigger is performed in that JVM with the same rules specified, the rules will be found in the cache and the query will not actually require going to the server.

The disadvantage of using the client-side cache is the inability to recognize Business Rule Beans rule configuration changes immediately. A cache polling frequency determines how often the client-side cache is refreshed. The next time a trigger is performed after a polling interval has passed, the cache will check to see if any changes at all have been made to the persistent rule data stored on the server. If no changes have been made, then the cache is not refreshed. If any changes at all have been made, then the entire cache is cleared so that the changes will be picked up. The default polling frequency is 10 minutes. You can change the polling frequency in the Rule Management Application.

1. Start the Rule Management Application if it is not already started.
2. Expand the **Rule Namespace** folder.
3. Expand the **com -> ibm -> websphere -> brb** folder.
4. You should see BRB CacheRule as shown in Figure 10-20.

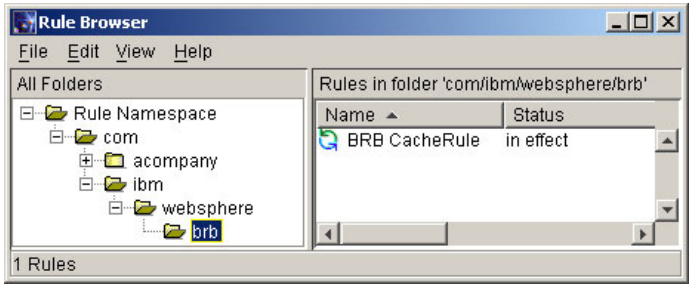


Figure 10-20 Rule Management Application - BRB CacheRule

5. Select **BRB CacheRule**, right-click, and select **Properties**.
6. In the Properties window, select the **Implementation** tab.
7. Select the **PollFrequency** initialization parameter, and click the **Change** button. Note that the value for this initialization parameter has the format hh:mm:ss, where hh stands for hours, mm stands for minutes, and ss stands for seconds.
8. Change the value of the PollFrequency initialization parameter, as shown in Figure 10-21 on page 453. Click **OK**.

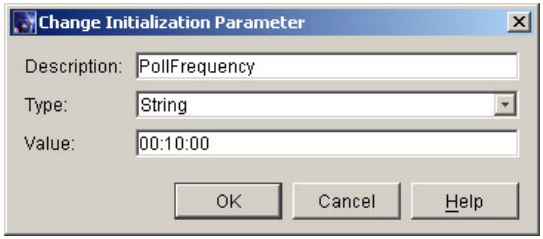


Figure 10-21 Changing rule initialization parameters

Caching can be disabled for a particular TriggerPoint object using the disableCaching method. After disableCaching is called, any triggers performed using that TriggerPoint object will not use the cache. Triggers performed using other TriggerPoint objects are not affected.

10.8.2 Rule firing location

The Business Rule Beans framework allows you to specify where a particular rule should be fired. This determines where the rule implementor for the rule is actually instantiated and invoked. There are three possible values for the firing location:

1. Local

   Fires the Java rule implementor local to the client that fired the rule, that is, in the same JVM in which the trigger was performed. The disadvantages of firing the rules locally is the requirement to install rule implementors on each client system that uses Business Rule Beans rules. The best performance is achieved by using the client-side cache and local firing location, since no remote server calls are performed in such a configuration.

2. Remote

   Fires the Java rule implementor on the application server where the Business Rule Beans are installed. RuleImplementors must be in the server classpath. Without caching, the set of actions done to fire a rule require a remote call to the server. The following are the actions performed to fire a rule remotely:

   – Finding the rule
   – Determining whether the rule is to be fired locally or remotely
   – Calling the fire on the remote rule

3. Anywhere

   First tries to fire the Java rule implementor locally. If the Java rule implementor cannot be found, then it fires the Java rule implementor remotely. This is the default value.

The firing location parameter for a specific rule is set on the Implementation tab of the rule Properties window in the Rule Management Application, as shown in Figure 10-22.

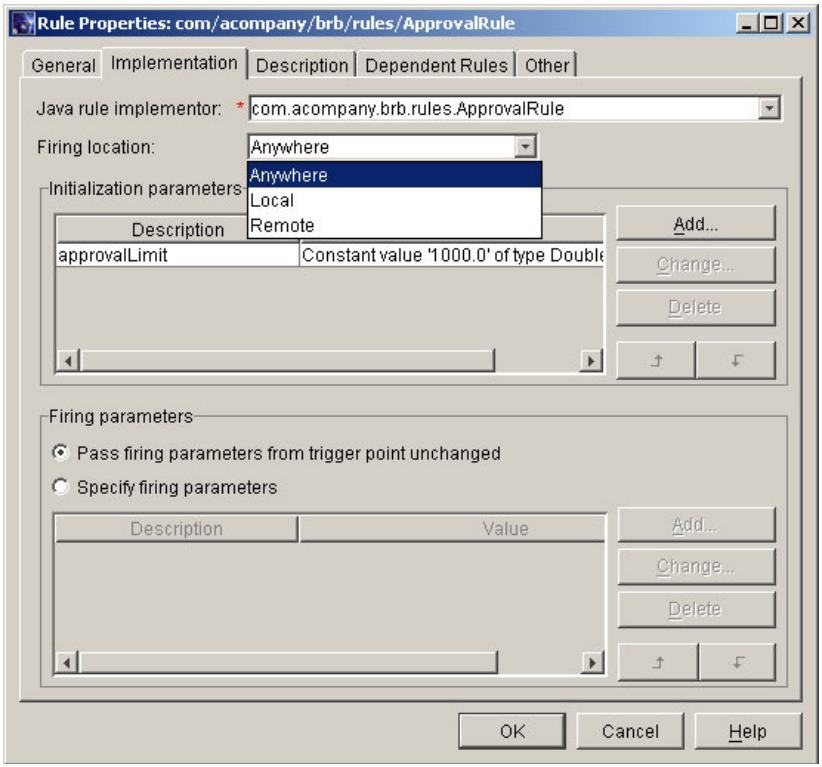


Figure 10-22 Configuring rule firing location

Note that, in addition to performance, maintenance must be considered in relation to specifying a rule firing location. The rule implementor classes for rules that are defined to be fired locally must be present on any client system that tries to fire those rules. Otherwise, the implementor cannot be instantiated when the rule is fired. This can result in maintenance problems when the rule implementors are changes, since they must be updated on many different systems.

10.8.3 Creating database indexes

Creating an index over the database table that is used to store rules is an important way to improve the performance of rule queries. It is recommended that an index be created over the rulename column of the table containing the rules. This greatly improves the performance of rule-triggered queries that are

looking for rules with specific names. The index saves the query the effort of searching every row in the table.

10.9 Security considerations

As discussed earlier, Business Rule Beans EJBs are major components of a runtime environment for the framework. At runtime, Business Rule Beans EJBs are responsible for finding and firing Business Rule Beans rules. These are three main EJBs in the Business Rule Beans framework: Rule, RuleFolder, and RuleHelper. Thus Business Rule Beans don't have specific security considerations. They just need to follow regular EJB security handling.

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 that need to be aware of the role or identity or the caller can programmatically call the J2EE API methods isCallerInRole and getCallerPrincipal to determine a caller's role and principal, respectively. When using isCallerInRole, security role references are used, which are later mapped to security roles.

There are three security roles attached to the Business Rule Beans framework:

- RuleUser: The person who uses an application that contains trigger points.
- RuleManager: The person who will use the Rule Management Application or other application that uses the Rule Management APIs. This person will be allowed to create, modify, and delete rule and rule folders.
- DenyAllRole: Deny all access role.

For more details about securing Enterprise Java Beans, refer to *IBM WebSphere V5.0 Security, WebSphere Handbook Series*, SG24-6573.

11

Dynamic Query

This chapter introduces the benefits of the Dynamic Query service and how this feature of WebSphere Application Server Enterprise V5 fits your business needs.

This chapter covers the following topics:

- An introduction to Dynamic Query service, and its use in enterprise applications
- Design guidelines and considerations
- Development process

The second part of this chapter explores the runtime aspects of dynamic queries:

- Deployment
- Troubleshooting
- Performance considerations
- Security considerations

11.1 Planning

Dynamic Query service is provided by WebSphere Application Server Enterprise in the area of data access through CMP EJBs. The base application server supports the standard EJB QL specifications, with some extensions that increase the flexibility of the syntax of EJB QL. However, one of the characteristics of EJB QL is its static nature. Once you have associated an EJB QL query to a finder method and deployed your EJB, the query can only be modified by redeploying and re-installing the EJB. The Dynamic Query service removes this limitation. It allows applications to formulate queries at runtime and have them submitted and executed on the fly, providing a behavior that resembles that of traditional dynamic SQL. The Dynamic Query service works at the object schema level, just like regular EJB QL. So this is a form of object query, and not a database query. It works for CMP Entity EJBs, allowing you to perform queries on both CMP and CMR fields.

11.1.1 Dynamic Query

Dynamic Query service maintains the same features of EJB 2.0 QL and of the base WebSphere Query enhancements, while providing even more functionality. Table 11-1 summarizes the extent of query support in WebSphere.

Table 11-1 Query support in WebSphere

Function	EJB 2.0 Query	Base WebSphere Query	Dynamic Query service
Select clause	required	optional	optional
Delimited identifiers	no	yes	yes
String comparisons	= and <>	= <> > <	= <> > <
Scalar functions	some	more	more
Calendar comparisons	yes	yes	yes
Order by	no	yes	yes
SQL date/time expressions	no	yes	yes
Query over inheritance graph	no	yes	yes
Subqueries	no	yes	yes
EXISTS predicate	no	yes	yes
DISTINCT predicate	no	yes	yes

Function	EJB 2.0 Query	Base WebSphere Query	Dynamic Query service
Aggregate functions	no	in subquery	yes
GROUP BY, HAVING	no	in subquery	yes
Bean business methods	no	no	yes
Dependent Value methods	no	no	yes
Multiple element select	no	no	yes
Dynamic Query execution	no	no	yes

It is clear from the table that the Dynamic Query service overcomes the need to statically define the query in the EJB Deployment Descriptor, by supporting query definition at runtime, unlike the WebSphere Application Server Base Query, where there is no dynamic capability. Moreover, the Dynamic Query service supports a number of other syntactical extensions, such as the aggregate functions (MAX, MIN, SUM, etc.) for all queries. In WebSphere Application Server base, these functions are only supported in subqueries. The GROUP BY and HAVING clauses are also supported by the Dynamic Query service.

In addition to these syntactical extensions, Dynamic Query service supports the use of business methods in the query projection and predicate clause. Let's say for example that we have a Department entity bean, and we need to find all department numbers where the department purchases are more than $10000. The query can be formulated as shown in Example 11-1. In this example, the business method computeTotalOrders is used in the query where clause. For WebSphere Enterprise to perform this query containing the method call, instances of the Department bean need to be activated. If the query runs across a large number of instances, these queries may take significant time and resources to complete.

Example 11-1 Business methods use in Dynamic Query

```
select d.id from Department d where d.computeTotalOrders() > 10000
```

Another extension that the Dynamic Query service provides is the support for data array queries. For example, the query in Example 11-2 will find all the `cid`, `cname`, and `cdeptid` of a Customer object.

Example 11-2 Data Array queries

```
select c.cid, c.name, c.cdeptid from Customer c
```

Why use Dynamic Query?

WebSphere Application Server Enterprise, through the Dynamic Query service, provides high-performance, scalable access to objects representing processes and entities, supplemented by the speed and efficiency of a relational database for querying, grouping, and sorting. The capability to query EJBs based on their attributes and methods, and their relationships with other EJBs, serves as the foundation for a variety of enterprise applications. For example, let's say that an airline has modeled their business domain as a set of EJBs. Let's say that a Flight EJB encapsulates the data and business functionality of one flight. Pricing of a seat on a flight is a dynamic, complex process, so it is implemented by a calculatePrice() method on the Flight EJB. There is an endless set of scenarios where it would be useful to include the calculatePrice() method on an EJB query. For instance, travel agents, or passengers might want to search for flights between a set of locations, with prices falling within a defined range.

The use of dynamic queries can be beneficial in financial applications, where an assessRisk() method on a FinancialInstrument EJB dynamically calculates a risk score for a particular instrument, based on current market conditions. This would clearly be a useful element to include in the select or where clauses of an EJB query. This would enable easy selection of instruments with risk measurement falling in a specified range and inclusion of the risk scores in the return set.

The possibilities of the Programming Model Extensions are vast. They are useful in almost every industry and in every enterprise. For instance, data mining and focused consumer marketing applications require the ability to execute queries with criteria that are not known at development time. In fact, query criteria might change from day to day. Such applications also typically need to group and sort query results in multiple ways.

The use of the Dynamic Query service in an enterprise application has the following benefits:

- There is no need to know the query search criteria until application runtime.
- Dynamic Query service allows you to return multiple CMP or CMR fields from a query, while deployment queries allow only a single element to be specified in the SELECT clause.
- Dynamic Query service allows you to perform aggregation in the query, while deployment queries do not allow use of the aggregation function SUM, AVG, COUNT, MAX, MIN in the top level SELECT of a query.
- Dynamic Query service allows you to use value object methods or bean methods in the query statement.
- In some cases, using the Dynamic Query service may alleviate performance issues with EJB QL queries. Using the Dynamic Query service, you can formulate queries that would require a lot of hand-coded processing if you

used the standard EJB QL. For example, assume that you have a query that returns the maximum salary of employees, on a department-by-department basis. With Dynamic Query service, you can formulate and execute your query, and get exactly the result you need, without further processing and without requiring EJB activation. With regular EJB QL, your program would have to manually do the grouping and aggregation, and EJBs would have to be massively activated.

- Dynamic Query service may reduce the need for specific finders. You may add finding behavior to an existing application without even redeploying it.
- You can use the Dynamic Query service when you want to interactively test an EJB query during development but do not want to repeatedly deploy your application each time you update a finder or select query.

11.2 Design

This section introduces you to the technology behind the design of the Dynamic Query service, and the possible interaction between enterprise application clients and the Dynamic Query service. Then we will introduce some design considerations and recommendations when using this Programming Model Extension.

Dynamic Query service

The Dynamic Query service is implemented using “push down” technology. The technology involves taking an object query statement and using metadata that describes the mappings of EJB attributes to database tables and columns, to translate it to an SQL statement that can be executed by the database management system. In effect, the EJB query is pushed down to the database, which is designed to perform this type of query evaluation, and groups and sorts very efficiently. As would be expected, not all of an EJB query can be pushed down. The application server must still evaluate some query criteria, such as results of method invocations. By delegating most of the work, including navigation of relationships, to the database, excellent performance can be provided. Lastly, this push down technology is designed to be independent of the database vendor. It works with any relational database supported by WebSphere.

The Dynamic Query service is provided by the stateless Query session bean. The client of the Query session bean may be a remote client or it may be a local client, depending on whether the client makes use of the bean’s remote or local interfaces. A remote client accesses the query bean through the bean’s remote interface, and remote home interface. A remote client can be any Java program such as an application, JSP, applet, or servlet. A local client accesses the query

bean through the bean's local interface and local home interface. A local client is collocated in the same JVM with the query bean and can be another enterprise bean such as a session bean, entity bean or message-driven bean, as shown in Figure 11-1. The Query Engine can perform queries on entity beans, on CMP fields, and on CMR fields, represented by Entity EJB X and Entity EJB Y in the figure.

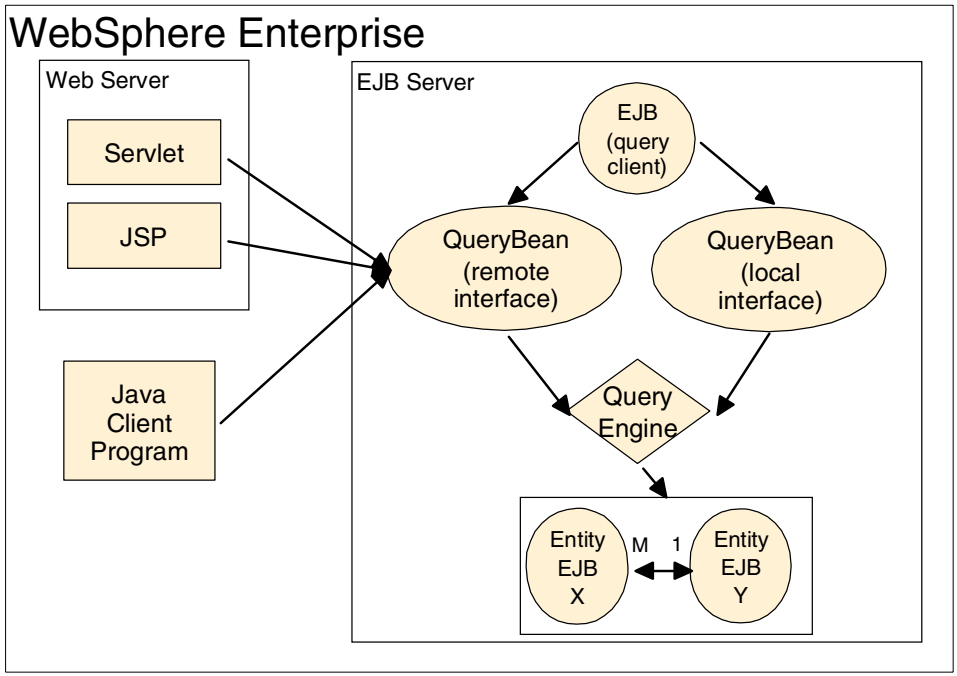


Figure 11-1 Dynamic Query service

Design concerns and recommendations

There are two major concerns to explore before deciding to use Dynamic Queries with your enterprise applications. One of these concerns is related to performance, while the other is related to security.

- If you have a query that has a high frequency of execution, you should define it as a finder or select method and consider using SQLJ as a deployment option for best performance. The Dynamic Query service always uses JDBC and must parse and process the EJB query at runtime.
- If you need security control over which queries a user can execute, you need to define the queries as finder or select methods and use EJB method authorization. The Dynamic Query service does not have fine-grain security control at this time. You can control who is permitted access to the remote

query bean and the local query bean, but once authorized a user can execute any valid query and return any data in the server.

11.3 Development

The Dynamic Query API, the stateless session bean named query, is provided in QueryClient.jar. Using the Dynamic Query API is similar to using any other J2EE EJB application bean. The default JNDI name for the query bean is com/ibm/websphere/ejbquery/Query. The query bean has both a remote and a local interface to support both remote and local clients, as shown in Table 11-2.

Table 11-2 Query bean interfaces

Interfaces	Classes
remote interface	com.ibm.websphere.ejbquery.Query
remote home interface	com.ibm.websphere.ejbquery.QueryHome
local interface	com.ibm.websphere.ejbquery.QueryLocal
local home interface	com.ibm.websphere.ejbquery.QueryLocalHome

Table 11-3 shows the JAR files that comprise the Dynamic Query service.

Table 11-3 Dynamic Query service JAR files

JAR File	Usage
query.jar	Query parser and runtime
qjcup.jar	Auxiliary classes for parser
querybean.jar	Query session bean
qryclient.jar	Client stubs and classes
querymd.jar	Auxiliary classes for query
queryws.jar	Auxiliary classes for query

The following sections discuss in detail how to use the query bean and the services it provides, then focus on the development environment setup and the development process.

11.3.1 Dynamic Query Bean

The QueryBean interface has three client methods:

- executeQuery(): this method parses and executes the query in a single operation. The executeQuery method in the remote interface has some extra arguments over the method provided by the local interface.
  - The common arguments to local and remote interface are:
    - java.lang.String queryStatement, which is a string containing the query statement to be invoked, for example”

      ```
      select Object(c) from Customer c
      ```

    - java.lang.Object[] parameterVars, which is an array of objects used as value holders for literal values. Queries can include optional parameters, and you can specify values for those parameters in the parameterVars array. For example:

      ```
      select Object(c) from Customer c where c. cdeptid > ?1
      ```

    - Note that positional numbering starts at 1.
    - java.util.Properties queryDomain, which allows you to point to the EJB that you want to run the query on, by specifying the Abstract Schema Name and Application/Module/Component AMC pairs. The AMC name is used to distinguish between multiple installed applications sharing the same Abstract Schema Name. This situation occurs if multiple EJB JARs are installed on the same application server, and they have beans with the same Abstract Schema Name. If the AMC is not specified, then the Schema Name is assumed to be unique.
  - The remote interface extra arguments allow the remote client to select a portion of the resultset, rather than having to deal with the entire resultset. These arguments are:
    - `int skipRows`: used to request a subset of the results. Records are retrieved starting skipRow+1.
    - `int maxRows`: used to specify the number of rows required for retrieval. If set to java.lang.Integer.MAX_Value, all results starting at skipRow+1 are returned.
  - The return value type for this method is QueryIterator(). It covers:
    - A collection of objects implementing the entity bean’s remote or local interface
    - A collection of values returned by business methods
    - A collection of CMP or CMR fields

- prepareQuery(): this method invokes the query parsing and plan creation process. It receives exactly the same three common input parameters described above (queryStatement, parameterVars, and queryDomain). This method returns the optimized query plan generated from the query string input parameter. Normally a client would not directly invoke the prepareQuery or executePlan methods. The return type of this method is a string containing the created query plan, which is a query statement parsed, validated, and optimized, and it is presented as an input to the executePlan method.

  Note: There might be times when the same query is being executed multiple times. Therefore, some performance optimization can be received by invoking prepareQuery once, and then invoking executePlan multiple times, rather than invoking executeQuery multiple times. Doing this helps save parsing effort.

- executePlan(): this method executes a query plan that is in a string text form. It receives two input parameters for the local interface, and an additional two for the remote interface.
  - The two common parameters are:
    - queryPlan: represents the query plan in string format
    - parameterVars: same as executeQuery
  - The additional parameters for the executePlan method in the remote interface are:
    - skipRows: same as executeQuery
    - maxRows: same as executeQuery

Local client programming model

For a local client to use the Dynamic Query service, the client implementation should perform the following steps:

1. Look up the QueryLocalHome.
2. Populate the parameter list.
3. Create an instance of the Query bean.
4. Formulate the query string.
5. Run the query.
6. Iterate through the resultset. Each iteration will retrieve a tuple. For each tuple, the client should get the name of the field and its corresponding value. If one of the values is an EJB reference, then that will be an EJB local reference. You just need to cast it to the correct type using normal Java

casting. It has to be narrowed to the correct type using java.rmi.PortableRemoteObject.

This sequence of steps is shown in Example 11-3.

Example 11-3 Local client using Dynamic Query

```
QueryLocalHome home =
   (QueryLocalHome)context.lookup (“java:comp/env/ejb/QueryLocalHome”);
Object[] params = new Object[] {new Integer(10000)};
QueryLocal queryEngine = home.create();
userTransaction.begin();
String query =
   “select Object(c) from Customer where c.cdeptid() = ?1”;
try{
   QueryLocalIterator it = queryEngine.executeQuery(query, params, null);
} catch(QueryException e){
   System.out.println(e.getMessage());
}
System.out.println(“There are “ + it.getFieldsCount() + “fields.”);
while (it.hasNext()){
   IQueryTuple tuple = (IQueryTuple) it.next();
   Customer c = (Customer) tuple.getField(1);
   System.out.println(“Customer Name = “ + c.getName());
   System.out.println(“Total Orders = “ + c.calculateTotalOrders());
}
```

Note: When using a local client with Dynamic Query, the query must be invoked from within a transaction scope. At transaction termination, local query iterator is invalidated.

Remote client programming model

For a remote client to use the Dynamic Query service, the client implementation should perform the following:

1. Look up the QueryBean home.
2. Populate the parameter list.
3. Create an instance of the Query bean.
4. Formulate the query string.
5. Specify the desired number of rows to be retrieved.
6. Run the query.
7. Iterate through the resultset. Each iteration will retrieve a tuple. For each tuple, the client should get the name of the field and its corresponding value. If

one of the values is an EJB reference, that it has to be narrowed to the correct type using java.rmi.PortableRemoteObject.

An example of this sequence of steps is shown in Example 11-4.

Example 11-4 Remote client using Dynamic Query

```
QueryHome h =
Object[] params = new Object[] {new Integer(10000)};
Query q = h.create();
String query =
   “select Object(c) from Customer where c.cdeptid() = ?1”;
int skip = 0;
int max = 100; //fetch first 100
try{
   QueryIterator it = q.executeQuery(query, params, null, skip, max);
} catch(QueryException e){
   System.out.println(e.getMessage());
}
System.out.println(“There are “ + it.getFieldsCount() + “fields.”);
while (it.hasNext()){
   IQueryTuple tuple = (IQueryTuple) it.next();
   Customer c =
   (Customer)PortableRemoteObject.narrow(tuple.getField(1),Customer.class);
   System.out.println(“Customer Name = “ + c.getName());
   System.out.println(“Total Orders = “ + c.calculateTotalOrders());
}
```

Important: When an object is selected by the query, the return value is an EJB local reference, which needs casting to the correct type using normal Java casting, when using a local client. When using a remote client, the EJB reference has to be narrowed to the correct type using java.rmi.PortableRemoteObject.

11.3.2 Development environment setup

When developing applications that use the Dynamic Query service, you need to start by setting up WebSphere Studio WebSphere Studio IE to develop and run applications using dynamic queries.

Import the Dynamic Query EAR file

The following are the steps for setting up the environment for developing dynamic queries. From the J2EE perspective:

1. From the main menu, select **File ->Import**. Select the option to import an EAR file, and click **Next**.

2. Click the **Browse** button to look for the query.ear file, which is available in <*WebSphere_Studio_IE_root*>\runtimes\ee_v5\installableApps. Select the **query.ear** file, and click the **Open** button.
3. Give a name for the project that you will import the Dynamic Query code to, for example QueryEJB, and click **Finish**.

You will have as a result a new QueryBean project containing the Query session bean. To have a look at it, switch to the J2EE Hierarchy view in the J2EE perspective, and expand the node **EJB Modules**. Expand the **querybean** project, and there you will find the Query session bean.

11.3.3 Development of Dynamic Query sample

In our sample application, described in Chapter 3, “Sample scenario” on page 25, we have three main entity beans: Catalog, Orders, and Customers. We are going to use the Dynamic Query service to query the entity beans, and the relationship Customers-Orders. The components of the sample are shown in Figure 11-2.

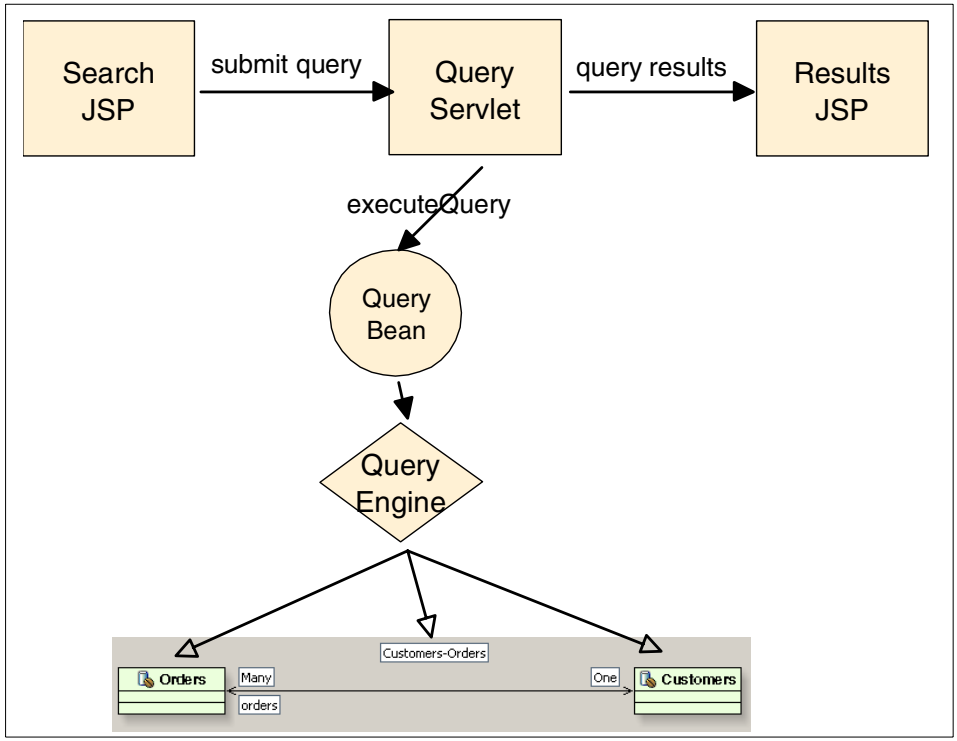


Figure 11-2 Dynamic Query sample

The Search JSP provides the user with a set of predefined queries, such as:

- List all orders by a customer on a specific item.
- List all orders by a customer since a specific date.
- Get the total price of all orders by a customer since a specific date.
- List all orders by a customer's department on a specific item.
- List all orders by a customer's department since a specific date.
- Get the total price of all orders by a customer's department since a specific date.

In addition to these predefined queries, a user can define a query using the Search JSP. Once the user submits a query, the query is redirected to the QueryServlet, which performs a lookup for the Query home, creates an instance of the query bean, formulates the query string, and runs the query. Once the query is executed, the servlet forwards the resultset to the Results JSP, which iterates on the resultset, displaying the selected fields and the resulting values.

Customers EJB built-in query

According to the above list of queries, we need to have the means to get a list of all customers in a specific department, in order to be able to retrieve all the orders placed by that department. In this section, we are going to add this feature to the Customers entity bean. Following are the steps:

1. Switch to the J2EE perspective, J2EE hierarchy view. Expand **EJB Modules -> ACompanyEJB -> Customers**. Double-click the **Customers** bean. This will open the EJB Deployment Descriptor, and the CustomersBean will be selected in the Beans tab.
2. Scroll down to the section titled **Queries**, to define an EJBQL query for the Customers bean. Click the **Add** button.
3. Check **New** to specify that you are creating a new method, and select the radio button **ejbSelect** for the method type. Give the method a name, for example ejbSelectCustomersInDept. Click the **Add** button to specify the parameters that this method receives. In our sample application, this method needs a String argument representing the department ID. Specify that the method return type is java.util.Collection. Click **Next**.
4. Edit the query statement.

   ```
   SELECT t.cid FROM Customers t WHERE t.cdepartmentid = ?1
   ```

5. Click **Finish**.
6. Save the EJB Deployment Descriptor and close. If you now open CustomersBean you will find the new select method at the end of the class implementation. It will look as follows:

   ```
   public abstract java.util.Collection ejbSelectCustomersInDept(
         java.lang.String departmentID)
   ```

```
        throws javax.ejb.FinderException;
```

7. Now having the query method ready, we will create a method in the CustomersBean that is going to use the ejbSelect method, and convert the Collection to a Vector. We will have a Customers EJB representing the customer placing an order, and we will need to retrieve the IDs of all the customers who are in the same department as the customer placing the order. So from our search JSP, we will invoke this method getCustomersInDepartment and it will return a Vector containing the IDs of all customers in the department.

Example 11-5 getCustomersInDepartment method

```
public Vector getCustomersInDepartment() {
   try {
      Vector v = new Vector();
      Collection allCustomers  = ejbSelectCustomersInDept(getCdepartmentid());
      Iterator iterator = allCustomers.iterator();
      while (iterator.hasNext()) {
         v.add(iterator.next());
      }
      return  v;
   } catch (javax.ejb.FinderException e) {
      System.out.println("**** Finder Exception ***:" + e.getMessage());
      e.printStackTrace();
   }
   return null;
}
```

8. Save and close CustomersBean.
9. Promote the newly created method to the beans's local interface, and generate the deploy code for the Customers bean.

bpeWebclient project configuration

In our sample application, instead of creating a new Web project for our JSPs, we are using the business process Web client bpeWebclient. You need to perform the following two actions, in order by able to use the Dynamic Query service with the JSPs you create in this project:

1. Switch to the J2EE perspective, select the **J2EE Navigator** view. Select the **bpeWebclient** project, and right-click to select the project properties. In the project properties window, select the option **Java Build Path**. Select the tab **libraries**, and click the button **Add External JARs**. Browse to add the file qtyclient.jar, which is present in <*WebSphere_Studio_IE_root*>\runtimes\ee_v5\lib. Select the **qryclient.jar** file, and click the **Open** button. Then click **OK**.

2. Expand the **bpeWebclient** project, select and double-click **Web Deployment Descriptor**. This will open the Deployment Descriptor editor. Select the **Source** tab from the top tabs. You will need to add local references for the Query and Customers beans, as shown in Example 11-6.

Example 11-6 bpeWebclient local references

```
<ejb-local-ref id="EJBLocalRef_1049752101031">
   <ejb-ref-name>ejb/QueryLocalHome</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
   <local-home>com.ibm.websphere.ejbquery.QueryLocalHome</local-home>
   <local>com.ibm.websphere.ejbquery.QueryHome</local>
</ejb-local-ref>
   <ejb-local-ref id="EJBLocalRef_1049840476047">
   <ejb-ref-name>ejb/Customers</ejb-ref-name>
   <ejb-ref-type>Entity</ejb-ref-type>
   <local-home>com.acompany.ejbs.CustomersLocalHome</local-home>
   <local>com.acompany.ejbs.CustomersHome</local>
</ejb-local-ref>
```

3. After modifying the source, switch back to the References tab and select the EJB Local tab at the top.
4. Select the ejb/QueryLocalHome entry and enter the value for the JNDI Name as `com/ibm/websphere/ejbquery/Query`.
5. Select the ejb/Customers entry and enter the value for the JNDI Name as `ejb/Customers`.
6. Save and close the descriptor.

Search JSP development

The steps to develop this sample search JSP are as follows:

1. Expand the **bpeWebclient** project, select the **Web Content** and right-click. Select **New -> JSP File**. Fill in the necessary information to create the JSP whose name is DQuery, for example, which will represent our search JSP. For detailed implementation of the JSP, refer to sample code provided in Appendix B, “Sample scenario” on page 665.

   a. The search JSP receives two parameters in the HTTPRequest: one is the ID of a customer and the other is the ID of an item in the catalog. So in the beginning of the JSP, you need to extract these two parameters from the request, as shown in Example 11-7 on page 472. After doing that, you need to create a Customers EJB, and use the findByPrimaryKey() method with the customer ID to construct a Customers EJB populated with the customer data. This bean is used to retrieve the list of customers in the same department as the current one. This is done using the method getCustomersInDepartment described in the previous section. This

method returns a vector of the customers' IDs. Then we use these lists to construct a string of these IDs to use later on in our queries.

Example 11-7 Search JSP

```
String itemID = request.getParameter(“itemID”);
String customerID = request.getParameter("customerID");
javax.naming.Context c = new javax.naming.InitialContext();
CustomersLocal customerHome =
   ((CustomersLocalHome) c.lookup("java:comp/env/ejb/Customers")).
   findByPrimaryKey(new CustomersKey(customerID));
String deptID = customerHome.getCdepartmentid();
Vector allCustomers = customerHome.getCustomersInDepartment();
String ids = "'" + ((Object)allCustomers.elementAt(0)).toString() + "'";
for (int i = 1; i < allCustomers.size(); i++)
   ids = ids + ",'" + ((Object)allCustomers.elementAt(i)).toString() + "'";
```

b. The following list shows the predefined query statements that are defined in the search DQuery JSP.

– List all orders by customer on a specific item:

```
SELECT o.odate as OrderDate, o.oqty as Quantity, o.oprice as UnitPrice,
(o.oqty*o.oprice) as TotalPrice  FROM Customers c, IN(c.orders) o WHERE
c.cid = ?1 AND o.oitem = ?2 ORDER BY o.odate
```

This select statement queries both the Customers EJB and the Customers-Orders relationship, retrieving order date, order quantity, unit price, and order total price of all previous orders performed by a customer on a specific item. The parameter list for this query includes the customer ID (cid) and the item ID (oitem). The retrieved results are ordered by the order date.

– List all orders by a customer since a specific date:

```
SELECT o.odate as OrderDate, x.cname as ItemName, o.oqty as Quantity
FROM Customers  c, IN(c.orders) o, Catalog x  WHERE c.cid = ?1 AND x.cid
= o.oitem AND o.odate > ?2
```

This select statement queries the Customers EJB, the Customers-Orders relationship, and the Catalog EJB, retrieving order date, order item name, and order quantity, for all orders by a customer since a specified date. The parameter list for this query includes the customer ID (cid) and the date. The retrieved results are ordered by the order date.

– Get total price of all orders by a customer since a specific date:

```
SELECT SUM(o.oqty*o.oprice) as TotalPrice from Orders o where
o.ocustomer = ?1 and o.odate > ?2
```

This select statement uses the summation aggregate function SUM to retrieve the sum of prices of all orders by a customer since a certain date.

– List all orders by a customer's department on a specific item:

```
SELECT o.odate as OrderDate, c.cid as CustomerID, o.oqty as Quantity,
o.oprice as UnitPrice, (o.oqty*o.oprice) as TotalPrice from Customers c,
IN(c.orders) o WHERE c.cid in ?1 AND  o.oitem = ?2 ORDER BY o.odate
```

This select statement queries both the Customers EJB and the Customers-Orders relationship, retrieving order date, customer ID, order quantity, unit price, and order total price of all previous orders performed by a customer on a specific item. The parameter list for this query includes the list of customer IDs (cid) and the item ID (oitem). The retrieved results are ordered by the order date. The list of customer IDs in the department is retrieved using the method getCustomersInDepartment shown in Example 11-5 on page 470, as mentioned earlier.

– List all orders by a customer's department since a specific date:

```
SELECT o.odate as OrderDate, c.cid as CustomerID, x.cname as ItemName,
o.oqty as Quantity, (o.oqty*o.oprice) as TotalPrice FROM Customers c,
IN(c.orders) o, Catalog x WHERE c.cid in ?1 AND  o.oitem = x.cid AND
o.odate > ?2
```

This select statement queries the Customers EJB, the Customers-Orders relationship, and the Catalog EJB, retrieving order date, the customer ID, order item name, order quantity, and the order total price for all orders by a customer's department since a specified date. The parameter list for this query includes the list of customer IDs (cid) in the department and the date. The retrieved results are ordered by the order date.

– Get total price of all orders by a customer's department since a specific date:

```
select SUM(o.oqty*o.oprice) as TotalPrice from Orders o where
o.ocustomer in ?1 and o.odate > ?2
```

This select statement uses the summation aggregate function SUM to retrieve the sum of prices of all orders by all customers in a department since a certain date.

c. The Submit Query button in the JSP submits to the QueryServlet. The parameters list to the servlet will include the selected query string, the customer ID, the item ID, the queries date argument, and the list of customer IDs in the department.

Query Servlet development

To create the query servlet, in the J2EE perspective, select the **J2EE Navigator** view, expand the **bpeWebclient -> Web Content**, select **Java Source**, right-click and choose to create a new servlet. You need to perform the following actions:

1. Specify a package and a class name for the servlet, such as QueryServlet, and click **Finish**.

2. The doPost method of the QueryServlet needs to begin a user transaction, since we are using the QueryBean local interface, look up the QueryLocalHome, and create the bean. Finally the servlet needs to invoke the executeQuery method passing the query string available in the HTTPRequest, in addition to the needed query parameters. The QueryServlet will set the resultset as an attribute in the HTTPRequest, and forward to the search results JSP.

Example 11-8 QueryServlet doPost method

```
javax.transaction.UserTransaction ut =
   (javax.transaction.UserTransaction) c.lookup("java:comp/UserTransaction");
ut.begin();

queryEngine = ((com.ibm.websphere.ejbquery.QueryLocalHome) c
             .lookup("java:comp/env/ejb/QueryLocalHome"))
             .create();
com.ibm.websphere.ejbquery.QueryLocalIterator iter =
      queryEngine.executeQuery(req.getParameter("query"), params, null);
req.setAttribute("resultset",iter);
getServletContext().getRequestDispatcher("DQueryResult.jsp").forward(req,resp);
```

Results JSP development

In the J2EE perspective, select the **J2EE Navigator** view, expand the **bpeWebclient** project, select **Web Content** and right-click. Select **New -> JSP File**. Fill in the necessary information to create the JSP whose name is DQueryResult, for example, which will represent our search results JSP. Example 11-9 shows the implementation of the body of the results JSP. Mainly, the resultset attribute needs to be retrieved from the HTTPRequest, and then iterate through the resultset, to display the query results in a table format.

Example 11-9 Results JSP body

```
<BODY>
<%
   QueryLocalIterator iter = (QueryLocalIterator)
                              request.getAttribute("resultset");
%>
<table cellpadding="0" cellspacing="5" border="2" width="100%">
<%
   int fieldCount = iter.getFieldsCount();
   for(int k =0; k < fieldCount; k++) {
%>
      <td><%=iter.getFieldName(k+1)%></td>
<%  } // end for k
    while (iter.hasNext()) {
      IQueryTuple rows = (IQueryTuple)iter.next();
```

```
%>
      <TR>
<%
         for(int i = 0; i < fieldCount; i++){
%>
         <TD><%= rows.getObject(i+1).toString()%></TD>
<%
         } // end for
%>
      </TR>
<%
    } // end while
%>
</TABLE>
</BODY>
```

11.3.4 Integration of Dynamic Query with sample application

The main point of integration of the Dynamic Query sample and the sample application is adding this function to the Purchase Order process Web client. The scenario is as follows. The person approving an order can use the Dynamic Query sample to retrieve information about previous orders by the person placing the new order. The approver can select any of the already defined queries to execute, and also can specify queries at runtime, using the process Web client.

The Dynamic Query sample can be integrated into the default process Web application. Open the Activity.jsp and insert the code, shown in Example 11-10, into the file at line 221, right after the `<%-- End of Headline of Content frame --%>` line.

Example 11-10 Dynamic Query code

```
<%-- inserting the Dynamic Query application --%>
<%
   String DQprocessStarter=null;
   String DQitemID=null;
   String DQattributeName=null;
   String DQattributeValue=null;

   if(process!=null) {
      DQprocessStarter=process.getStarter();
   }

   if (((attributeInputMessageNames != null) &&
(attributeInputMessageNames.size() != 0)) || (inputMessageJSP != null)) {

      for (int i=0; i < attributeInputMessageNames.size(); i++) {
```

```
            DQattributeName = (String) attributeInputMessageNames.elementAt(i);
            DQattributeValue = (String)
context.getInputMessageAttribute(DQattributeName);
            if(DQattributeName.equals("itemID")) {
               DQitemID=DQattributeValue;
               break;
            }
         }
   }
%>
<% if(DQprocessStarter!=null && DQitemID!=null) { %>
<FORM NAME="queryForm" ACTION="Dquery.jsp" METHOD="GET">
<INPUT TYPE="hidden" NAME="customerID" VALUE="<%= DQprocessStarter %>">
<INPUT TYPE="hidden" NAME="itemID" VALUE="<%= DQitemID %>">
<INPUT TYPE="submit" VALUE="Run research for this order.">
</FORM>
<% } %>
<%-- end of inserting the Dynamic Query application --%>
```

This code puts a form onto the Web page with a button that takes the user to the Dynamic Query page. The information for the query, process starter and item ID is gathered from the process instance, which is available as an object in the request.

Note: The code above can be found in the MyBpewebclient Web application. Since a new process Web client is generated for every WebSphere Enterprise test server, the code is integrated into the customized process Web client.

11.4 Unit test environment

The QueryBean is packaged in a query.ear file that is automatically installed on the default server during the WebSphere Enterprise installation. This EAR file needs to be manually installed in the Universal Test Environment, in order to be able to test an enterprise application that uses the Dynamic Query service.

11.4.1 Configure application server

The following steps show how to set up and start the ACompanyUnitTestServer to be able to test the sample Dynamic Query application:

1. Switch to the Server perspective, in the Server Configuration view, expand Servers, select **ACompanyUnitTestServer**, right-click, and select **Add**. Select the **QueryEJB** project in order to add it to the server.

2. Select **ACompanyUnitTestServer**, and double-click to open the server for editing. In the editor, select the **Configuration** tab. If you are using a remote client for your Dynamic Query, then in the Server Settings section, the Application class loader policy should be set to SINGLE. If a remote client is used then, it should be set to MULTIPLE.
3. Save the server configuration and close.

11.4.2 Running the sample application

The following steps show how to run the sample Dynamic Query application:

1. Start ACompanyUnitTestServer.
2. Open a browser window and type the following URL:

   ```
   http://localhost:9080/bpe/Dquery.jsp?customerID=1&itemID=1
   ```

   The JSP user interface is shown in Figure 11-3 on page 478. The user can select any of the buttons to select a query. This will display the query statement in the text field at the bottom of the page. Some of the queries require that the user specify a date using the day, month, and year drop-down lists before selecting the query. After the user selects the desired query, or types a query in the text field, the user should click the **Submit Query** button. The search results page will display the query resultset in a tabular form as shown in Figure 11-4 on page 478.

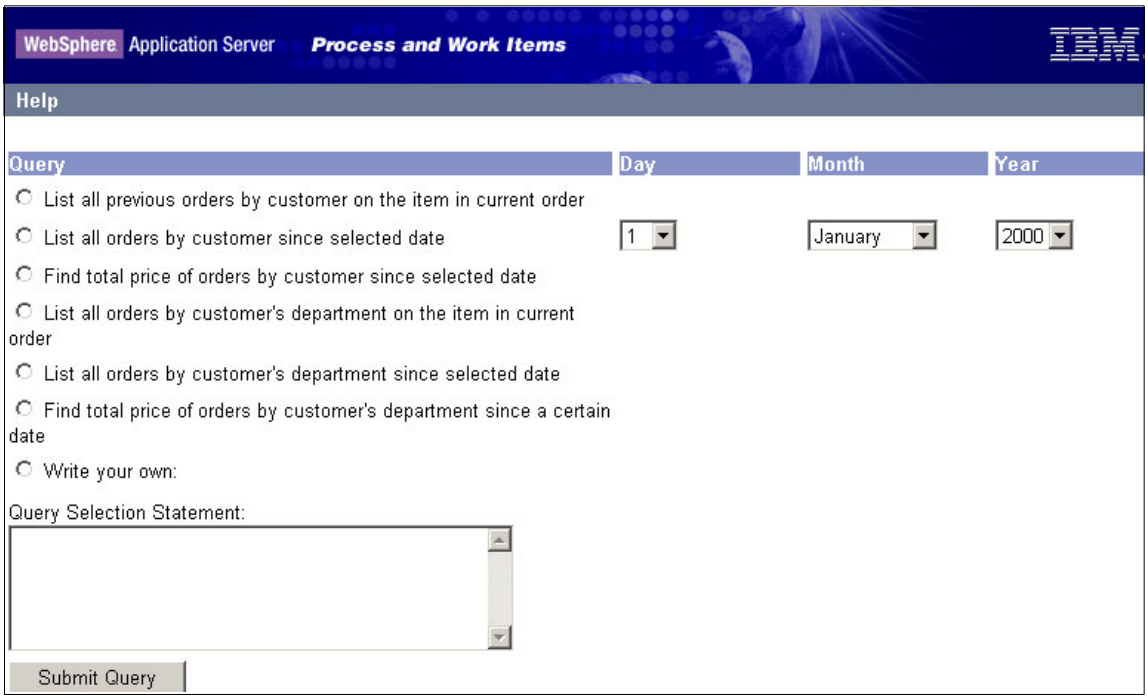


Figure 11-3 Sample search JSP

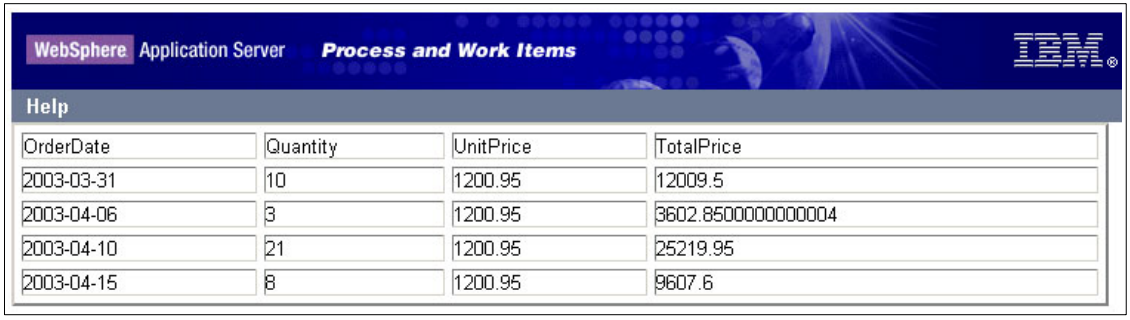


Figure 11-4 Sample search results

11.5 Assembly

After successful completion of testing the sample application integrated with Dynamic Query, you need to assemble two components: the ACompany EAR file and the bpeWebclient WAR file. After exporting the necessary files, you can open the application's EAR files using the Application Assembly Tool. You will

need to configure the EJBs' Access Intent when invoked by a Dynamic Query. Once the necessary changes are done to the EAR file, it can be saved using AAT and deployed in WebSphere.

The second action that you will need to do is to incorporate the bpeWebclient with our new JSPs into the business process container in WebSphere. This can also be done using the Application Assembly Tool.

The following section guides you through the process of exporting the project files, and doing the necessary changes using AAT.

11.5.1 Projects export

In "Customers EJB built-in query" on page 469, we modified the original Customers bean by adding the query and the method to retrieve the IDs of all the customers in a department. This change requires redeploying the ACompany EJBs. So you need to do the following:

1. From WebSphere Studio IE main file menu, select **Export -> EAR file**. Click **Next**.
2. Select **ACompany** as the resource you want to export, and browse to specify the location where you want to save your EAR file.
3. Click **Finish**.

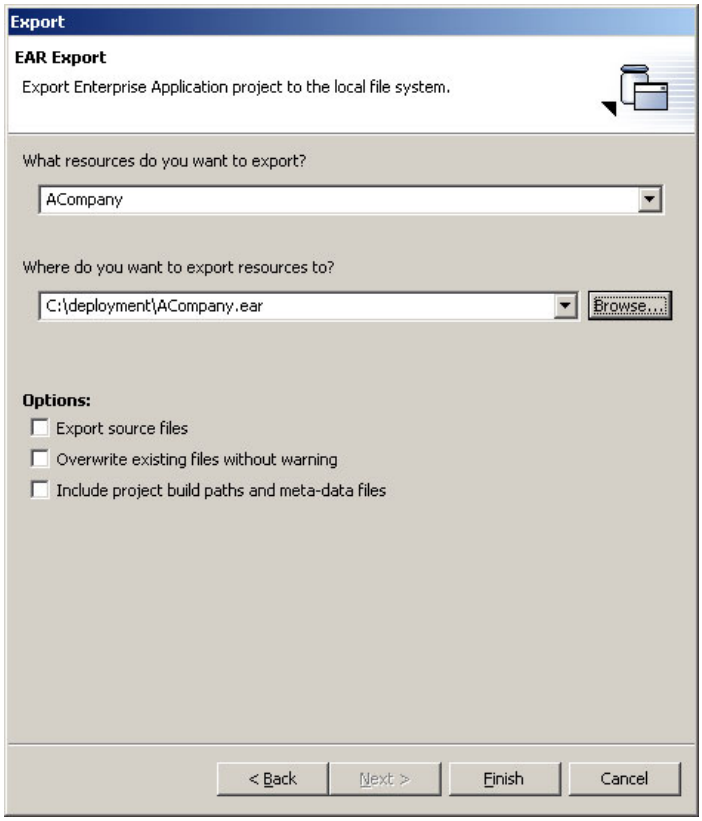


Figure 11-5 Exporting ACompany EAR

Now you need to export the bpeWebclient WAR file. From the main menu, select **File -> Export -> WAR file** and click **Next**. Select the **bpeWebclient** as the resource you want to export, specify a location for saving the file, and click **Finish**.

11.5.2 Configuring EJB Access Intent for Dynamic Query

You can set an Entity EJB to be accessed with a specific Access Intent when it is loaded as a result of a Dynamic Query. This Access Intent is defined at the bean level. For details about the possible policies that you can assign for the bean's Access Intent, refer to "Access Intent Policies" on page 349 and specifically Table 8-2 on page 351. To configure an EJB Access Intent Policy, you need to use the Application Assembly Tool. Following are the steps for assigning bean Access Intent for dynamic queries:

1. Open the ACompany project EAR file using the Application Assembly Tool.

2. Expand the ACompany project, and select **EJB Modules -> ACompanyEJB -> Dynamic Query**. Right-click and select **New**. The New Dynamic Query Access Intent window opens, as shown in Figure 11-6.

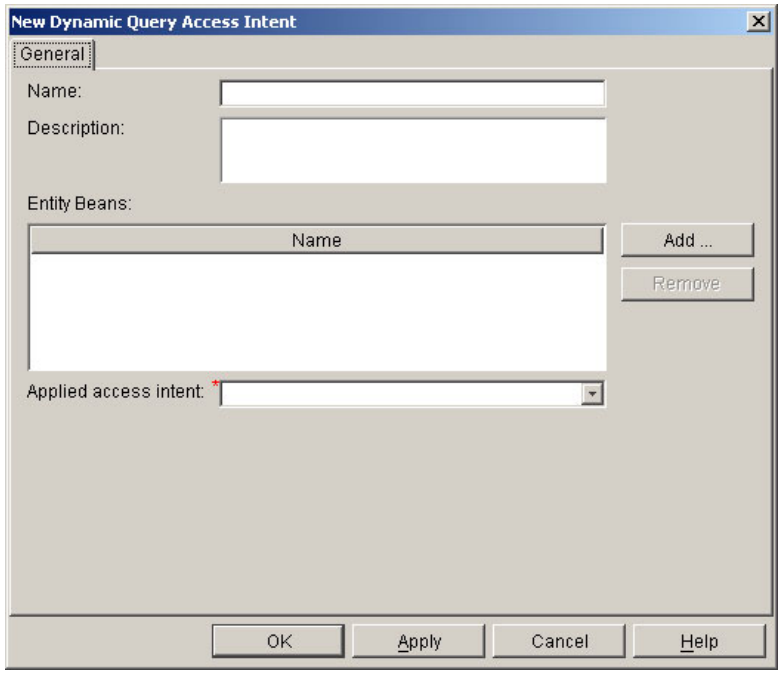


Figure 11-6 Application Assembly Tool - New Dynamic Query Access Intent

3. For each EJB in the project, you need to do the following:
   a. Give a name for the EJB Dynamic Query Access Intent.
   b. Click the **Add** button close to the Entity Beans field. This will open a window displaying all entity beans in your project. Select the bean for which you are creating the Access Intent.
   c. Select the Access Intent Policy of your choice from the drop-down list, and click **Apply**.
4. After you finish, save your changes, and save the EAR file to be used for deployment.

11.5.3 Incorporating bpeWebclient.war

Before doing this step, you need to make sure that business process container application (BPEContainer_cellName_ACompanyServer) that is installed on

ACompanyServer in your runtime environment is stopped. Start the Application Assembly Tool and follow these steps:

1. Open an existing EAR file, which will be located in <*WebSphere_root*>/installedapps/cellName/BPEContainer_cellName_ACompanyServer.ear.
2. Expand the **Web Modules** node. Select **BPEWebClient**, right-click and select **Delete**.
3. After you delete the existing Web client, select the **Web Modules** node, right-click and select **Import**. Browse and locate the bpeWebclient.war file that you have exported from WebSphere Studio IE, and select **Open**. The file name should be bpewebclient.war, the context root should be /bpe, and the display name BPEWebClient.
4. Save your changes and close AAT.

11.6 Configuration

The system administrator might need to install the query.ear application into the application server, if a new server is being created, since the WebSphere product install does this only for the default server. In the case of our sample, we are using a new server ACompanyServer, thus we need to install query.ear on ACompanyServer.

11.6.1 Installing query.ear

Since the WebSphere Enterprise installation only provides the Dynamic Query service with the default server (server1), we will need to install the service to the new server we have created for our sample application ACompanyServer. To do that, you need to start the WebSphere Administrative Console, and choose to install a new application. You will need to point to the query.ear file available in <*WebSphere_root*>\installableApps.

Follow the steps to install the query.ear, but you need to note two things. First, you need to assign a new name to the application, since the name query is already available in the repository with the server1. So for example you can name the new installation ACompanyQuery. The second thing that you should notice is to install this application on the ACompanyServer. After you finish the installation, you need to save your configuration.

After saving, and starting the ACompanyQuery application, the ACompanyServer will be ready for deployment of applications that use the Dynamic Query service.

11.6.2 Application class loader policy configuration

In our sample application we are using the local interface of the query bean. In order to use the local interface of the query bean, you must configure your server to use Application Classloader Policy = SINGLE.

Important: Using a value of MULTI may result in your application being unable to find the local interface for the query bean home.

To configure the ACompanyServer, select **Servers -> Application Servers -> ACompanyServer**. This will open the editor for configuring the server's general properties. For the Application class loader policy, select the value SINGLE from the available drop-down list, and click **OK**. You will need to save the updated server configuration and restart ACompanyServer.

11.7 Deployment

In Chapter 11, "Dynamic Query" on page 457, and in particular "Customers EJB built-in query" on page 469, we modified the original Customers bean by adding the query and the method to retrieve the IDs of all the customers in a department. And in 11.5.2, "Configuring EJB Access Intent for Dynamic Query" on page 480, we configured the Access Intent for the beans when invoked through a Dynamic Query. These changes require redeploying the ACompany EJBs. Thus, you will need to uninstall the ACompany application from ACompanyServer, and install it again in the new EAR file. Thus you need to do the following:

1. Make sure that ACompanyServer in your runtime environment is started.
2. Stop the ACompany application, and uninstall it.
3. Once the ACompany application is uninstalled, select **Applications -> Install New Application**.
4. Follow the steps for installing the newly exported ACompany EAR file. You don't need to select the deploy EJB option, since we have already generated the deployed code using WebSphere Studio IE. Make sure that you are installing the ACompany application on ACompanyServer, and not on the default server (server1).
5. Start the ACompany application, and make sure that the application BPEContainer_cellName_ACompanyServer is started too.

Now you are able to run the Dynamic Query sample, simply by opening a Web browser and typing the URL:

```
http://localhost:9081/bpe/Dquery.jsp?customerID=1&itemID=1
```

If the code is integrated into a process Web client (see 11.3.4, “Integration of Dynamic Query with sample application” on page 475), start a new PO process, navigate to the activity details page, where you should find a button on the top of the page that takes you to the Dynamic Query page.

11.8 Performance considerations

As mentioned in Chapter 11, “Dynamic Query” on page 457, dynamic queries can be performed on EJB objects and on CMP fields. The following section discusses some performance considerations regarding performing queries on objects and on fields.

11.8.1 Transactions and Dynamic Query

By default Dynamic Query makes calls against the database with the lowest level of locking possible. Using Dynamic Query, one can either select CMP fields or can select EJB objects. In cases when CMP fields are selected, there is no locking of the data selected. So in this case, you should be aware that the data you are selecting is volatile. Currently there is no way to enforce locking for the records when you are selecting CMP fields. This means that the use of Dynamic Query can be dangerous in some instances. Let’s say you want to make a transaction based on information returned in the data set. Example 11-11 runs a query that returns all the account balances. Then customers pay interest based on the account balance. However, this example is not safe. The reason is that the value of balance can change between when we run the query and when we assign a new balance. Moreover, the account might be deleted between running the query and looking up the account to set the balance. This is because the back-end rows are not locked when a data query runs.

Example 11-11 Selecting CMP fields

```
Select a.id, a.balance from Account a
For (x in resultset)
   if (a.balance  > 1000 && a.balance < 5000) {
            Account A = findByPrimaryKey(a.id);
            A.setBalance(a.balance*1.01);
   }
   if (a.balance  > 5000) {
            Account A = findByPrimaryKey(a.id);
            A.setBalance(a.balance*1.02);
   }
```

So the code for the balance update may be rewritten as Example 11-12 on page 485 to overcome the problem of the possibility of account deletion. After the

data is selected, findByPrimaryKey() method is invoked, and if the account has been deleted between running the query and the findByPrimaryKey(), AccountNotFoundException will be thrown. If the account still exists, the findByPrimaryKey() method has the side effect of locking the account for the transaction. The level of locking is based on the object's Access Intent. So there is still a possibility that the account balance will change during iterations, if the Account Bean Access Intent is optimistic.

Example 11-12 Selecting CMP fields with exception handling

```
Select a.id, a.balance from Account a
For (x in resultset)
try {
   Account A = findByPrimaryKey(a.id);
} catch (AccountNotFoundException e) {

}
if (A.getBalance() > 1000 && A.getBalance() < 5000) {
   A.setBalance(A.getBalance()*1.01);
}
if (A.getBalance() > 5000) {
   A.setBalance(A.getBalance()*1.02);
}
```

The only way to lock a piece of information is to return the instance that relates to it. The only way to implement Example 11-11 on page 484 safely is to select EJB objects instead of selecting CMP fields, and then perform the update, as shown in Example 11-13.

Example 11-13 Selecting EJB instances

```
Select object(a) from Account a
For(x in resultset)
   If (a.getBalance() > 1000 && a.getBalance < 5000){
      a.setBalance(a.getBalance()*1.01);
and so on..
```

The side effect of selecting the EJB object instead of selecting the data fields is that all of the resulting instances are instantiated. This makes the performance much slower because it must return and lock each and every instance of Account. We will use an enormous amount of back-end resources. All work with Account will wait for this transaction to complete. If you have 10000 accounts to update their balances, you will end up instantiating 10000 objects in your runtime, which will bring the system performance down. Moreover, while all this is going on, you will have all these records blocked. The optimal use of query in this case would be to set a lock on the data that prevents updating but allow others to read.

A possible workaround to reduce the number of instances available in memory during runtime is to use the Query bean remote interface. The executeQuery method in the remote interface allows you to divide the resultset into groups, by specifying the cursor position skipRows arguments, and the maxRows argument. But remember that you need to commit after each transaction to be able to release the lock on the selected objects.

One more issue worth noticing when using Dynamic Query is the possibility of using one the bean's method in the query where clause. Let's say we need to retrieve accounts whose balance is greater than 10000. In Example 11-14 the first query selects an Account object, which gets instantiated and used to retrieve the account balance. The second query selects data only. The Account bean is instantiated to evaluate getBalance() in the where clause, so you get a lock on the object based on the Account bean's default Access Intent. But since you are selecting CMP fields, the lock is released at the end of the transaction, not at the end of the query. The problem in this situation is in cases when the Access Intent is optimistic. The getBalance method will get invoked indicating that the balance is greater than 10000, then a moment later that balance might get changed by another user reducing the balance.

Example 11-14 Selecting objects with condition

```
select object(a) from Account a where a.getBalance()>10000
or
select a.id, a.balance from Account a where a.getBalance()>10000
```

11.9 Security considerations

As discussed earlier the Query session bean is responsible for executing queries on entity beans, and on CMP/CMR fields, through the use of its executeQuery() method. Security can be controlled on the use of this bean. That is you can control the use of the Dynamic Query service through granting or denying access to the Query bean. If you want to deny somebody the ability to perform queries, this is done by denying this user access to the Query session bean and its method. Once a user has access to the Query bean and the executeQuery() method, then the user can perform any query. You can control who is permitted access to the remote query bean and the local query bean, but once authorized a user can execute any valid query and return any data in the server. If you need security control over which queries a user can execute, you need to define the queries as finder or select methods and use EJB method authorization. The Dynamic Query service does not have fine-grain security control at this time.

A user may submit a query in which an object is being selected, or data array query. Access to a certain bean information can be controlled by

granting/denying access to the bean and its methods. So if is an object query and the user does not have access to the bean, then the user won't be able to retrieve the desired data. For example, let's say the user submits a query such as:

```
SELECT OBJECT(e) FROM Employee e
```

If the user running the query doesn't have access to the Employee bean, then the user won't be able to execute this query, since this query requires instantiating the Employee bean.

Unfortunately WebSphere does not have security access control for CMP and CMR fields. So let's say that the user doesn't have access to the Employee bean, and submits a query such as:

```
SELECT e.name, e.salary FROM Employee
```

Submitting such a query will return the desired results to the user, although this user is not granted access to the Employee bean, since there is no security on the fields level, and since in such a type of query the Employee bean is not instantiated.

Another example that is worth watching for is a situation such as having two EJBs: an Employee EJB, which a user is not allowed to access, and a Department bean, which the user can access. Let's say the user submits a query such as :

```
SELECT e.name FROM Department d, IN (d.employees) as e WHERE d.deptNum = 10
```

This user will be able to retrieve the list of employee names, although this user is not allowed to access the Employee bean.

If the submitted query has a method call, then the user access privileges on that method are checked first before the method is invoked and the query is executed.

12

Startup Bean

Startup Beans service is an enterprise service that you can use to do some initialization or cleanup before your J2EE application starts and after it stops. This chapter explains the usage of Startup Beans in WebSphere Enterprise V5.

12.1 Introduction

Throughout the evolution of J2EE application design, there was always a need to do initialization or cleanup tasks before the application starts. This can be solved in different ways. In most cases, we end up without a JDNI environment for locating applications resources and also a lack of security.

For this purpose, WebSphere Enterprise introduces the Startup Bean service. A Startup Bean is a special session EJB loaded and executed by the EJB container just before application begins to start. An application will not complete its startup until the Startup Bean has completed the execution.

12.1.1 Why use Startup Beans?

The advantages of using Startup Beans instead of some other proven and already tested mechanisms are as follows:

- Startup Beans run with full security context.
- Startup Beans run within WebSphere's name space. Therefore it uses JNDI to find and use other resources.
- If we compare it to a servlet, the a servlet's init() method is often used as a startup mechanism for starting initialization tasks.

  The Startup Bean also provides a method that executes upon the application's shutdown.

12.2 Design

Startup Beans can be used in various situations, for example:

- Application runtime dependencies as an availability of database or legacy system connection can be checked, before starting up the application.
- A Startup Bean could load data into the cache.
- Similar to data cache is to "warm up" entity beans—by executing finder method and storing the EJB handles for future use. This is also known as "commit option A" EJB cache and can be even more effective together with Async Beans.
- Startup Beans can work well together with asynchronous programming. If an application is using a large cache of some objects, but it can still work without the cache, it is not necessary for the application to wait until the full cache is built. With a Startup Bean we can initiate an Async Bean which populates the cache and starts the application.

- Another proven example using Startup Beans with Asynchronous Beans is independent fast logging. Asynchronous Beans are used for writing application logs to the database and Startup Beans serve as a fast bootstrap mechanism.
- Startup Beans can also be used together with the scheduler service. Before application starts up, we can create a task and schedule it with the Scheduler service.

In our sample scenario, we use a Startup Bean for two tasks:

- For calling the CatalogUpdate process.
- For creating a task and schedule it with the scheduler service.

12.3 Development

In this section we describe how to use Startup Beans.

Steps for using Startup Beans

Startup Beans are just another session EJB in your application. In order to add Startup Beans to an application, do the following:

- Create a Startup Bean according to provided programming API.
- Add it to the application EAR.
- Deploy the EAR in the application server.

Important: In order to successfully compile your application and generate deploy code, startupbean.jar must be added to the Java build path. In WebSphere Studio IE, this file is located in the *WebSphere_Studio_IE_root*>\WAS_EE_V5\lib directory. In WebSphere V5.0 Enterprise, the file is in the <*WebSphere_root*>\lib directory.

Remote and Home interface

A Startup Bean is a user-defined EJB 2.0 session bean. It can be either stateful or stateless. If it is stateful, the same instance is used for start and stop. Otherwise, two instances are created.

It is *mandatory* to use the following home and remote interfaces:

1. EJB home interface must be:

   ```
   com.ibm.websphere.startupservice.AppStartUpHome
   ```

2. EJB remote interface must be:

   ```
   com.ibm.websphere.startupservice.AppStartUp
   ```

This will assure that the WebSphere runtime recognize the bean as a Startup Bean and proceed accordingly. A Startup Bean is *not exposed* in WebSphere's JNDI name space. You can have more than one Startup Bean per EAR or EJB module.

A remote interface defines the following methods that must be implemented:

- **boolean start()**

  This method is called before an application starts. From this method we call all the user-defined initialization code we want to execute.

  The start() method returns true when everything goes right, indicating that the application will start. Returning false or throwing any exceptions indicates that application start will be aborted.

  There are no restrictions on what you can do in this method. The full IBM WebSphere Enterprise programming model is available.

  Because the application will not start before this method completes successfully, we encourage the use of Asynchronous Beans if the initialization task is not necessary to complete fully prior to the application's start.

- **void stop()**

  This method is called before the application stops. More exactly, it is called after you initiate the stop of the application and all the requests have been served. This method can call all the user-defined "cleanup" code we want to execute.

  There are no restrictions on what you can do in this method. The full IBM WebSphere Enterprise programming model is available.

  Exceptions thrown by this method will be logged but ignored, and the application will continue to stop.

Transactional considerations

The startup session bean must use Container Managed Transactions. On stop() and starts() methods, any transactional attribute can be used, *except* TX_MANDATORY, because the methods are not started from a thread that has a transactional context defined. If you use the TX_MANDATORY attribute, an exception is thrown and application start is aborted.

Read more about the behavior of Startup Beans in 12.9, "Runtime environment" on page 502.

Security considerations

There is no specific security setting for Startup Beans.

If you use security enabled for WebSphere V5.0 Enterprise, the default identity for Startup Bean is *system*. This means if you do not specify any deployment-related security settings on the Startup Bean, the identity of the Startup Bean will be the same as the one used for starting WebSphere.

If your Startup Bean uses some secured object that requires a different caller identity, then the security identity must be enabled on the Startup Bean and Run-as on the Startup Bean's start() and stop() methods must be configured to use the correct identity.

Required session bean timeout

Set the timeout to 0. If it is set to any other value, it will be ignored anyway, since startup service automatically checks and sets it to 0.

12.4 Unit test environment

For Startup Beans, there is nothing special to configure in the WebSphere Studio IE test environment. In WebSphere Application Server Enterprise, the Startup Bean service is always enabled. After the development is done, you just publish the project that contains Startup Beans onto the WebSphere Studio IE WebSphere V5.0 Enterprise test environment server.

12.5 Assembly

At the assembly time, you might want to specify the following settings for Startup Beans:

- Startup Bean priorities
- Security identity
- Transactional properties

Startup Bean priorities

If there is more than one Startup Bean in your application, you can specify priorities among them. See 12.9.1, "Priorities when using multiple Startup Beans" on page 504 for more details and explanation about how to set it.

Security identity

You can set security for the bean using Application Assembly Tool. This can be done as shown on Figure 12-1 on page 494.

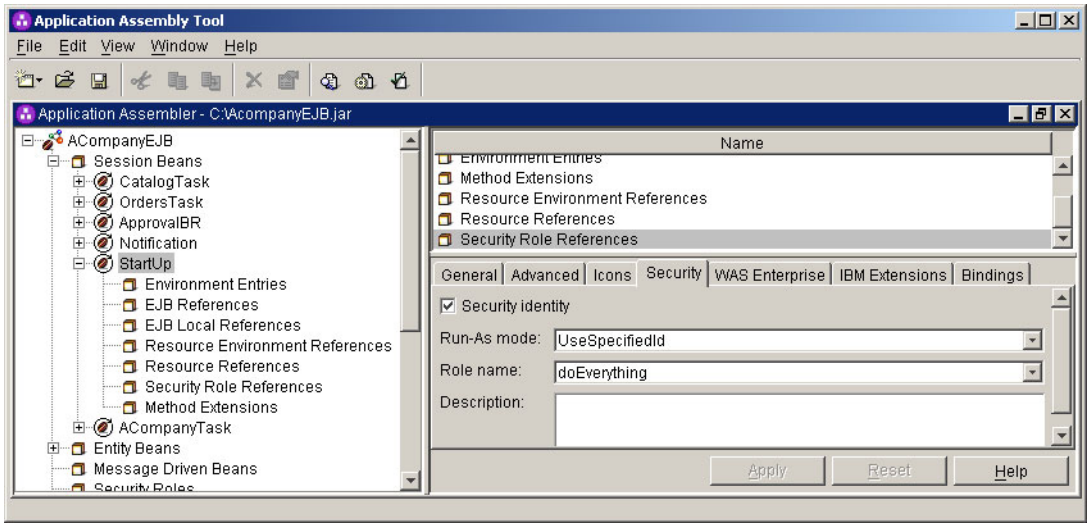


Figure 12-1 Setting Startup Bean security, on bean's scope

Or you can set security just on stop() and start() methods. This is the recommended way. This is not possible within standard J2EE scope. It is a WebSphere extension. Take a look at Figure 12-2 on page 495 to see how to do it with Application Assembly Tool.

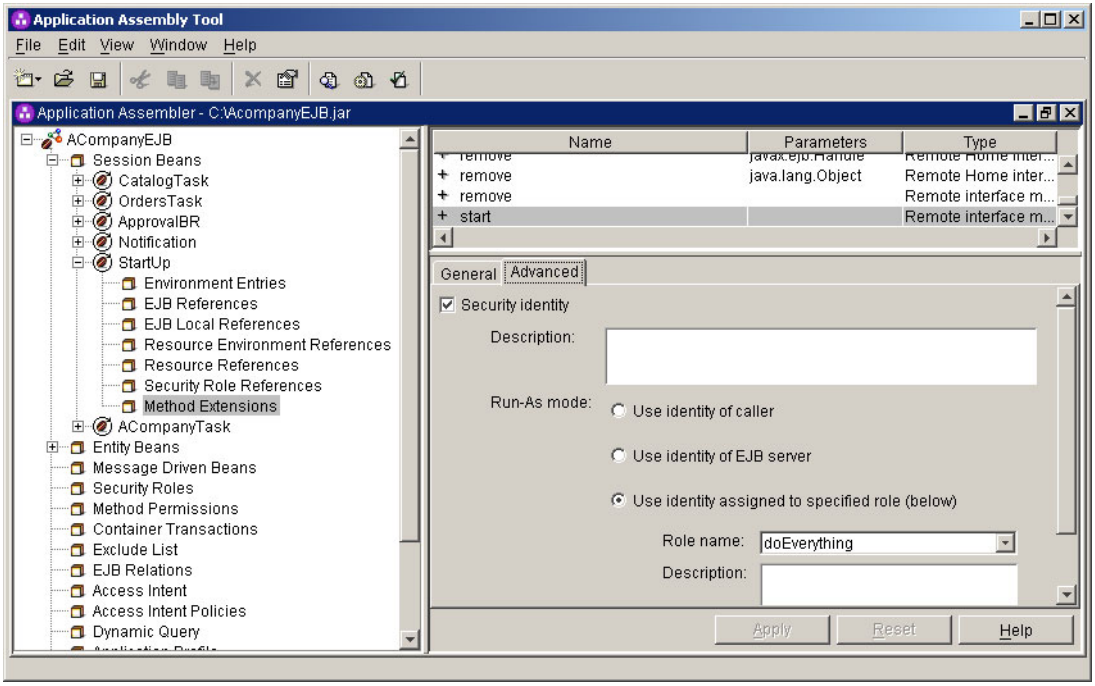


Figure 12-2 Setting security for stop() and start() methods in Startup Bean

Transactional properties

See Figure 12-3 on page 496 for details about settings transactional properties on stop() and start() methods with the Application Assembly Tool.

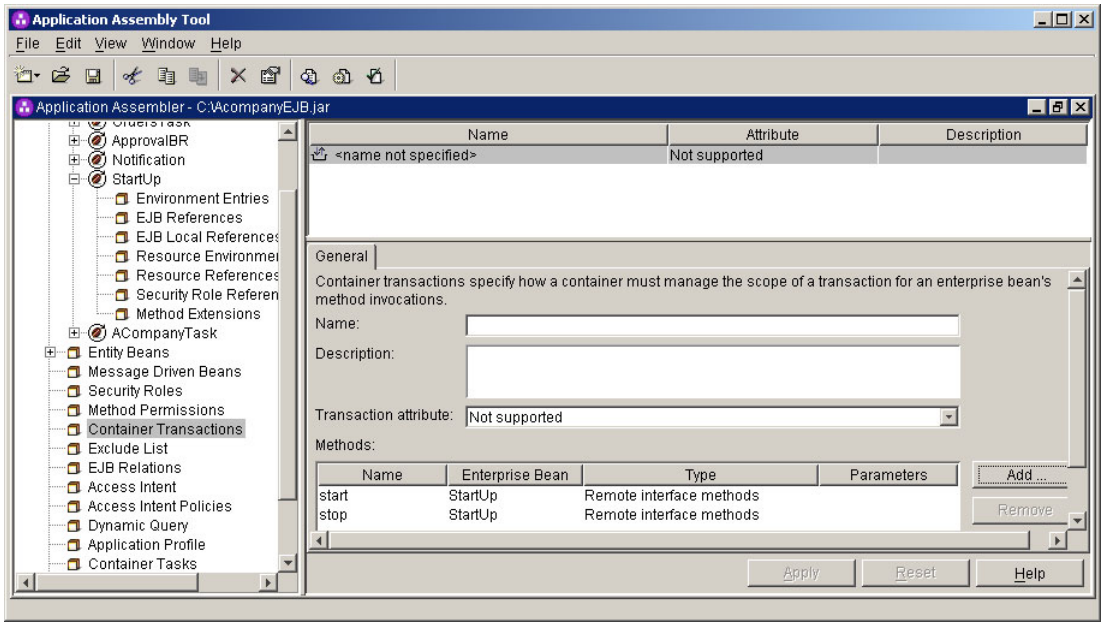


Figure 12-3 Setting transactional properties on start() and stop() methods

JNDI considerations

During application assembly we can specify a JDNI name for the Startup Bean. However, Startup Beans are not exposed in JNDI, and they do not need to be since the only client for Startup Beans is the application server runtime.

12.6 Development

There is no special support from WebSphere Studio IE or any other tool for Startup Beans development. You just create a session bean using the right home and remote interfaces. Make sure that startupbean.jar is in the Java classpath.

12.6.1 Sample scenario

In our sample scenario, we will first use a startup session bean to trigger the CatalogUpdate process.

Prerequisites

Before starting with the development of a Startup Bean for our sample scenario, you need to do the following:

- A database must be created; the database contain the tables necessary for our sample scenario. See “Database” on page 666 for details about database creation.
- For testing our sample with WebSphere Studio IE there must be an instance of WebSphere V5.0 Enterprise server created. See “Development environment” on page 667 for details.
- The startupbean.jar file must be added to the Java build path of the EJB module which will contain the Startup Bean. We use the ACompanyEJB module here. If you are using WebSphere Studio IE select the module, right-click and select Properties, switch to **Java Build Path** tab, choose **Add Variable**, select **WAS_EE_V5**, go to the lib directory and choose **startupbean.jar**. Apply the changes.

Creating a service proxy

The first step is to create a CatalogUpdate service proxy class. It is not a necessary step; it just makes calling CatalogUpdate service easier.

Follow these steps to create a service proxy:

1. Select **Business Integration Perspective**, navigate to the Deployable Services and select **CatalogUpdatePortTypeEJBService.wsdl** under ACompanyEJB. Right-click the service, select **Enterprise Services -> Generate Service Proxy**.
2. Enter data as in Figure 12-4 on page 498, and click **Next**.
3. Select **Client stub** for the proxy style, select **CatalogUpdatePortType** for the operations to include in the proxy, and click **Finish**.

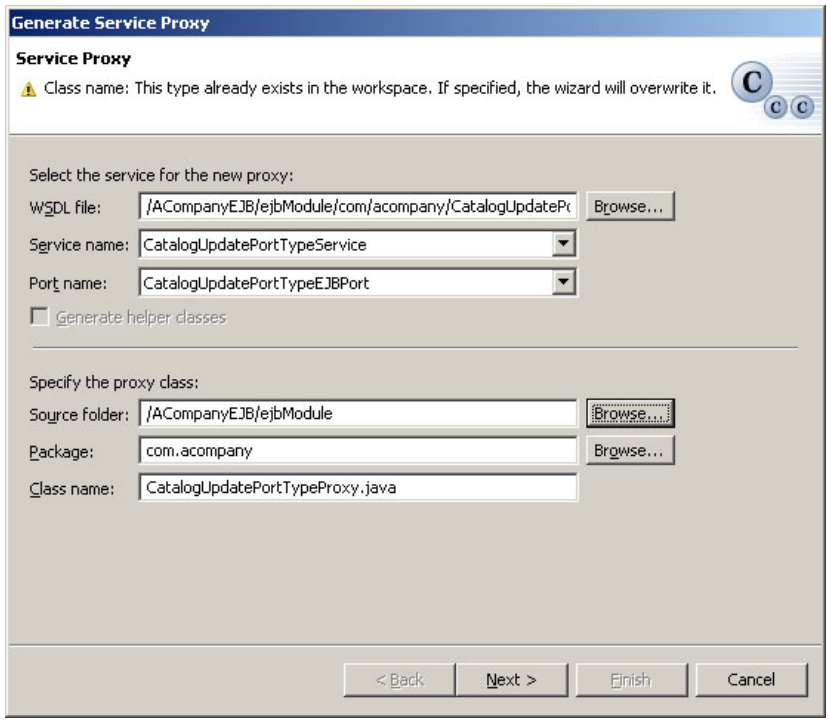


Figure 12-4 Creating service proxy bean

Creating startup session bean

The next step is to create the startup session bean, follow the steps below:

1. In J2EE hierarchy window, go to EJB Modules and right-click the **ACompanyEJB** module. Select **New -> Enterprise Bean**.
2. In the first Enterprise Bean Creation window, make sure that ACompanyEJB is selected for the project, click **Next**.
3. Configure the second Enterprise Bean Creation window as on Figure 12-5 on page 499Click **Next**.

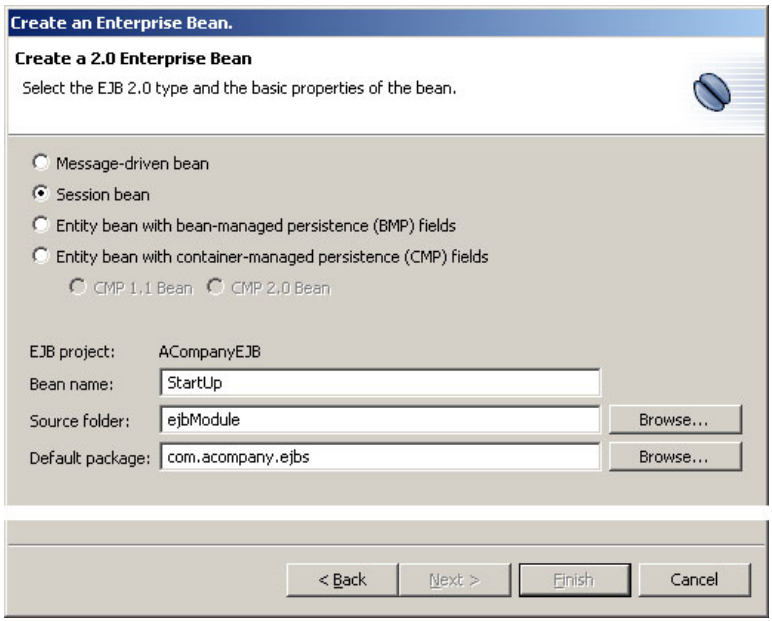


Figure 12-5 Creating startup session bean, part 1

4. Configure the second Enterprise Bean Creation window as in Figure 12-6 on page 500. Make sure that you select **com.ibm.websphere.AppStartUpHome** for the Remote home interface and **com.ibm.websphere.AppStartUp** for the Remote interface. If you correctly added startupbean.jar file to Java build path, the interfaces should be listed in the class selection pop-up window. Click **Finish**.

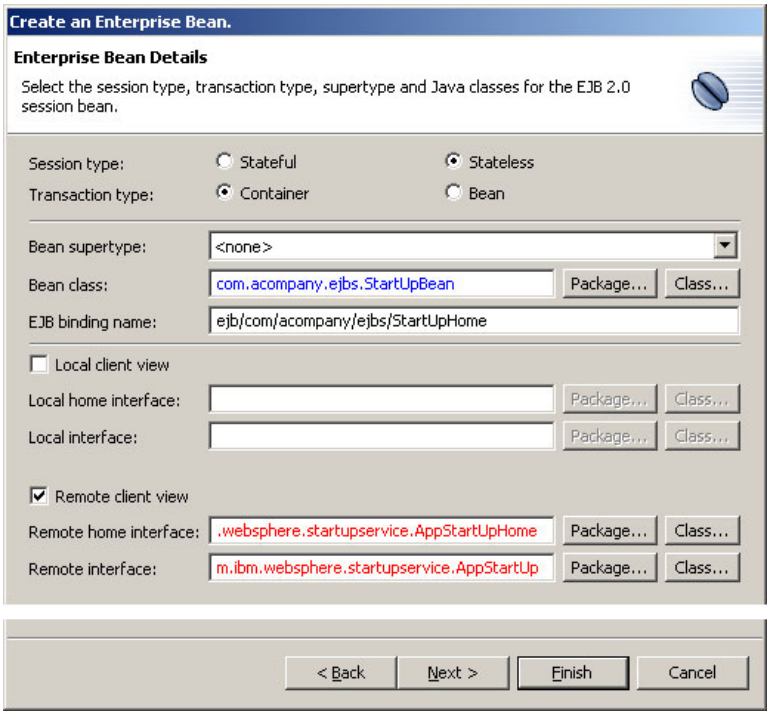


Figure 12-6 Creating startup session bean, part 2

Adding code for the Startup Bean

In this step you will add the code for the start() and stop() methods to the Startup Bean created in the previous section. Open the StartUpBean.java file and add the methods as in Example 12-1.

Example 12-1 StartUpBean.java

```
import com.acompany.CatalogUpdatePortTypeProxy;
...
   public boolean start() {
      try {
         System.out.println("StartUpBean: executing start method");
         CatalogUpdatePortTypeProxy aProxy =
            new CatalogUpdatePortTypeProxy();

         try {
            // this line can be used for quicker testing
            aProxy.InputOperation("1", 0.30, "");
            // the following code should go to production
            // we can only use URL if there is a Web server
            // with the catalog.xml
```

```
            // aProxy.InputOperation("",0,"http://acompany.com/catalog.xml");
          } catch (org.apache.wsif.WSIFException e) {
             System.out.println("StartUpBean: calling CatalogUpdate failed");
          }

       } catch (Exception e) {
          e.printStackTrace();
          return false;
       }
       System.out.println("StartUpBean: start method executed successfully.");
       return true;
    }

    public void stop() {
       System.out.println("StartUpBean: executing stop method");
    }
...
```

What is the start() method doing?

1. In the start() method, we create a new instance of CatalogUpdatePortTypeProxy service proxy.
2. Then we call proxy's InputOperation() method passing ItemID and Price, for which we use hardcoded values "1" and 0.30.
3. The InputOperation() method takes the values and send a message to CatalogUpdate service, which then updates our CATALOG database.

In the source, you can see two InputOperation() method calls. The commented InputOperation() method call is using the CatalogUpdate service in a different way. Instead of passing values for ItemID and Price, you can also pass an URL, under which is an XML file with the updates to the service. In our case we had a predefined XML file served under the local Web server. More details about CatalogUpdate service are in Appendix B, "Sample scenario" on page 665.

In Example 12-2, you can see a part of what we get in test environment server output console. For more detailed traces of startup service execution for the same sample, check 12.10, "Problem determination and troubleshooting" on page 506.

Example 12-2 Startup Bean output

```
ApplicationMg A WSVR0200I: Starting application: ACompany
WebContainer  A SRVE0169I: Loading Web Module: ACompanyWeb.
WebGroup      I SRVE0180I: [ACompanyWeb] [/acompany] [Servlet.LOG]: JSP 1.2 Processor: init
WebGroup      I SRVE0180I: [ACompanyWeb] [/acompany] [Servlet.LOG]: FormLoginServlet: init
WebGroup      I SRVE0180I: [ACompanyWeb] [/acompany] [Servlet.LOG]: FormLogoutServlet: init
WebGroup      I SRVE0180I: [ACompanyWeb] [/acompany] [Servlet.LOG]: SimpleFileServlet: init
```

```
WebGroup      I SRVE0180I: [ACompanyWeb] [/acompany] [Servlet.LOG]: InvokerServlet: init
EJBContainerI I WSVR0207I: Preparing to start EJB jar: ACompanyEJB.jar
EJBContainerI I WSVR0037I: Starting EJB jar: ACompanyEJB.jar
SystemOut     O StartUpBean: executing start method
SystemOut     O com.ibm.bpe.util.MessageLogger: com.ibm.bpe.engine.Engine is using JRas message
logger
SystemOut     O Updated item:1, new price:0.3
SystemOut     O Updated items:1
SystemOut     O StartUpBean: start method executed successfully
ApplicationMg A WSVR0221I: Application started: ACompany
```

12.7 Configuration

For Startup Beans, there is no need for any special configuration in WebSphere Application Server Enterprise. The Startup Bean's behavior is configured during development and assembly time. During deployment, Startup Beans are deployed the same as any EJB.

12.8 Deployment

Startup Beans deploy like any other EJB. It is a part of your application, a part of the JAR file. What makes the difference is that upon application startup WebSphere recognizes them and runs them as a Startup Bean at application startup.

12.9 Runtime environment

This section describes how Startup Bean service behaves from the runtime point of view.

Startup service runtime flow

The following steps represent different states in the life cycle of a Startup Bean in runtime. The diagram is shown in Figure 12-7 on page 504.

1. Application startup

   After the application's EAR is loaded, service looks for all the Startup Beans. It finds them according to the home interface. Then, for each Startup Bean it looks for the presence of wasStartupPriority as a property defined for the startup EJB in the Deployment Descriptor. If there is no priority property defined or it is the wrong type, it would be set to the lowest possible value for the integer.

Startup Beans are ordered according to priority and the startup service takes the bean with the highest priority from the queue. It checks its transactional properties on start() and stop() methods. If a TX_MANDATORY exception is thrown, startup of the application is aborted. Then it finds the actual bean using JNDI and runs its start() method. If there was no problem with the lookup and the start() method returned true upon its execution, the startup service stores the bean handle and proceeds to next Startup Bean in the ordered queue as long there are any Startup Beans left. When it finishes successfully with all the Startup Beans, WebSphere starts the application and its JMS, IIOP and HTTP listeners.

Important: Startup service waits until the start() method finishes its execution. So, if there is a demanding task triggered by the start() method, it could take some time before applications actually start. In those situations, it is recommended that you use Asynchronous Beans to do the task and therefore finish with the start() method as soon as possible.

2. Application shutdown

   In the application shutdown process, right after the containers stopped serving the requests, Startup Beans are taken one by one in the reverse order of precedence regarding the application's startup. For each bean, stop() is executed. Any exception thrown at this stage is ignored, and startup service continues with the next bean. Finally, it shuts down the application and the listeners.

3. EJB Module re-start

   Let's suppose we have an application with more than one EJB module and each contains Startup Bean(s). If one of the EJB modules is stopped and then restarted, the startup service will consider it an application start. Therefore, it will execute the Startup Beans that are part of the restarted EJB module.

4. Application server crash

   If the application server crashes, the Startup Bean stop() method may not be executed. When restarting the server, the start() method will be executed normally.

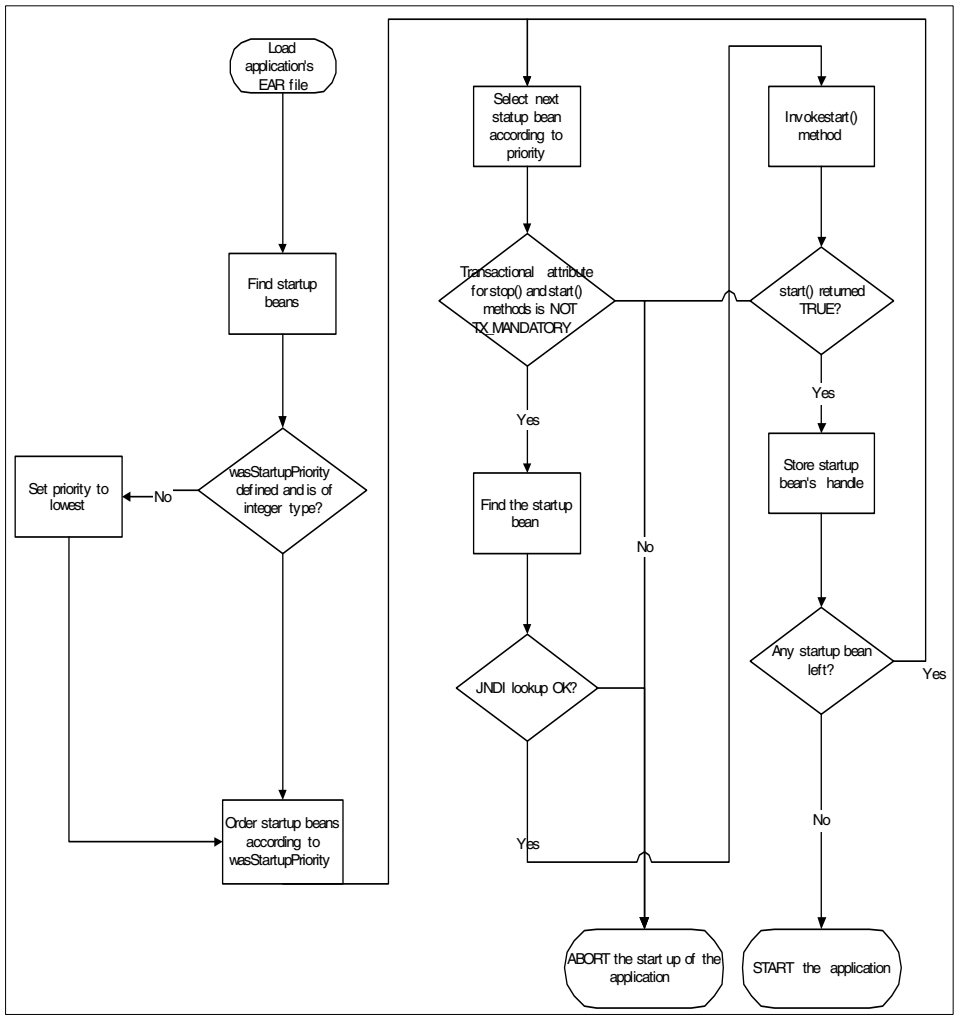


Figure 12-7 Startup Startup Bean service runtime flow on startup

12.9.1 Priorities when using multiple Startup Beans

An application can have more than one Startup Bean. In this situation, priorities must be defined for the beans.

For this purpose, an environment property must be defined on each Startup Bean. The name of this priority must be wasStartupPriority and its type must be java.lang.Integer.

In Figure 12-8, you can see how to add an environment property with the Application Assembly Tool. In Figure 12-9 on page 506, you can see how to add an environment property with WebSphere Studio IE.

The default priority of a Startup Bean is 0. The property can be looked up using JNDI via java:comp/env/wasStartupPriority.

Within one application's scope, a Startup Bean with a higher priority will be executed first. Beans with the same priority will be executed in undefined order. Beans are stopped in the reverse order that they are started in.

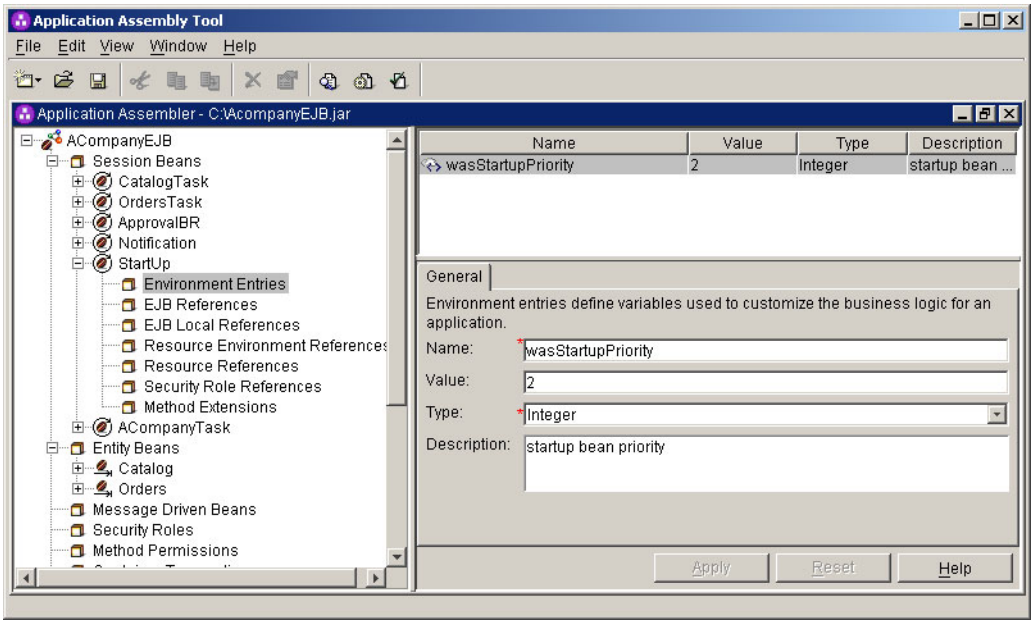


Figure 12-8 Using AAT to add wasStartupPriority environment property to Startup Bean

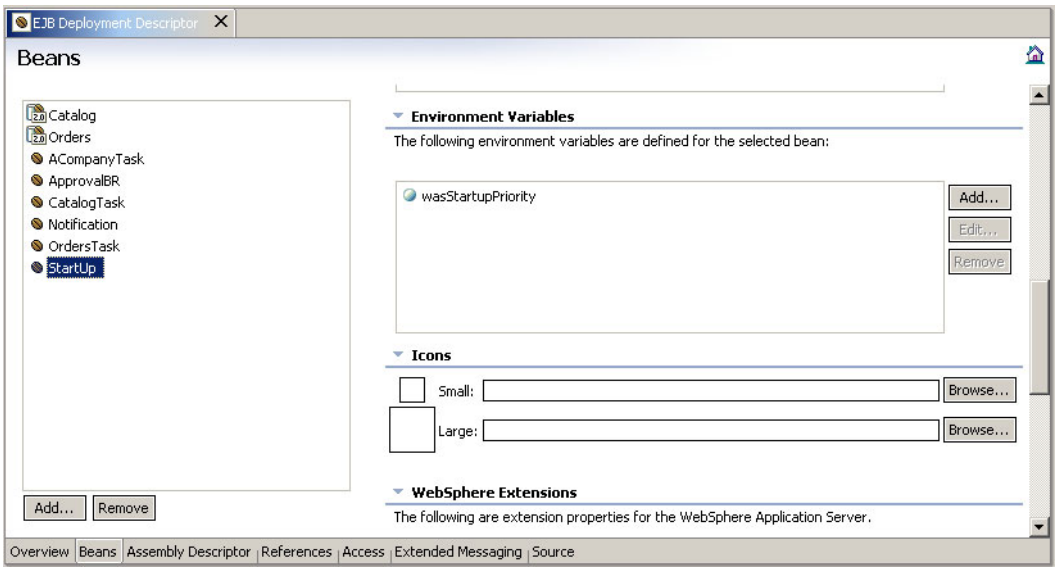


Figure 12-9 Using WebSphere Studio IE to add wasStartupPriority environment property to Startup Bean

12.9.2 Scalability

If we use Startup Beans on clustered applications, multiple clones will run the applications. That is true also for the Startup Bean. Every clone will have a Startup Bean instance of the application. The Startup Bean is invoked for each clone in a cluster.

Startup Bean service of a clone is not aware of other clones and it does not synchronize any actions between instances of Startup Beans among clones. Therefore, it is the programmers' responsibility to take appropriate actions to avoid dead locks or data inconsistencies.

12.10 Problem determination and troubleshooting

You will quickly know if there is a configurational problem with Startup Beans you are using: the application simply will not start. Start will be aborted by the startup service and it will continue to start other applications.

As a first step, check if the transactional and security settings are correct.

If you encounter problems using Startup Beans and you suspect that the problem source is somewhere within the startup service, then you can additionally use the following trace string to monitor what is happening in the execution time.

```
com.ibm.ws.startupservice.*=all=enabled
```

In Example 12-3, you can see the trace output for the startup service for our sample scenario described in Appendix B, “Sample scenario” on page 665.

Example 12-3 Sample trace output

```
ApplicationMg A WSVR0200I: Starting application: ACompany
StartUpServic > stateChanged
  com.ibm.ws.runtime.deploy.DeployedObjectEvent[source=ACompany [class
com.ibm.ws.runtime.component.DeployedApplicationImpl]]
StartUpServic < stateChanged
StartUpServic d handleModuleMetaData:App name is ACompany
StartUpServic > stateChanged
  com.ibm.ws.runtime.deploy.DeployedObjectEvent[source=ACompanyWeb.war [class
com.ibm.ws.runtime.component.DeployedModuleImpl]]
StartUpServic < stateChanged
WebContainer  A SRVE0169I: Loading Web Module: ACompanyWeb.
WebGroup      I SRVE0180I: [ACompanyWeb] [/acompany] [Servlet.LOG]: JSP 1.2 Processor: init
WebGroup      I SRVE0180I: [ACompanyWeb] [/acompany] [Servlet.LOG]: FormLoginServlet: init
WebGroup      I SRVE0180I: [ACompanyWeb] [/acompany] [Servlet.LOG]: FormLogoutServlet: init
WebGroup      I SRVE0180I: [ACompanyWeb] [/acompany] [Servlet.LOG]: SimpleFileServlet: init
WebGroup      I SRVE0180I: [ACompanyWeb] [/acompany] [Servlet.LOG]: InvokerServlet: init
StartUpServic > stateChanged
  com.ibm.ws.runtime.deploy.DeployedObjectEvent[source=ACompanyWeb.war [class
com.ibm.ws.runtime.component.DeployedModuleImpl]]
StartUpServic < stateChanged
EJBContainerI I WSVR0207I: Preparing to start EJB jar: ACompanyEJB.jar
StartUpServic d handleModuleMetaData:App name is ACompany
StartUpServic d Skipping bean. Not a startup bean:ACompany#ACompanyEJB.jar#CatalogTask
StartUpServic d Skipping bean. Not a startup bean:ACompany#ACompanyEJB.jar#OrdersTask
StartUpServic d Skipping bean. Not a startup bean:ACompany#ACompanyEJB.jar#ApprovalBR
StartUpServic d Skipping bean. Not a session bean:ACompany#ACompanyEJB.jar#Catalog
StartUpServic d Skipping bean. Not a session bean:ACompany#ACompanyEJB.jar#Orders
StartUpServic d Skipping bean. Not a startup bean:ACompany#ACompanyEJB.jar#Notification
StartUpServic d Found startup bean StartUp in ACompany module ACompanyEJB.jar
StartUpServic d Overriding Stateful Session Timeout to zero.  Configured value=600
StartBeanInfo d Transaction Attribute for method: start = TX_NOT_SUPPORTED
StartBeanInfo d Transaction Attribute for method: stop = TX_NOT_SUPPORTED
StartUpServic d Skipping bean. Not a startup bean:ACompany#ACompanyEJB.jar#ACompanyTask
StartUpServic > stateChanged
  com.ibm.ws.runtime.deploy.DeployedObjectEvent[source=ACompanyEJB.jar [class
com.ibm.ws.runtime.component.DeployedModuleImpl]]
StartUpServic < stateChanged
EJBContainerI I WSVR0037I: Starting EJB jar: ACompanyEJB.jar
StartUpServic > stateChanged
  com.ibm.ws.runtime.deploy.DeployedObjectEvent[source=ACompanyEJB.jar [class
com.ibm.ws.runtime.component.DeployedModuleImpl]]
StartUpServic < stateChanged
```

```
StartUpServic > stateChanged
  com.ibm.ws.runtime.deploy.DeployedObjectEvent[source=ACompany [class
com.ibm.ws.runtime.component.DeployedApplicationImpl]]
StartUpServic d EAR Started
StartUpApplic > starting ACompany
StartUpModule > start
  ACompanyEJB.jar
StartUpModule d Starting Bean: ejb/com/acompany/ejbs/StartUpHome
StartBeanInfo > start
  ejb/com/acompany/ejbs/StartUpHome
StartBeanInfo > getStartUpBean
StartBeanInfo > getStartUpHome
StartBeanInfo < getStartUpHome
StartBeanInfo < getStartUpBean
SystemOut     O StartUpBean: executing start method
SystemOut     O com.ibm.bpe.util.MessageLogger: com.ibm.bpe.engine.Engine is using JRas message
logger
SystemOut     O Updated item:1, new price:0.3
SystemOut     O Updated items:1
SystemOut     O StartUpBean: start method executed successfully
StartBeanInfo d Startup Bean start method called
  ejb/com/acompany/ejbs/StartUpHome
  true
StartBeanInfo < start
  ejb/com/acompany/ejbs/StartUpHome
StartUpModule d StartUpBean.start returned true: ejb/com/acompany/ejbs/StartUpHome
StartUpModule < start
  ACompanyEJB.jar
StartUpApplic < starting ACompany
StartUpServic < stateChanged
ApplicationMg A WSVR0221I: Application started: ACompany
```

13

Scheduler service

The Scheduler service is a WebSphere Enterprise mechanism that is responsible for starting actions at specific times or intervals. This chapter describes the usage of the Scheduler service.

13.1 Introduction

Until now, there has been no functionality providing time-driven actions in WebSphere. As always, there was an option to build a proprietary mechanism, but in the end it would not be well integrated into the WebSphere runtime.

With the Scheduler service, the actions can be scheduled to happen only once, some time in the future, or on a recurring basis or at regular intervals. It can also receive notifications about task activity.

Why use the Scheduler service?

Beside the better integration, here is a list of the advantages of using the Scheduler service:

- Administration

  Scheduler service administration is consistent with WebSphere resource management.

- Persistence and transactional robustness

  Scheduler tasks can be persistent. Basically this means that the task definitions are stored in a relational database and can be executed for indefinite repetitions and arbitrary long time periods.

- Scalability

  The Scheduler service can be clustered using workload management for performance and availability.

- Calendar independence

  Each task is executed according to the specified calendar. You can provide your own calendar classes that suit your local business calendar.

- Design flexibility

  Scheduler tasks can be EJB-based tasks or they can be triggered using JMS messages.

- Notifications

  The Scheduler can perform notification actions on various task activity events.

13.2 Design

The Scheduler is useful any time you need to ensure that certain actions are taken on a regular basis. For example:

- Schedule regular backups, cleanups, night-time, or “batch” processing.

- Implement automatic, non-user driven activities, as reminders, to-do actions, etc.

Tasks can be persistent or nonpersistent.

In our sample scenario we will use the Scheduler for periodical calling of the CatalogUpdate process to update the catalog from external resources.

13.3 Development

In the following text, we describe how to use the Scheduler service from your application.

13.3.1 Scheduler API

The Scheduler API is contained in the com.ibm.websphere.scheduler package. Check the complete description of the Scheduler API in the WebSphere Enterprise InfoCenter:

http://publib7b.boulder.ibm.com/wasinfo1/en/info/ee/javadoc/ee/com/ibm/websphere/scheduler/package-summary.html

13.3.2 Steps for using the Scheduler service

There are no tools that provide help creating and scheduling tasks. All must be done programmatically in your J2EE application. The following steps must be done when using the service:

1. Configure the Scheduler service.
2. Develop the Scheduler client code:
   a. Define tasks
   b. Define notifications
   c. Define calendar
   d. Find Scheduler and scheduling tasks
   e. Manage scheduled tasks
3. Add the code to your application.
4. Deploy.

Configure the Scheduler service

The first step is to have the Scheduler service and its prerequisites configured as the work manager and Scheduler database. This can be done using the

WebSphere administration. Refer to “Scheduler configuration using the Administrative Console” on page 530 for more details.

Important: Remember the JNDI name you used for the Scheduler service configuration.

Define task

Defining a task consists of creating a task information object and creating a task action object.

Task information

A task is defined by constructing a task information object. This object contains all the task information such as start times, repeat interval, the calendar to use, start interval, state, and persistence.

Generic task information is defined by the TaskInfo interface. Additionally two interfaces extend TaskInfo:

- BeanTaskInfo

  Used when the target object is an EJB session bean.

- MessageTaskInfo

  Used to create a scheduled task that sends a JMS message to either a queue or topic.

WebSphere Enterprise provides implementation for both interfaces described above. Programmatically you create an instance of these two objects by executing the appropriate create() method of the Scheduler service.

Task

The Task information object also specifies the target action and the executable action of the task. This target can be one of the following:

- JMS message

  A message that is sent by JMS to a queue or topic. In this case you use MessageTaskInfo object for target action. With this object you can specify all the JMS-related properties (connection factory, queue, and message).

- Session EJB

  Within the application EAR, there is a session EJB that implements the IBM-provided home and remote interface. A method of this bean is then called when the task fires. It should be implemented as an EJB 2.0 stateless session bean. You must use the TaskHandlerHome interface for EJB’s home interface and the TaskHandler interface for EJB’s remote interface. When the Scheduler executes the action, it runs TaskHandler EJB’s process() method.

- A user-supplied JMS message is queued or published by WebSphere JMS service to the user-supplied queue or topic. In this case you use the MessageTaskInfo object for the target action. With this object, you specify all the JMS-related properties (connection factory, queue, and message).

Note: The action that is executed by a scheduled task can be either a special session EJB or a specific JMS message.

Define notification

Note: Defining notification is optional. You can schedule tasks without using the notification mechanism.

Various events regarding task status can be generated by the Scheduler service, for example when the task is scheduled, upon successful execution of the task, upon cancelled execution, upon deleting, etc. When using the notification mechanism, the following must be defined:

- **Notification mask** represents the mask of which event will be monitored and upon which we want to be notified. This is done by creating and configuring an instance of the TaskNotificationInfo class.
- **Notification action** is performed upon the occurrence of the specified events. The action is specified by an EJB 2.0 session bean. You must use the IBM provided NotificationSinkHome interface for EJB's home interface and NotificationSink interface for EJB's remote interface.

  The handleEvent() method is called when the notification is fired.

When you have both the notification event mask and notification action bean defined, you set the notification via the TaskInfo object by using its setNotificationSink() method.

Define user calendar

Note: Using user-defined calendars is optional. You can schedule tasks without providing a calendar. WebSphere implements two calendars that cover the majority of the needs. These calendars are used by default.

When creating a task information object, there is an option to specify user-defined calendar(s). This is done by calling the setUserCalendar(String homeJNDIName, String specifier) method on the TaskInfo object.

This can be very useful, since different types of businesses rely on different calendars for calculations. Even within the same business, we can have different

calendars. For example national holidays differ from country to country, so the term “next business day” according to a given date can have a different value.

Calendar is represented by a stateless session bean, which must use the IBM-provided UserCalendarHome interface for EJB’s home interface and UserCalendar interface for EJB’s remote interface. A calendar bean can contain more than one calendar. In the bean, you need to implement the following remote methods:

- void validate(String calendar, String delta)

  It checks if a given calendar name matches any of the calendars that are defined in the bean. It also checks if the delta specifier is valid for the given calendar.

- Date applyDelta(Date baseTime, String calendar, String delta)

  This method takes the given baseTime date and then, according to the given calendar implementation and delta interval, calculates the exact time of execution.

These two methods are used by the Scheduler service when calculating the firing interval.

Default user calendar

Since the Scheduler service always use a calendar, WebSphere Enterprise V5 already implements one.

If no calendar is specified when defining a TaskInfo object, then the default calendar implementation (actually its simple arithmetic calendar) will be used.

The other choice is to do a lookup for the default UserCalendar using its JNDI name: com/ibm/websphere/scheduler/calendar/DefaultUserCalendarHome. Then you can reuse it in your own calendar.

The default UserCalendar supplied with the WebSphere Scheduler implements two types of calendars:

- **CRON-like calendar**

  The calendar can only be set using setUserCalendar() on the TaskInfo object, specifying the default calendar’s JNDI name and “CRON” as a calendar identifier.

  The CRON calendar’s time interval argument looks like a UNIX CRON-based interval. It is a string consisting of the following substrings:

  “`second minute hourOfDay DayOfMonth Month DayOfWeek`**”**

  Refer to the Scheduler API in WebSphere Enterprise V5 InfoCenter for more details.

- Simple arithmetic calendar

  By default this calendar is used. It simply calculates the delta interval according to a specified interval, which depends on how the starting time is set in the TaskInfo object.

  Calendar's time interval argument is a string consisting of NumberTimeinterval substrings, where Number is an integer and Timeinterval is one the following: **"**`seconds`**"**, **"**`minutes`**"**, **"**`hours`**"**, **"**`days`**"**, **"**`months`**"**, **"**`years`**"**.

  For example a valid string for an arithmetic calendar would be:

  "`10days 3minutes 1months`**"**

  Note: How you order the substrings does not depend on the Timeinterval value, but be careful because the order directly affects the final date result. Parsing the string and applying values to the time base goes from left to right. In some cases we can get different values for the same time base, using the same NumberTimeinteval substrings but in a different order.

Find the Scheduler service and schedule the task

The Scheduler service is exposed in JNDI. Programmatically, you use JDNI lookup to find the Scheduler service on the WebSphere Enterprise V5 application server and then schedule the task.

Restriction: The Scheduler is available only to server-side components where the Scheduler daemon is running. If a Scheduler resource is defined at the node level, for example, all the application servers on that node have access to that Scheduler and will run its daemon. You can selectively disable the daemon on an application server.

Manage scheduled tasks

Once tasks are scheduled, you can manage them through the Scheduler service. Once you obtained the service using the JNDI, you can do the following with already created tasks:

- Suspend a task. The task does not run until it has been resumed.
- Resume a previously suspended task.
- Cancel a task.
- Permanently delete a completed or cancelled from the persistent store.
- Return the current status of the task.

Other methods, such as finding a task or getting a task's ID, are also implemented. Refer to the Scheduler API for more details.

Note: Tasks are managed only programmatically by using the Scheduler API within the code. There is no administration available for created tasks, so be aware of all the created tasks. Once a persistent task is created, it will exist and run until we (programmatically) delete it from the database.

13.4 Unit test environment

The Scheduler service must be configured in WebSphere Studio IE test environment for the WebSphere Enterprise V5 test server. In WebSphere Studio IE, there is a limited set of administration tasks for the test server available through the GUI. In particular, there is no Scheduler administration using the server editor tool in WebSphere Studio IE. You have to have the test environment server running and the Administrative Console enabled for the server. Once the server is running, use a Web browser to access the Administrative Console where you can configure the Scheduler service the same way as for the runtime application server. See 13.8, “Configuration” on page 527.

13.5 Assembly

The following section discusses the assembly-related actions for the Scheduler service.

In the application code, local name references are used for the Scheduler service so before you can use the Scheduler you have to tell your application where they are in the JNDI name space. You can use the Application Assembly Tool by doing the following:

1. In the Application Assembly Tool, open your application EAR file. Select the module from where the Scheduler is used. If there is more than one module, you will have to create a resource environment reference for each one. That means each module—client, EBJ or Web module—from which you use the Scheduler service needs a resource environment reference for the Scheduler.
2. If you have selected an EJB module, you have to find the bean that uses the Scheduler service and then expand it. Select **ACompany -> EJB Modules -> ACompanyEJB -> Session Beans -> StartUp**. If you have selected a client application module, the resource environment property is already in the subroot directory. Right-click **Resource Environment References** and select **New**.
3. On the General tab, enter the name you are using in your code as a local reference, for example `ACompanyScheduler`. For the type, enter `com.ibm.websphere.scheduler.Scheduler`.

4. On the Bindings tab, for the JNDI name enter the JNDI name that you specified when creating the new Scheduler service in WebSphere, for example `ACompanyScheduler`.
5. Save the EAR and close it.

13.6 Building and tools

There is no special support from WebSphere Studio IE or any other tool for the Scheduler. You just create a session bean using the correct home and remote interfaces. Make sure that the schedulerclient.jar file is in the Java classpath. See “Prerequisites” on page 517 for more details.

13.7 Sample scenario

In our sample scenario of using the WebSphere Enterprise Scheduler service, we will use the session bean option for the target action. The development part consists of developing a task session bean and developing the code that will get the Scheduler, create a task information object, set the properties and the target action, and finally schedule the task.

Our task bean will execute exactly the same code as in the Startup Bean in Chapter 12, “Startup Bean” on page 489. Every time the Scheduler service executes the task, it updates the catalog database for the application.

We will use already created Startup Beans for assigning the task for the Scheduler service.

Prerequisites

You need to meet the following prerequisites:

- The application database needs to be in place. See “Database” on page 666 for details.
- For testing the sample with WebSphere Studio IE, a test server needs to be created and configured. For more information, refer to “Development environment” on page 667.
- The schedulerclient.jar file needs to be in the Java build path of the EJB module that will contain the Startup Bean. It is the ACompanyEJB module.
- Since the Scheduler sample reuses the Startup Bean sample, make sure that you have already done that sample in WebSphere Studio IE.

- The Scheduler service will be used from the Startup Bean. For the startup EJB, you need to define a resource reference that points to the Scheduler service.

Creating the task session EJB

Take the following steps to create the task session EJB:

1. In the J2EE hierarchy window, select **EJB Modules** and right-click the **ACompanyEJB** module. Select **New -> Enterprise Bean**.
2. In the first Enterprise Bean Creation window, make sure that ACompanyEJB is selected for the project, and click **Next**.
3. Configure the second Enterprise Bean Creation window as in Figure 13-1. Click **Next**.

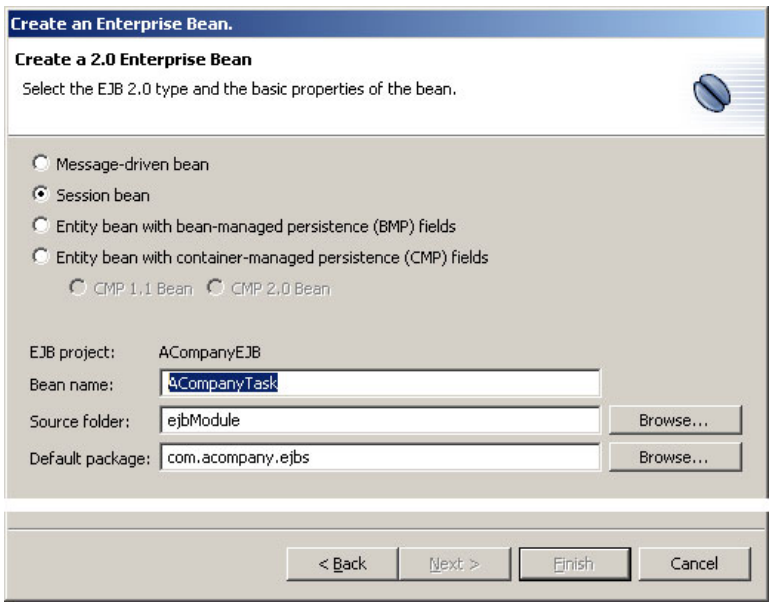


Figure 13-1 Creating the task session EJB, screen 1

4. Configure the second Enterprise Bean Creation window, as in Figure 13-2 on page 519. Make sure that you select **com.ibm.websphere.scheduler.TaskHandlerHome** for the Remote home interface and **com.ibm.websphere.scheduler.TaskHandler** for the Remote interface. Click **Finish**.

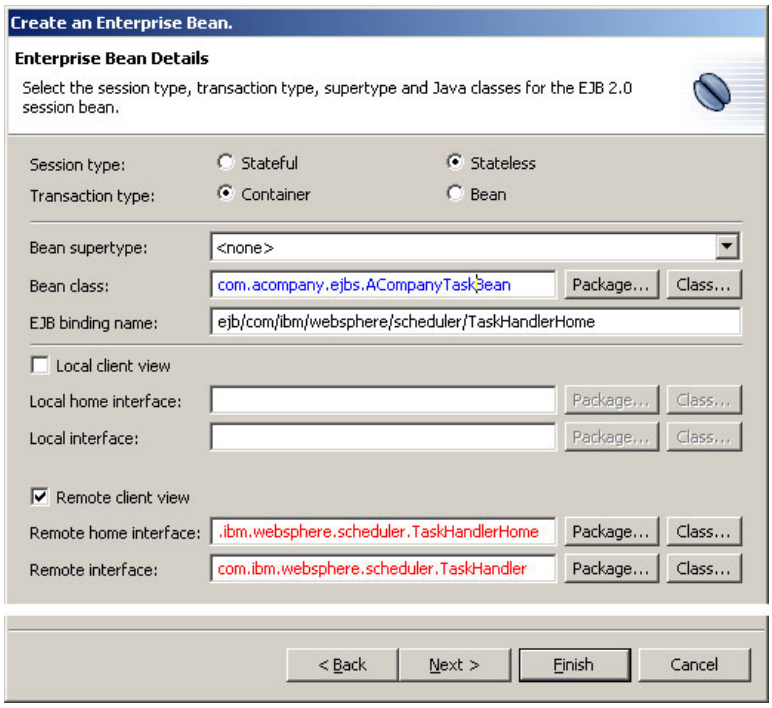


Figure 13-2 Creating the task session EJB, screen 2

Adding code for the Task Handler bean

The next step is to add the code for the process method of ACompanyTask. Open ACompanyTaskBean.java and add the method as in Example 13-1.

Example 13-1 ACompanyTaskBean.java

```
import com.acompany.CatalogUpdatePortTypeProxy;
import com.ibm.websphere.scheduler.TaskStatus;
...
public void process(TaskStatus status) {
   System.out.println("ACompanyTaskBean: executing process");
   try {
      CatalogUpdatePortTypeProxy aProxy = new CatalogUpdatePortTypeProxy();
      try {
         // this line can be used for testing
         // we can only use URL if there is a Web server with the catalog.xml
         aProxy.InputOperation("1", 0.30, "");
         // the following code should go to production
         // aProxy.InputOperation("",0,"http://acompany.com/catalog.xml");
      } catch (org.apache.wsif.WSIFException e) {
         System.out.println("ACompanyTaskBean: calling CatalogUpdate failed");
      }
```

```
   }
   catch (Exception e) {
      e.printStackTrace();
   }
   System.out.println("ACompanyTaskBean: process executed successfully");
}
```

In the process method we do the same as in the previous case with the Startup Bean. We create a new instance of the service proxy and then call its InputOperation method. The InputOperation method takes the values and sends a message to CatalogUpdate service, which then updates the CATALOG table in the database.

Note: Originally, the InputOperation method passes a URL to the external catalog. For faster and easier testing, use the line with one item update.

The commented InputOperation method call uses the CatalogUpdate service in a different way. You can also pass a URL under which is an XML file with the updates to the service. In our case we had a predefined XML file served from the local Web server. More details about CatalogUpdate service can be found in 3.4.1, “Catalog update business process” on page 29.

Adding the Scheduler service enablement code

We will use the Startup Bean used in 12.3, “Development” on page 491. In the start method, we will find a Scheduler and deploy the previously created task. Open StartUpBean.java and add the scheduleTask method as in Example 13-2.

Example 13-2 Adding scheduleTask() method to StartUpBean.java

```
import com.ibm.websphere.scheduler.BeanTaskInfo;
import com.ibm.websphere.scheduler.Scheduler;
import com.ibm.websphere.scheduler.TaskHandlerHome;
import com.ibm.websphere.scheduler.TaskStatus;
...
boolean scheduleTask(long startInterval, int repeats, long repeatInterval,
String taskName) {
   boolean result=false;
   try {
      //Get the Scheduler
      Context initialContext = new InitialContext();
      System.out.println("StartUpBean.scheduleTask: Getting the Scheduler");
      Object schedulerObj =
initialContext.lookup("java:comp/env/ACompanyScheduler");
      Scheduler scheduler = (Scheduler)
javax.rmi.PortableRemoteObject.narrow(schedulerObj, Scheduler.class);
      // Get the task handler which will be called when the task runs.
```

```
      System.out.println("StartUpBean.scheduleTask: Getting the task
handler");
      Object taskHandlerObj =
initialContext.lookup("java:comp/env/LocalTaskHandlerReference");
      TaskHandlerHome taskHandlerHome =
         (TaskHandlerHome) PortableRemoteObject.narrow(taskHandlerObj,
TaskHandlerHome.class);
      // Create our Schedule Task Info for our ACompanyTask task handler
      System.out.println("StartUpBean.scheduleTask: Creating the task");
      BeanTaskInfo taskInfo = scheduler.createBeanTaskInfo();
      taskInfo.setTaskHandler(taskHandlerHome);
      taskInfo.setStartTimeInterval(startInterval + "minutes");
      taskInfo.setRepeatInterval(repeatInterval + "minutes");
      taskInfo.setNumberOfRepeats(repeats);
      taskInfo.setName(taskName);
      //set the name of the task which is associated to a TaskID
      TaskStatus status = scheduler.create(taskInfo);
      System.out.println("StartUpBean.scheduleTask: Task submitted");
      System.out.println("StartUpBean.scheduleTask: Task ID is: " +
status.getTaskId());
      System.out.println("StartUpBean.scheduleTask: The name of the Task is: "
+ status.getName());
      System.out.println("StartUpBean.scheduleTask: Task status is: " +
status.getStatus());
      result=true;
   }
   catch (Exception e) {
      System.out.println("StartUpBean.scheduleTask: failed");
      e.printStackTrace();
   }
   return (result);
}
```

What the scheduleTask method does:

1. Finds the configured Scheduler service using JNDI and creates an instance.
2. Finds the Task Handler bean using JNDI and creates an instance.
3. Creates a task and submits it to the Scheduler service.

The next step is to modify the start method of the Startup Bean. See Example 13-3.

Example 13-3 Modifying start() method in StartUpBean.java

```
public boolean start() {
   try {
      System.out.println("StartUpBean: executing start method");
      CatalogUpdatePortTypeProxy aProxy = new CatalogUpdatePortTypeProxy();
```

```
      try {
         // this line can be used for testing
         // we can only use URL if there is a Web server with the catalog.xml
         aProxy.InputOperation("1", 0.30, "");
         // the following code should go to production
         // aProxy.InputOperation("",0,"http://acompany.com/catalog.xml");
      }
      catch (org.apache.wsif.WSIFException e) {
         System.out.println("StartUpBean: calling CatalogUpdate failed");
      }
   }
   catch (Exception e) {
      e.printStackTrace();
      return false;
   }
   System.out.println("StartUpBean: calling scheduleTask");
   return scheduleTask(1, 5, 1, "TheACompanyTask");
}
```

As you can see, we are still calling the CatalogUpdate from the start method of the Startup Bean. But, after we also call scheduleTask and schedule ACompanyTask.

The arguments used in the call are `(1, 5, 1, "TheACompanyTask")`, which means 1 minute of startup delay, 5 repetitions, 1 minute between each repetition and `TheACompanyTask` is a name we chose for our task. Because we do not specify any specific calendar with the task, all the time values are relative to the default calendar.

User-defined calendar

In this section we create our own user-defined calendar. It will not be used in this sample scenario. It is used in the Internationalization sample. Find the details about the Internationalization sample in Chapter 16, “Internationalization (i18n) service” on page 583.

This sample reuses the default calendar and adds some hooks for specific settings. So, when the specified time interval will not meet our criteria, our calendar will pass the control to the default calendar.

Create the bean

Create the bean with the following steps using WebSphere Studio IE:

1. In J2EE hierarchy window, navigate to the EJB Modules and right-click the **ACompanyEJB** module, then select **New** -> **Enterprise Bean**.
2. In the first Enterprise Bean Creation window, make sure that ACompanyEJB is selected for the project, and click **Next**.

3. Configure the second Enterprise Bean Creation window, as in Figure 13-3. Click **Next**.

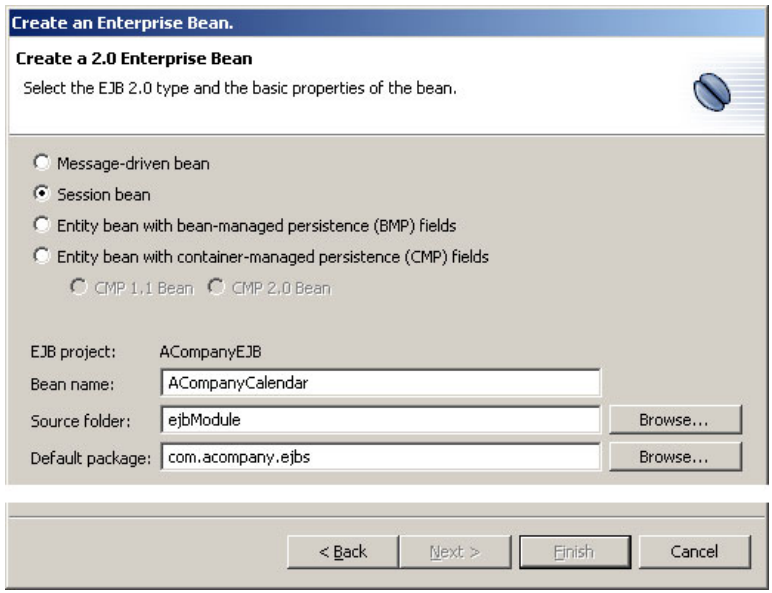


Figure 13-3 Creating UserCalendar bean, first window

4. Configure the second Enterprise Bean Creation window as in Figure 13-4 on page 524. Make sure that you select the **com.ibm.websphere.scheduler.UserCalendarHome** for the Remote home interface and **com.ibm.websphere.scheduler.UserCalendar** for the Remote interface. Click **Finish**.

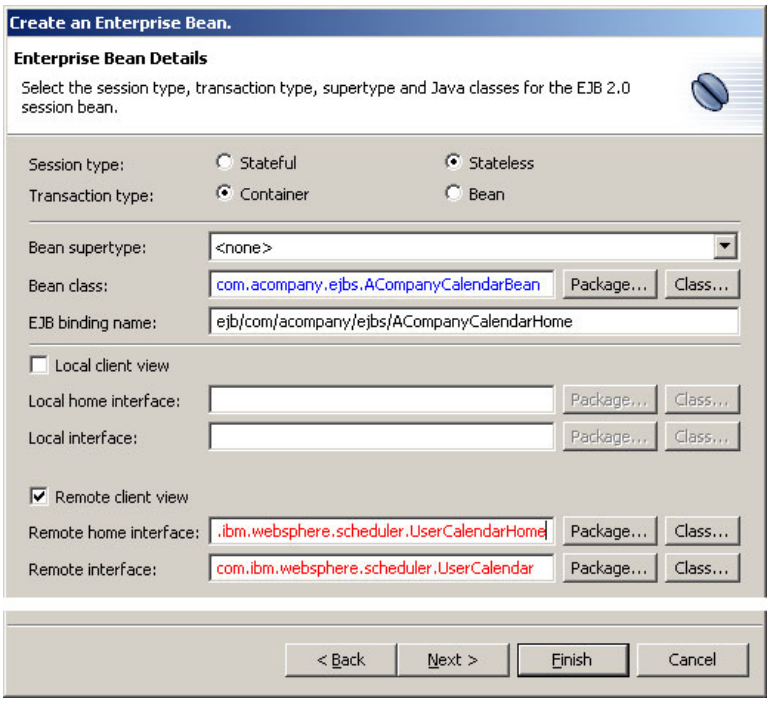


Figure 13-4 Creating UserCalendar bean, second window.

Add resource environment reference

Since we use the default calendar bean in our calendar bean, a resource reference must be added to the Deployment Descriptor of EJB module. Take the following steps:

1. Open the Deployment Descriptor of ACompanyEJB project by double-clicking **ACompanyEJB**. Switch to the References tab, select the **ACompanyCalendar** bean and click **Add**.
2. Select **Resource environment reference** and click **Next**. Enter `UserDefaultCalendar` in the Name field, and in the Type field enter `com.ibm.websphere.scheduler.calendar.DefaultUserCalendarHome`. Click **Finish**.
3. Select the created resource environment reference and in WebSphere Bindings in the JDNI name field, enter `com/ibm/websphere/scheduler/calendar/DefaultUserCalendarHome` for the value.
4. Save the changes and close the file.

Adding code for the ACompanyCalendar

Open ACompanyCalendarBean.java and copy the code as in Example 13-4.

Example 13-4 ACompanyCalendarBean.java

```
package com.acompany.ejbs;

import java.util.Date;
import javax.naming.InitialContext;
import com.ibm.websphere.scheduler.UserCalendar;
import com.ibm.websphere.scheduler.UserCalendarHome;
import com.ibm.websphere.scheduler.UserCalendarPeriodInvalid;
import com.ibm.websphere.scheduler.UserCalendarSpecifierInvalid;

public class ACompanyCalendarBean implements javax.ejb.SessionBean {
   UserCalendar defaultCalendar;

   private javax.ejb.SessionContext mySessionCtx;

   public javax.ejb.SessionContext getSessionContext() {
      return mySessionCtx;
   }

   public void setSessionContext(javax.ejb.SessionContext ctx) {
      mySessionCtx = ctx;
   }

   public void ejbCreate() throws javax.ejb.CreateException {
      try {
         InitialContext initialContext = new InitialContext();
         UserCalendarHome defaultUserCalendarHome = (UserCalendarHome)
initialContext.lookup("java:comp/env/UserDefaultCalendar");
         defaultCalendar = defaultUserCalendarHome.create();
      } catch (Exception e) {
         System.out.println("ACompanyCalendarBean: create failed");
         e.printStackTrace();
      }
   }

   public void ejbActivate() {}

   public void ejbPassivate() {}

   public void ejbRemove() {}

   public Date applyDelta(java.util.Date baseTime, java.lang.String calendar, java.lang.String
delta) throws UserCalendarSpecifierInvalid, UserCalendarPeriodInvalid {
      try {
         if (delta.equalsIgnoreCase("begintrade")) {
```

```
            if (calendar.equalsIgnoreCase("Europe/Brussels"))
               return (defaultCalendar.applyDelta(baseTime, "CRON", "0 0 8 ? ? Mon-Fri"));
            else
               return (defaultCalendar.applyDelta(baseTime, "CRON", "0 0 9 ? ? Mon-Fri"));
         } else
            return (defaultCalendar.applyDelta(baseTime, calendar, delta));
      } catch (Exception e) {
         System.out.println("ACompanyCalendarBean: applyDelta failed");
         e.printStackTrace();
         throw new UserCalendarPeriodInvalid();
      }
   }

   public String[] getCalendarNames() {
      try {
         return (new String[] { "Europe/Brussels" + defaultCalendar.getCalendarNames()});
      } catch (Exception e) {
         System.out.println("ACompanyCalendarBean: getCalendarNames failed");
         e.printStackTrace();
      }
      return (new String[] { "Europe/Brussels" });
   }

   public void validate(java.lang.String calendar, java.lang.String delta) throws
UserCalendarSpecifierInvalid, UserCalendarPeriodInvalid {
      try {
         if (!delta.equalsIgnoreCase("begintrade"))
            defaultCalendar.validate(calendar, delta);
      } catch (Exception e) {
         System.out.println("ACompanyCalendarBean: validate failed");
         e.printStackTrace();
      }
   }
}
```

As you can see in the code for ACompanyCalendar's creation in the ejbCreate() method, it does a lookup for the default user calendar that will be used later.

Since a calendar bean generally defines more than one calendar, the implemented calendars are distinguished by their calendar name, which has to be specified when we specify which calendar bean we will use with our task. The calendar identifier of our user-defined calendar ACompanyCalendar is "`Europe/Brussels`". If any other calendar name is used, our calendar will pass the handling to the default calendar.

So, our calendar's name is added to the default calendar name list in getCalendarNames() and returned to the caller.

The only time delta interval that our calendar recognizes is `beginTrade`. If we use it with any other delta interval, our calendar will pass the handling to the default calendar.

Generally, in a calendar bean, delta time for the next task fire up is calculated using the applyDelta() method. If you use the right interval, begintrade, it will schedule the task using the default interval. The task will be scheduled to run every week, Monday to Friday, at 8 am or at 9 am. If you specify the calendar identifier, which is `Europe/Brussels`, the task will be scheduled at 8 am; otherwise it will be scheduled at 9am.

13.8 Configuration

In this section we explain the configuration procedures of the Scheduler service and its dependencies.

Work manager service in relation to the Scheduler service

The Scheduler service depends on the Asynchronous Beans service. It uses asynchronous alarm managers for firing task actions. This means that a work manager resource needs to be created and configured for use by the Scheduler.

Launch the Administrative Console and choose **Resources -> Work manager**. Select the scope where you want to create the work manager and click **New**. You will get the work manager configuration window, shown in Figure 13-5 on page 528. Specify the required properties, then click **OK**.

Important: The Scheduler service uses the work manager's alarm threads, so setting the number of alarm threads indirectly affects the Scheduler behavior. In fact it affects how many tasks can be scheduled concurrently. One alarm thread is used for each task action and another alarm thread for the daemon, so keep in mind how many tasks would be scheduled and set the number of alarm threads accordingly.

An important work manager property is Service names, which specify which context information is propagated to the task. For example, if one of the scheduled tasks also uses the Internationalization service, make sure that **Internationalization** is selected when configuring work manager for the Scheduler.

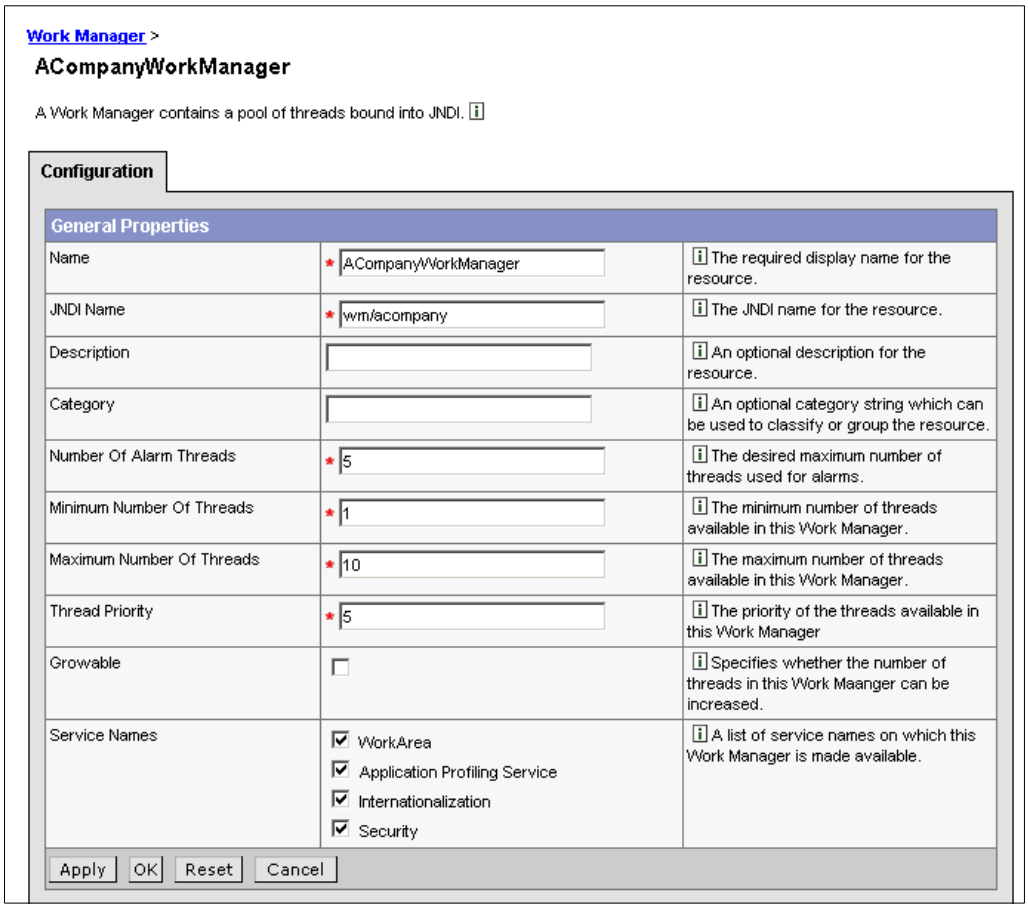


Figure 13-5 Work manager configuration

Save the configuration for WebSphere.

The Scheduler database

The Scheduler service stores task information in a database. The database needs to be created together with a JDBC provider and a data source resource.

There are batch scripts provided for creating the Scheduler database with different vendor database types. The scripts can be found in the <*WebSphere_root*>\Scheduler directory. You will have to modify the script to suit it to the database name, tablespace name, and the Scheduler table prefix name that you choose.

The table prefix is a property of the Scheduler configuration. It is a prefix for the task tables' names. Independent Schedulers can share one Scheduler database. You just have to use different table prefixes so a different set of tables are created and used by different schedulers within the same database.

Restriction: When using DB2, the table name length was restricted to 18 characters. As you may see in the Scheduler database creation scripts, nine characters are added to the table prefix for two of the table names. For example, if the table prefix property is “ACompSch”, one of the tables would be named “ACOMPSCHTASK_IDX1”. Keep this 18-character limit in mind when specifying the table prefix property.

To create the database in IBM DB2 UDB, do the following:

1. Copy the following files from the <*WebSphere_root*>\Scheduler to a temporary directory, for example: C:\temp\database.
   - createTablespaceDB2.ddl
   - createSchemaDB2.ddl
2. Open the createTablespaceDB2.ddl in a text editor.
   a. Replace all occurrences of `@SCHED_TABLESPACE@` with a tablespace name, for example `SCHEDTBLSPC`.
   b. Replace the `@location@` with a location in your system, for example `C:\ACompanyDB`. Make sure that the directory exists in your system.
3. Open the createSchemaDB2.ddl in a text editor.
   a. Replace all occurrences of `@TABLE_PREFIX@` with a table prefix, for example `SONE`.
   b. Replace all occurrences of `@SCHED_TABLESPACE@` with a tablespace name, for example `SCHEDTBLSPC`.
4. Start a DB2 command window and run the following commands:

```
cd c:\temp\database
db2 connect to REDBOOK user dbuser using passwOrd
db2 -tf createTablespaceDB2.ddl
db2 -tf createSchemaDB2.ddl
db2 disconnect current
```

5. Close the command window.
6. You can remove the C:\temp\database directory.

Scheduler configuration using the Administrative Console

The Scheduler service is configured using the Administrative Console. Launch the Administrative Console and choose **Resources -> Scheduler Configuration**. Select the scope where you want to create a Scheduler service and click **New**. You will see the Scheduler service configuration window shown in Figure 13-6. Specify the required properties. Make sure that you enter the database name where the Scheduler tables reside and specify the Scheduler table prefix (see “The Scheduler database” on page 528 for more details).

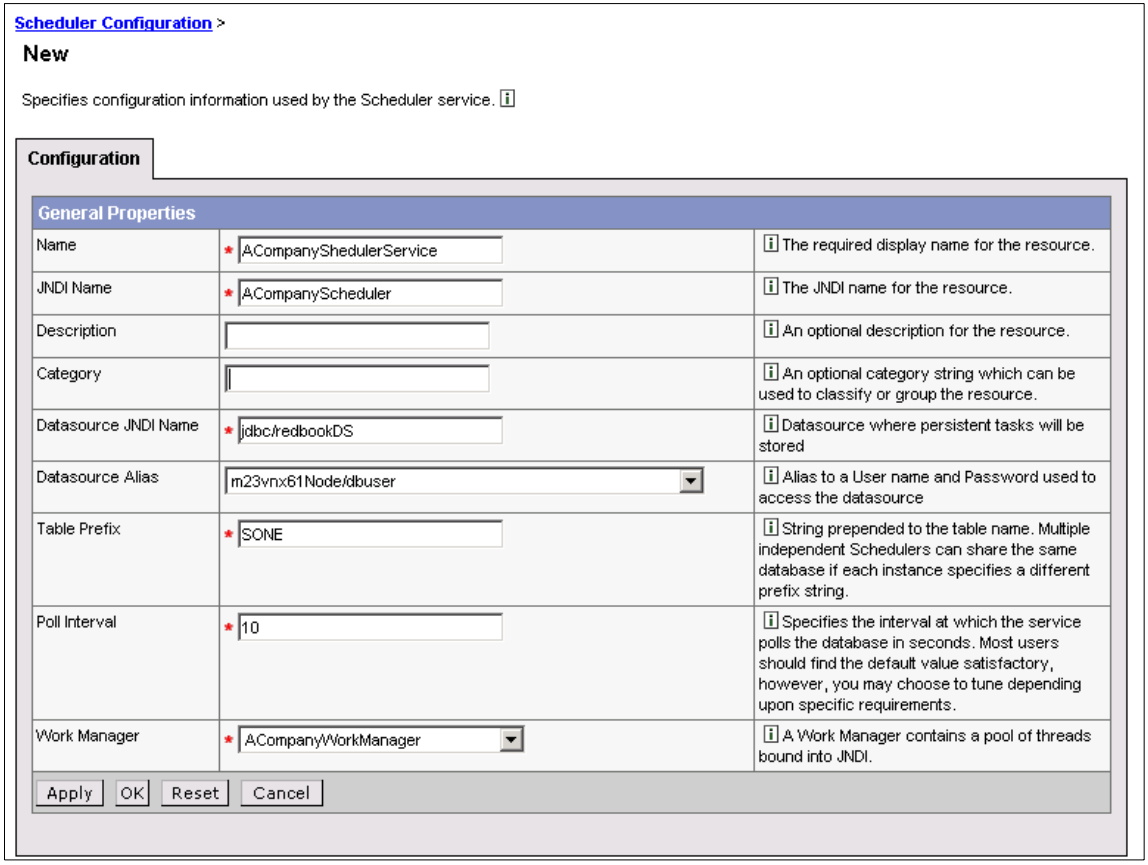


Figure 13-6 Scheduler service configuration

Save the configuration for WebSphere.

Disable the Scheduler service

The Scheduler service is enabled by default, which means it starts during the WebSphere startup.

In order to disable the service, perform the following steps:

1. Launch the Administrative Console, and choose **Servers -> Application Servers**. Select the application server that has defined the Scheduler service that you want to disable.
2. On the Configuration tab, click **Scheduler service**. Under General Properties, unmark the **Startup** property.
3. Save the configuration for WebSphere.

13.9 Deployment

Beside the configuration of the Scheduler service, you also have task definition EJBs, notification EJBs, and the code that uses the Scheduler with those beans. This is all included in your application as a part of the EAR. There are no specific steps related to the deployment of Scheduler beans in the WebSphere Enterprise runtime.

13.10 Scheduler service runtime

An important part of the Scheduler service is the wakeup daemon. It is an Asynchronous Bean created when the Scheduler service is started.

The wakeup daemon thread wakes up periodically and reads the tasks from the database that will start in the next scheduled interval according to the Scheduler service settings. It then creates alarms for each of the forthcoming tasks by using alarm threads from the assigned work manager pool.

Each alarm thread has the firetime set according to the specified task's firetime. When the alarm thread wakes up, it tries to run the action that is associated with its task. To prevent concurrent firing of the same action, there is a locking mechanism built in. The alarm thread is aborted if the locking mechanism finds out that some other daemon has already fired the task.

Locking mechanism: Every task has an integer value variable dedicated. When the alarm thread wakes up it starts a transaction and reads the task (together with the integer variable) from the database. Then it tries to update the variable, adding 1 to its value. If it succeeds, then the alarm thread fires the action of the task. After this, the alarm thread updates the task and commits the changes. If the alarm thread was not successful in updating the variable in the first place, this means that some other daemon has already fired the task, so the current transaction is rolled back and the alarm thread is aborted.

13.11 Problem determination and troubleshooting

If you encounter problems using the Scheduler service and you suspect that the source of the problem is somewhere within the service, then you can also use the following trace string to monitor what is happening in the execution time:

```
com.ibm.ws.scheduler.*=all=enabled
```

The example below shows a trace for our sample scenario during startup. You can see how the Scheduler service initializes and binds to the JNDI name space. Next, you can see how the Scheduler daemon starts up and starts reading the tasks.

Example 13-5 Sample trace

```
SchedulerServ > initialize
   com.ibm.websphere.mls.config.schedulerservice.impl.SchedulerServiceImpl (enable: true)
SchedulerServ I SCHD0001I: The Scheduler Service has started.
SchedulerServ < initialize
...
SchedulerServ > propertyChange
   java.beans.PropertyngeEvent[source=null [class
com.ibm.ws.runtime.component.ApplicationServerImpl]]
   STARTING
SchedulerServ < propertyChange
ResourceMgrIm I WSVR0049I: Binding DefaultWorkManager as wm/default
ResourceMgrIm I WSVR0049I: Binding ACompanyWorkManager as wm/acompany
SchedulerReso > isUsedFor
   com.ibm.websphere.mls.config.scheduler.impl.SchedulerConfigurationImpl (name:
ACompanySchedulerService, jndiName: ACompanyScheduler) (datasourceJNDIName: jdbc/redbookDS,
datasourceAlias: dbuser, tablePrefix: SOne, pollInterval: 10)
SchedulerReso < isUsedFor
   true
SchedulerReso > getBindingObject
SchedulerConf d Scheduler parameters for:     ACompanyScheduler
SchedulerConf d StartDaemon:                  true
```

```
SchedulerConf d Partition Range:              1-4
SchedulerConf d Partitions:                   1,2,3,4
SchedulerConf d Task Max Batch Size:          1
SchedulerConf d Task Max Batch Range:         0ms
SchedulerConf d Max Tasks per sec per Poll:   200
SchedulerConf d Max Tasks per Poll:           2000
SchedulerConf d Max Task Load Size:           250
SchedulerReso < getBindingObject
ResourceMgrIm I WSVR0049I: Binding ACompanySchedulerService as ACompanyScheduler
...
SchedulerServ > startSchedulers
SchedulerServ I SCHD0001I: The Scheduler Service has started.
SchedulerServ d Setting alarms which will poll the database
SchedulerServ d    Counted a total of 2 schedulers
SchedulerServ d Starting Scheduler Daemon for instance: ACompanyScheduler
SchedulerImpl > <init>
com.ibm.ws.scheduleonfig.SchedulerConfiguration@663df35d
SchedulerImpl > initialize
DBHelper      > <init>
TransactionCo > preinvoke
TransactionCo d preinvoke: No tran is present...Starting global tran.
TransactionCo < preinvoke
TXStart:true, LTCSt:false, TXSuspend:false, LTCSuspend:false
DBHelper      d Connection user=USERID
DBHelper      d Connection password=XXXXXX
DBHelper      d Retrieved connection for SchedulerConfiguration_1.  Total=1. ISOLEVEL=2
DBHelper      d DataSource Class: com.ibm.ws.rsadapter.jdbc.WSJdbcDataSource
DBHelper      d Database product: DB2/NT
DBHelper      d Database version: 08.01.0000
DBHelper      d Driver name:      IBM DB2 JDBC 2.0 Type 2
DBHelper      d Driver version:   08.01.0000
TransactionCo > postinvoke
TransactionCo d postinvoke: Global transaction was started. Attempting to commit.
TransactionCo d postinvoke: Global transaction committed..
TransactionCo d postinvoke: No local or global transaction to restore
TransactionCo < postinvoke
DBHelper      d Returned connection for SchedulerConfiguration_1.  Total=0
DBHelper      < <init>
TaskStoreImpl > <init>
TaskStoreImpl < <init>
SchedulerImpl < initialize
SchedulerImpl > createMBean
SchedulerImpl < createMBean
SchedulerImpl < <init>
SchedulerImpl > startDaemon
...
SchedulerImpl d Creating poller AsynchScope: ~SchedulerImpl_ACompanyScheduler_Poller
SchedulerImpl d Creating Tasks AsynchScope: ~SchedulerImpl_ACompanyScheduler_Tasks
SchedulerImpl d Started daemon for: ACompanyScheduler
```

```
SchedulerImpl < startDaemon
...
TransactionCo > postinvoke
TransactionCo d postinvoke: Global transaction was started. Attempting to commit.
SchedulerImpl > fired
SchedulerImpl d Destroying Tasks AsynchScope: ~SchedulerImpl_ACompanyScheduler_Tasks
SchedulerImpl d Creating Tasks AsynchScope: ~SchedulerImpl_ACompanyScheduler_Tasks
SchedulerImpl > poll
TransactionCo > preinvoke
TransactionCo d preinvoke: Local transaction is present... Suspending it and starting a Global
Tran
TransactionCo < preinvoke
TXStart:true, LTCSt:false, TXSuspend:false, LTCSuspend:true
DBHelper      d Connection user=USERID
DBHelper      d Connection password=XXXXXX
DBHelper      d Retrieved connection for SchedulerConfiguration_1.  Total=1. ISOLEVEL=2
TaskStoreImpl > findTasksBeforeNotComplete
   Mon Apr 14 16:21:5DT 2003 (SOneTASK,1050351716234,2000)
DBHelper      < getSQLStatement
DBHelper      d Looking for property TASK_DB2_FINDTASKSBEFORENOTCOMPLETE
DBHelper      d Looking for property TASK_DEFAULT_FINDTASKSBEFORENOTCOMPLETE
DBHelper      < getSQLStatement
   SELECT TASKID, VERS,ROW_VERSION, TASKTYPE, TASKSUSPENDED, CANCELLED, NEXTFIRETIME,
STARTBYINTERVAL, STARTBYTIME, VALIDFROMTIME, VALIDTOTIME, REPEATINTERVAL, MAXREPEATS,
REPEATSLEFT, TASKINFO, NAME, AUTOPURGE, FAILUREACTION, MAXATTEMPTS, QOS, PARTITIONID,
CREATETIME from SOneTASK where ((NEXTFIRETIME < ?) AND (REPEATSLEFT <> 0) AND (CANCELLED = 0)
AND PARTITIONID IN (1,2,3,4)) ORDER BY NEXTFIRETIME ASC
TransactionCo d postinvoke: Global transaction committed..
TransactionCo d postinvoke: No local or global transaction to restore
TransactionCo < postinvoke
```

13.12 Performance monitoring

By default the performance monitoring within the Scheduler service is not enabled. To enable it using the Administrative Console do the following:

1. Select the application server you want to monitor, switch to either the Configuration or the Runtime tab (if the process is running), and click **Performance Monitoring Service**. The difference is that changing the property on the runtime will already affect the currently running application server process, while changing the property on the configuration level will take effect with the application server's next startup.
2. If you do not want to monitor all the WebSphere components, you should not set the initial specification level to Standard. It is better to change the PMI enablement just for the Scheduler service. Choose **Custom initial**

specification, find the schedulerModule property, which by default is set to N (none).

3. Change the schedulerModule property value to H (high). See Figure 13-7 on page 536. Apply the changes and save the configuration if necessary.
4. At this point, you can start monitoring the Scheduler service using the performance viewer client.

Tip: All the Scheduler service performance monitoring variables belong to the High group of PMI specification levels.

The following Scheduler service runtime behavior can be monitored using the WebSphere Performance Monitoring Infrastructure (PMI):

- Failed tasks

  Total number of tasks where execution failed for application reasons.

- Executed tasks

  Total number of tasks executed successfully.

- Number of polls

  The number of polls (of the task database) since the Scheduler service started up.

- Tasks per sec

  The number of tasks per second processed by the Scheduler service.

- Collisions per sec

  The number of collisions (trying to place the same task into alarm) between multiple Scheduler services.

- Time for poll query (ms)

  The time taken to get all tasks due to expire (that will be scheduled for the next alarm) from the database.

- Task execution Time (ms)

  The time taken to execute a task in milliseconds.

- Tasks expiring per poll

  The number of tasks in a poll.

- Task latency (secs)

  The average latency of tasks in seconds, which means how late tasks were executed.

- Poll time (secs)

  The time between pools.

- Tasks executed per poll

  The number of tasks executed per poll.

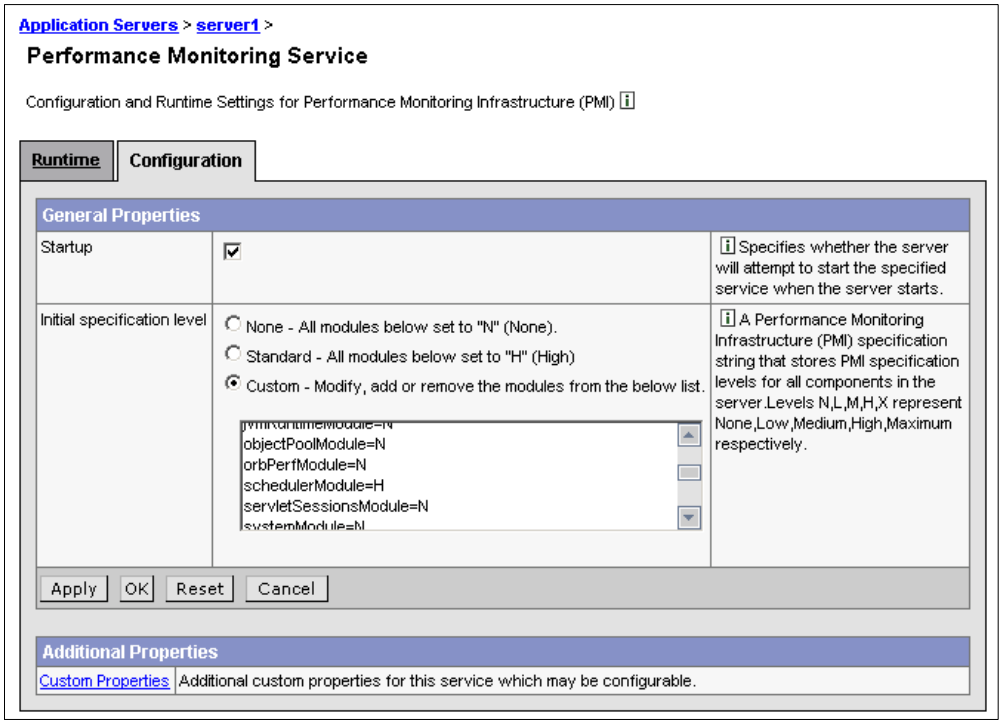


Figure 13-7 PMI configuration for the Scheduler service

13.13 Security considerations

If you use security on WebSphere V5.0 Enterprise, the default identity that the Scheduler service uses with the task action session bean is *system*. This means if you do not specify any deployment-related security settings on the bean, the identity of the bean will be the same as the one used for starting WebSphere.

If the task action session bean (TaskHandler) uses some secured objects that require a different caller identity, then the security identity must be enabled on the Startup Bean and The Run-as on the bean's process must be configured to use the correct identity.

14

Object pools

This chapter discusses the object pools service in WebSphere Application Server Enterprise V5.

The sample application is a J2EE client that uses the application server's object pool service. This sample is not part of the sample scenario used for most of the book.

14.1 Planning

Java by its design is a “memory safe” language. The Java Virtual Machine (JVM) provides a mechanism that takes memory handling tasks away from the programmer. Java programs create a huge amount of Java objects during execution, and the garbage collecting mechanism takes care of deallocating the memory of the objects that are no longer needed. However garbage collecting is still an expensive operation, since it can take a lot of processing resources. Beside garbage collecting, we also have time penalties for object instantiation. Every object includes some overhead information, which is also initialized. Naturally each new release of the JVM implementation tends to improve on these issues, but still there must be something done in the meantime. One step to improve performance is object pooling. A Java application can have its own pool of objects that are already instantiated and waiting for use. When the application at any point needs an object, it just “fetches” one from the pool and later when it is no longer needed, it returns the object to the pool for later reuse.

Why use object pools?

Using WebSphere’s object pooling has the following advantages:

- Administration

  WebSphere’s object pool service administration is consistent with WebSphere resource management. Object pools are managed through WebSphere's administration mechanisms, which include all the advantages of centralized management with the Network Deployment version.

- Implementation level

  With WebSphere Enterprise V5, object pooling is already implemented within the J2EE application container. It is an extension. In contrast with your own object pooling implementation, where it would be just a part of your J2EE application, the object pooling service in WebSphere Enterprise V5 is coupled closely with the rest of the services in J2EE application container.

- Performance monitoring and tracing

  Since object pooling is a part of WebSphere Enterprise V5, it also uses the PMI and the tracing infrastructure. You can monitor what is happening with the object pool in the runtime.

- Variety

  WebSphere’s object pool implementation provides two object pool types: synchronized and unsynchronized.

- Flexibility

  If the provided object pool implementation does not suit your application, you can create your own object pool implementation, which is then managed by WebSphere's object pool manager.

14.2 Design

Generally, you should consider using object pools if your application consists of many threads that frequently use a common complex object.
The performance benefits of using object pools should be carefully considered. The majority of applications do not need to use object pools to obtain adequate performance, but those who need it can gain significant performance improvements. When considering whether to use object pools or not, you should think about the following two issues:

- Size and complexity of the object

  Using object pools with simple object types, you could slow down your application performance unless the usage frequency is so high that you can achieve improvement over normal JVM garbage collection.

- Object usage frequency

  Logically, more frequent use of some object qualifies them better to be used with object pooling.

You can make estimations or run tests with simple code about how much time is expected to be used in JVM for instantiation of the given object type and later for garbage collecting, depending on the JVM memory settings.

Do not forget to select the right pool type, synchronized or unsynchronized. A synchronized pool is useful for applications where little contention exists for objects in the pool. A synchronized pool can be shared across threads, which means more efficient use of pooled objects, because there are fewer idle instances of objects in the pool at any given time. In addition, a single shared pool is easier to manage than multiple pools.

14.3 Development

In the following sections, we describe how to use object pools from your application.

There is no special support in WebSphere Studio IE or any other tool for object pools. You only have to make sure that the objectpool.jar exists in the project's Java classpath.

Important: In order to successfully compile your application and generate deploy code, the objectpool.jar file must be added to Java build path. In WebSphere Studio IE the file is in the <*WebSphere_Studio_IE_root*>\WAS_EE_V5\lib directory. In WebSphere V5.0 Enterprise, the file is in the <*WebSphere_root*>\lib directory.

14.3.1 Object Pools API

The Object Pools API is contained in the com.ibm.websphere.asynchbeans.pool package. For more details, check the complete description of the Object Pools API in the WebSphere Enterprise InfoCenter:

http://publib7b.boulder.ibm.com/wasinfo1/en/info/ee/javadoc/ee/com/ibm/websphere/asynchbeans/pool/package-summary.html

The following sections briefly introduce the main parts of WebSphere's Enterprise object pool implementation.

Object pool manager

The object pool manager interface describes the object pool manager. The object pool manager is a factory for object pools. WebSphere Enterprise V5 provides implementation for the object pool manager. You can define one or more managers and register them in the WebSphere JNDI name space using Administrative Console. Refer to "Object pool manager configuration" on page 553 for details.

The object pool manager interface defines two methods:

- createFastPool(): This method returns an unsynchronized object pool.
- getPool(): This method returns a synchronized object pool.

Object pool

The object pool is a pool of reusable objects of the same type. WebSphere's default object pool implementation implements this interface. Using the default implementation, any Java object that has a public default constructor can be pooled.

This interface define two methods:

- getObject(): Retrieves an object from the object pool.
- returnObject(): Returns an object that is no longer needed to the object pool.

Make sure that you return the objects to the right pool. If not, you will get a ClassCast exception.

Custom object pool

Custom object pools are used when WebSphere's object pool implementation does not suit the needs of your application, for example if you need a notification about taking and returning an object to and from the object pool. Since the default WebSphere object pool implementation does not provide this, you can use this interface to create your object pool implementation. The custom ObjectPool interface extends the ObjectPool interface with additional methods that you need to implement beside the methods inherited from the ObjectPool interface. The methods defined in the custom ObjectPool interface are:

- flushPool(): Called when the system needs memory, so it asks you to clean all the idle objects.
- setPr*operties()*: Called when the custom object pool is constructed. A map of the supplied parameters is supplied. These are the additional optional parameters added to the custom object pool configuration. Refer also to "Custom object pool configuration" on page 553.

Each custom pool must be registered within an object pool manager.

Important: Do not synchronize the methods of your custom object pool implementation. Your implementation is wrapped by an internal object pool implementation that adds synchronization when it is created using the ObjectPoolManager.getPool() method. When a fast pool is created, then your implementation runs unsynchronized.

PoolableObject

When pooling the object of your own defined Java class types using the default object pool implementation, you might need to have some initialization or cleanup code in the object when the object is taken from the pool or returned to the pool. Such an object needs to implement the PoolableObject interface with the following methods:

- init() is called when the object is about to be reused from the pool.
- returned() is called when the object is returned to the pool.

The default pool implementation checks if the object being pooled implements this interface. If it does, then it will call the init() method when getting the object from the pool and it will call the returned() method when the returning the object.

14.3.2 Steps for using object pools

The following steps must be done in your application when programming for object pools.

Find the object pool manager

The object pool manager is exposed in the JNDI name space. The manager has to be created in WebSphere Enterprise V5 using its Administrative Console. Refer to “Object pool manager configuration” on page 553 for more details about the object pool manager configuration.

Programmatically, you do a lookup on a context to find an object manager. See Example 14-1.

Example 14-1 Object pool manager lookup

```
initialContext = new InitialContext();
opm = (ObjectPoolManager) initalContext.lookup("java:comp/env/ObjectPool");
```

You can see that we do a lookup using a local reference, so make sure that this local reference is bound to the object pool manager JNDI, as described in “Add a resource environment reference” on page 546.

Get an object pool for specified class

When you have the object pool manager, you will ask it to get an object pool for the class you want to be pooled. This is done by calling the getPool() method on the object pool manager, which will then create an object pool for the given class and return it to your control. See Example 14-2. It gets an object pool for the ArrayList class.

Example 14-2 Getting an object pool from the object pool manager

```
ObjectPool oPool = opm.getPool(ArrayList);
```

Important: Only one object pool for the specified class exists in a JVM. If the object pool for the given class already exists, the object pool manager returns that object pool. It does not create another object pool for the same object type.

The createFastPool() method can also be called. In this case, the object pool manager creates and returns an object pool that is not a thread safe, unsynchronized object pool.

Get an object from the object pool

After we have the object pool created, we can get an object of the specified class by calling the getObject() method of the object pool. See Example 14-3.

Example 14-3 Getting an object from the object pool

```
ArrayList obj = oPool.getObject();
```

Return the object to the object pool

After we do not need the object in our application anymore, we return the object back to the object pool by calling returnObject() method of the object pool. See Example 14-4.

Example 14-4 Returning the object to the object pool

```
oPool.returnObject(obj);
```

If you “forget” to return the object to the object pool, its instance will remain there, since it is not used in future requests. This is a programmatic error and it will not do any harm, except consume more resources, but it will definitely hinder object pooling capabilities, so it will not be efficient at all. Make sure that you return the objects.

14.4 Unit test environment

To use object pools, there is nothing special to configure in WebSphere Studio IE test environment. The object pool service for the application server is enabled by default. After developing and publishing the project, the test server is ready to run the application with object pool support.

14.5 Assembly

This section describes the assembly time actions related to object pools.

Object pool manager resource environment reference

In your code, you should use local name references for the object pool manager. Before you can use an object pool, you need to specify the location in the JNDI name space. You can use the Application Assembly Tool to specify such references for an enterprise application:

1. Open the application EAR file in the Application Assembly Tool. Select the module from where object pools are used. If there are more than one, you will

have to create a resource environment reference for each one. That means each module—client, EJB or Web module—from which you use object pools needs a resource environment reference for the object pool manager.

2. If you selected an EJB module, you have to find the bean that uses object pool manager and then expand it. If you selected the client application module, the resource environment property is already in the subroot directory. Right-click **Resource Environment References** and select **New**.
3. On the General tab, enter the name you are using in your code as a local reference for the object pool manager. For the Type, enter `com.ibm.websphere.asynchbeans.pool.ObjectPoolManager`.
4. Under the Bindings tab, enter the JNDI name that you specified when creating the object pool manager on WebSphere Application Server.

14.6 Sample application

The following sample application demonstrates the use of object pools. The application does not relate to the sample scenario used for this book. It shows the difference between using some object with or without object pooling. It creates a specified number of job threads and starts them.

As command-line arguments, the application takes two parameters:

- Number of threads: The number of job threads it will create.
- Java class name: Specifies which Java class is being used for simulating jobs. With this you can see the difference in performance when using different classes with object pooling or not.

Each job thread runs two different units of work several times and calculates the execution time for both cases: the first unit of work is without object pools while the second unit uses object pooling. A unit of work in JobThread is represented by doSomething() and doSomethingUseObjectPool() methods. These two methods allocate an object of the specified class type, then they put the thread to a random wait between 1 and 6 milliseconds. This represents doing some work with the objects.

Prerequisites

The following prerequisites have to be met before starting with the sample application:

- For testing the sample with WebSphere Studio IE, there must be an instance of a WebSphere V5.0 Enterprise test server created.

- An object pool manager has to be created for the WebSphere Studio IE test application server. Start your test application server and launch the Administrative Console. Additionally, create an object pool manager using the steps in “Object pool manager configuration” on page 553. Remember the JNDI name you use for the object pool manager.

Create application client project

First, you have to create an application client project. Do the following:

1. In WebSphere Studio IE, select **File** -> **New** -> **Application Client Project**. Then select **Create J2EE 1.3 Application client project**. Enter `ObjectPtest` as the project name and `ObjectPtestEAR` as the new project name. Click **Finish**. A new application client project is created.
2. In the J2EE Navigator view, select the newly created **ObjectPtest** project and right-click it. Select **Properties** and you will get the Properties for ObjectPtest window. Go to the Java Build Path and select the **Libraries** tab.
3. In the Libraries tab, choose **AddVariable**, select **WAS_EE_V5**, double-click it, go to lib directory and choose **objectpool.jar**. Apply the changes.

Create ObjectTest application client

To add the main class into application client project, follow these steps:

1. Still using the J2EE navigator view, in the ObjectPtest project, select **appClientModule**, right-click it and select **New** -> **Class**. In the New Java Class window, enter `ObjectTest` as the value for the Name property. Leave all the other properties with their default values. Click **Finish**.
2. Copy the code as shown in Example 14-5 into the ObjectTest.java class.

In the code, you can see that the client first tries to find the object pool manager, then creates a specified number of job threads and starts them.

Example 14-5 ObjectTest.java

```
import javax.naming.InitialContext;
import com.ibm.websphere.asynchbeans.pool.ObjectPoolManager;

public class ObjectTest {
   public static void main(String[] args) {
      if (args.length < 2) System.out.println("usage:ObjectTest numberOfThreads className");
      int numberOfThreads = Integer.parseInt(args[0]);
      JobThread[] threads = new JobThread[numberOfThreads];
      ObjectPoolManager opm = null;
      String className = args[1];
      InitialContext initalContext;

      //Lookup for object pool manager
```

```
      try {
         System.out.println("ObjectTest: looking for ObjectPoolManager");
         initalContext = new InitialContext();
         opm = (ObjectPoolManager) initalContext.lookup("java:comp/env/ObjectPool");
      } catch (Exception ex) {
         ex.printStackTrace();
      }
      for (int i = 0; i < numberOfThreads; i++) {
         //Create a new JobThread thread
         threads[i] = new JobThread("JobThread_" + i, opm, className);
         System.out.println("ObjectTest: starting thread: " + threads[i].getName());
         //Start the thread thread
         threads[i].start();
      }
   }
}
```

Add a resource environment reference

Since a local reference to object pool is used in ObjectTest.java, you have to add a resource environment reference to the application, as follows:

1. Open the Client Deployment Descriptor of the ObjectPtest project. Switch to the **References** tab and click **Add**.
2. Select **Resource environment reference** and click **Next**. Enter `ObjectPool` in the Name field, and enter `com.ibm.websphere.asynchbeans.pool.ObjectPoolManager` in the Type field. Click **Finish**.
3. Select the created resource environment reference and under WebSphere Bindings and in the JDNI name field, enter the value you specified when you created the object pool manager on the test application server (see “Object pool manager configuration” on page 553 for the value). In our case, we entered `ACObjectPoolManager` for the JNDI name.
4. Save and close the descriptor.

Change the main class of the application client

The next step is to change the specified main class of the client, so that the J2EE client container will know which class to run when starting the application. Our main class is ObjectTest.java. The client container will start the program by starting its main() method. Follow these steps:

1. Using the J2EE navigator view, in the ObjectPtest project, select **appClientModule -> META-INF** and open the MANIFEST.MF file.
2. Under the Dependencies tab at the bottom, enter `ObjectTest` as the value for the Main-Class property.

3. Save and close the file.

Create JobThread

The last step is to add a JobThread class. Do the following:

1. On the J2EE navigator view, in the ObjectPtest project, right-click the **appClientModule** and select **New** -> **Class**.
2. In New Java Class window, enter `JobThread` as the value for the Name property. Clear the Public Static Void Main(String args[]) field under the Which method stubs would you like to create? property. Leave all the other properties with their default values. Click **Finish**.
3. Copy the code as in Example 14-5 on page 545 into the JobThread.java class.

Example 14-6 JobThread.java

```
import java.util.Random;
import com.ibm.websphere.asynchbeans.pool.ObjectPool;
import com.ibm.websphere.asynchbeans.pool.ObjectPoolManager;

public class JobThread extends Thread {
   static Double averageExecutionTime = new Double(0);
   static Double averageExecutionTimeUsingPool = new Double(0);
   static Long totalNumberOfRepetitions = new Long(0);
   Random rand = new Random(System.currentTimeMillis());
   ObjectPool oPool = null;
   ObjectPoolManager opm;
   String className = new String();
   Class customClass;
   //Initialize
   public JobThread(String name, ObjectPoolManager opm, String className) {
      super.setName(name);
      this.opm = opm;
      this.className = className;
   }

   public void run() {
      // Wait between 1-100 miliseconds before start
      synchronized (this) {
         try {
            this.wait(1 + rand.nextInt(100));
         } catch (InterruptedException e) {};
      }
      try {
         customClass = Class.forName(className);
         //Get an object pool for specified class
         oPool = opm.getPool(customClass);
```

```
      } catch (Exception ex) {
         ex.printStackTrace();
      }
      int repetitions = rand.nextInt(500);
      synchronized (totalNumberOfRepetitions) {
         totalNumberOfRepetitions = new Long(totalNumberOfRepetitions.longValue() +
repetitions);
      }
      //Do something using objects without object pooling
      //Repeat the job 1-500 times
      //Read the time before starting the job
      long startTime = System.currentTimeMillis();
      for (int i = 0; i < repetitions; i++) {
         doSomething();
      }
      //Read the time after ending the job
      long endTime = System.currentTimeMillis();
      //Do something using objects WITH object pooling
      //Repeat the job 1-500 times
      long startTimeUsingObjectPool = System.currentTimeMillis();
      for (int i = 0; i < repetitions; i++) {
         doSomethingUseObjectPool();
      }
      long endTimeUsingObjectPool = System.currentTimeMillis();
      //Calculate doSomething execution time and add it to the total value
      synchronized (averageExecutionTime) {
         averageExecutionTime = new Double(averageExecutionTime.doubleValue() + endTime -
startTime);
      }
      //Calculate doSomethingUseObjectPool execution time and add it to the total value
      synchronized (averageExecutionTimeUsingPool) {
         averageExecutionTimeUsingPool = new
Double(averageExecutionTimeUsingPool.doubleValue() + endTimeUsingObjectPool -
startTimeUsingObjectPool);
      }
      System.out.println("Exiting thread" + this.getName());
      System.out.println("doSomething average execution time = " + (endTime - startTime) /
repetitions);
      System.out.println("doSomethingUseObjectPool average execution time = " +
(endTimeUsingObjectPool - startTimeUsingObjectPool) / repetitions);
      System.out.println("Total average execution time = " +
averageExecutionTime.doubleValue() / totalNumberOfRepetitions.doubleValue());
      System.out.println("Total average execution time using pool= " +
averageExecutionTimeUsingPool.doubleValue() / totalNumberOfRepetitions.doubleValue());
   }

   public void doSomething() {
      try {
         Object aL = customClass.newInstance(); //Create a new object
```

```
            randomWait(1 + rand.nextInt(5)); //Wait between 1 and 6 miliseconds
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public void doSomethingUseObjectPool() {
        Object obj = oPool.getObject(); //Get an object from the object pool
        randomWait(1 + rand.nextInt(5));//Wait between 1 and 6 miliseconds
        oPool.returnObject(obj);//Return the object to the object pool
    }

    synchronized private void randomWait(int miliseconds) {
        try {
            this.wait(miliseconds);
        } catch (InterruptedException e) {}
    }
}
```

When a JobThread thread is started, it enters the run() method. It waits for 1-100 milliseconds, then it creates an object pool for the given class.

The next step is to run a unit of work without using object pooling. This is done by running the doSomething() method and is repeated for 1-500 times (a random value). After and before, the system time is read to calculate how long it took to run all the repetitions.

The next step is to run a unit of work, but this time using object pooling. This is done by running the doSomethingUseObjectPool() method and is repeated for the same number of repetitions as previously for the doSomething() method. After and before, the system time is read to calculate how long it took to run all the repetitions.

Keep in mind that every unit of work includes a random wait between 1 and 5 milliseconds after the application allocates the object and before it returns it to the object pool. In the long run, this should not have any effect on the time difference between the doSomething() and doSomethingUseObjectPool() methods.

After all the jobs are done in a thread, the code calculates and reports the time values for both job methods. Since the number of repetitions between different threads differs, generally it returns different values for different threads.

In the end, it calculates the total average time for both doSomething() and doSomethingUseObjectPool() methods.

Note: The total average time is an average of the time that it took to execute the method, including all the repetitions in all the threads. A time share from each thread is weighted by the number of repetitions for the thread, so we cannot have the accurate value until the last thread finishes. When the last thread finishes, it reports the correct value.

Test the application client

Export the project into EAR file and use the **launchClient** command to run the project in a J2EE client container. The detailed step-by-step instructions are as follows:

1. Select the **ObjectPtest** project, and select **Project -> Rebuild Project**.
2. Select **Export**, then select **EAR file**, and click **Next**.
3. Select the **ObjectPtestEAR** for export. Enter the file name to which you want to export the project, for example `C:\SG246932\ObjectPtest.ear`. Click **Finish**, and check if the project was exported.
4. Open a console or command prompt window. Change to the directory where the WebSphere launchclient.bat resides.
5. Make sure that the test server is running with the object pool sample application.
6. The following command is an example of how to use the client:

   ```
   launchClient C:\SG246932\ObjectPtest.ear -CCjar=ObjectPtest.jar
   -CCBootstrapPort=2809 -CCverbose=true 500 java.util.ArrayList
   ```

   The first argument is the number of threads, for example `500`. The second argument is the Java class for tests. Just make sure that they are in the classpath.

   In Example 14-7, you can see the sample output after running the command.

Example 14-7 Sample output for ObjectTest object pooling test application.

```
C:\Program Files\WebSphere\AppServer\bin>launchClient C:\ObjectPtest.ear -CCjar=ObjectPtest.jar
-CCBootstrapPort=2809 -CCverbose=true 500 java.util.ArrayList
IBM WebSphere Application Server, Release 5.0
J2EE Application Client Tool
Copyright IBM Corp., 1997-2002
WSCL0012I: Processing command line arguments.
WSCL0001I: Command line, property file, and system property arguments resolved to:
        File to launch          = C:\ObjectPtest.ear
        CC Property File        = null
        Client Jar File         = ObjectPtest.jar
        Alternate DD            = null
        BootstrapHost           =
```

```
        BootstrapPort           = 2809
        Trace enabled           = false
        Tracefile               = null
        Init only               = false
        Classpath Parameter     = null
        Security Manager        = disable
        Security Manager Class  = Not used. -CCsecurityManager=disable
        Security Manager Policy = Not used. -CCsecurityManager=disable
        Exit VM                 = false
        Soap Connector Port     = null
        Application Parameters  = 500 java.util.ArrayList
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0400I: Binding resource environment reference object:
           JNDI name: ObjectPool ==> ACObjectPoolManager @ corbaloc:iiop:localhost:2809
           Type: com.ibm.websphere.asynchbeans.pool.ObjectPoolManager
           Description:
WSCL0031I: The object was bound successfully.
WSCL0600I: Binding HandleDelegate object.
WSCL0031I: The object was bound successfully.
WSCL0900I: Initializing and starting components.
WSCL0910I: Initializing component: com.ibm.ws.activity.ActivityServiceComponentImpl
WSCL0911I: Component initialized successfully.
WSCL0910I: Initializing component:
com.ibm.ws.ActivitySession.ActivitySessionClientComponentImpl
WSCL0911I: Component initialized successfully.
WSCL0910I: Initializing component: com.ibm.ws.workarea.WorkAreaServiceClient
WSCL0911I: Component initialized successfully.
WSCL0910I: Initializing component: com.ibm.ws.appprofile.EEClientAppProfileComponentImpl
WSCL0911I: Component initialized successfully.
WSCL0910I: Initializing component: com.ibm.ws.i18n.context.I18nClientComponentImpl
WSCL0911I: Component initialized successfully.
WSCL0901I: Component initialization completed successfully.
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
WSCL0014I: Invoking the Application Client class ObjectTest
ObjectTest: looking for ObjectPoolManager
ObjectTest: starting thread: JobThread_0
ObjectTest: starting thread: JobThread_1
ObjectTest: starting thread: JobThread_2
ObjectTest: starting thread: JobThread_3
ObjectTest: starting thread: JobThread_4
...
Exiting threadJobThread_485
doSomething average execution time               = 37
doSomethingUseObjectPool average execution time = 30
Total average execution time           = 50.34385009889552
Total average execution time using pool= 31.956074635486864
Exiting threadJobThread_484
doSomething average execution time               = 37
doSomethingUseObjectPool average execution time = 30
```

```
Total average execution time           = 50.50452468887294
Total average execution time using pool= 32.088585881570424
Exiting threadJobThread_483
doSomething average execution time               = 37
doSomethingUseObjectPool average execution time = 30
Total average execution time           = 50.66519927885036
Total average execution time using pool= 32.22109712765399
```

Results

We received the following test result on our system:

```
Total average execution time           = 50.66519927885036
Total average execution time using pool= 32.22109712765399
```

The results can be interpreted as follows. On the tested system when using 500 threads and the java.util.ArrayList class, it took approximately 50 milliseconds on average per job task when no object pools were used. When object pooling was used, it took approximately 32 milliseconds on average per job task.

You can use the application to test the difference using your own classes. You can also check how the JVM memory settings affect behavior.

Important: Although this sample application shows the performance gain for object pools, it is just a proof of concept and it has flaws. You cannot expect the same performance gains with a “real” application, because there are many factors that have not been and cannot be calculated here.

14.7 Configuration

The object pool service is configured using the WebSphere Administrative Console. First you have to configure an object pool manager and then you can create object pools under the object pool manager.

Disable the object pool service

The object pool service is enabled by default. It starts during the WebSphere startup. If you want to disable the service, take the following steps:

1. Launch the Administrative Console and log in.
2. Select **Servers -> Application Servers**. Select the application server that has defined the object pool service that you want to disable.
3. Under the Configuration tab, click **Object Pool Service**. Under the General Properties, unmark the **Startup** property.
4. Apply the changes and save the configuration for WebSphere.

Object pool manager configuration

Perform the following steps to configure the object pool manager for WebSphere Enterprise V5:

1. Launch the Administrative Console.
2. Select **Resources -> Object pools**. On the first administration window, select the scope when you want to create a scheduler service and click **New**.
3. You will get the object pool manager configuration window, as shown in Figure 14-1. Specify the mandatory properties for the object pool manager name and the object pool manager JNDI name. Optionally, you can specify Description and Category properties.

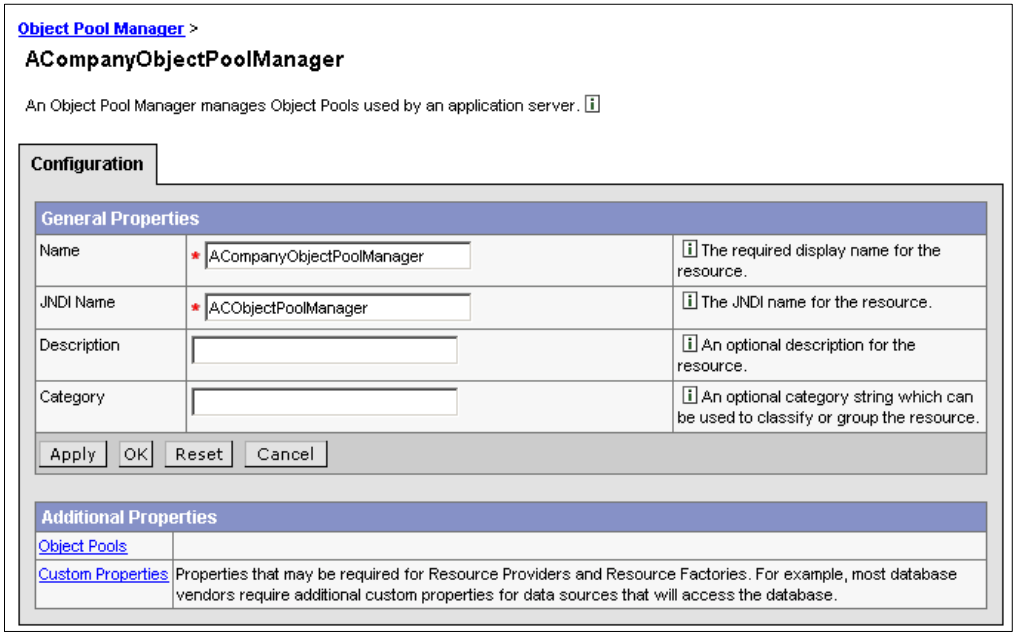


Figure 14-1 Object pool manager configuration

4. Save the configuration for WebSphere.

Custom object pool configuration

When WebSphere's object pool implementation does not suit your application's needs, you can provide a customized object pool implementation, which implements the CustomObjectPool interface. Every custom object pool has to be registered under an existing object pool manager. In order to define a custom object pool on a given object pool manager, click **Object Pools** in the Additional Properties section of the object pool manager configuration (see Figure 14-1). Then for the Pool Class Name, specify the type of class that is pooled. And in the

PoolImpl Class Name field, specify the class name of your object pool implementation. See Figure 14-2.

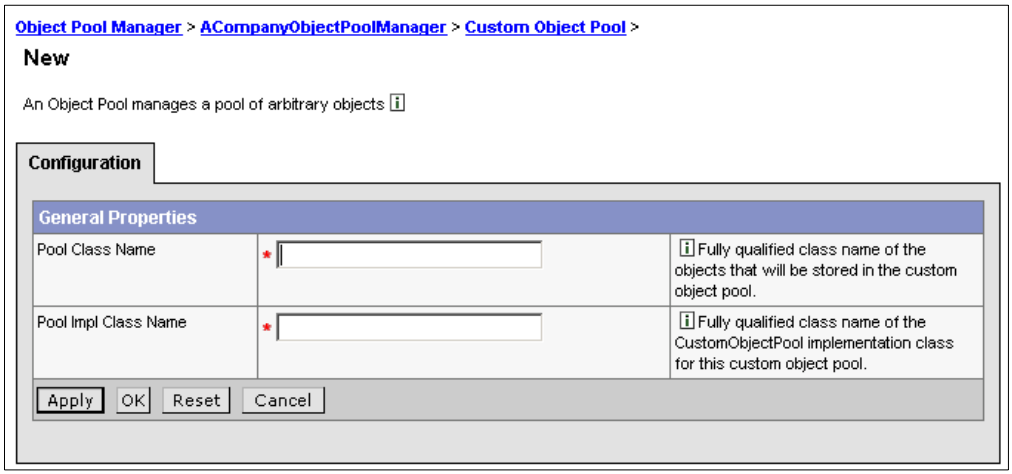


Figure 14-2 Configuring custom object pool

Additionally you can specify custom properties that will be given to your pool implementation in runtime, when the customer object pool is created. See “Custom object pool” on page 541 for more details. Click **Custom Properties** in the Additional Properties section of the object pool manager configuration (see Figure 14-1 on page 553).

14.8 Runtime environment

As you may already determined from the Java API name, object pools are related to Asynchronous Beans. Async Beans internally use object pooling, but not only the Async Bean service. The Internationalization service also uses object pools internally.

Synchronized object pools

WebSphere offers two types of object pools: synchronized and unsynchronized. The synchronized type of pool is thread safe. Applications can share the pool among different threads. Underneath, the synchronized pool is in fact a wrapped unsynchronized pool.

Unsynchronized object pools

Unsynchronized pools are not thread safe. Applications can only use them within one thread. They are faster than the synchronized, so if your application uses an object pool in one thread, we recommend the use of unsynchronized pool.

Sharing object pools

If the application server shares an object pool between applications, care must to be taken to avoid class loader problems. This is not an issue when pooling JDK supplied objects, such as java.util.HasMap, for example. But if you are pooling application objects, then class loader problems may occur. We recommend not sharing pools between applications when application objects are being pooled.

Workload management and failover

Object pool service does not have any special workload management or failover capabilities. It is running within a JVM scope. When you configure the object pool configuration manager on the cell or on the node level, every application server that belongs to the cell or node will have one with the specified name registered, and they are not aware of each other. If you work with object pools on one server and you lose it, your application can still continue to run on another server. However, since it exists only in memory, object pools on the crashing server will be lost.

Programmatically, you can still return the objects obtained from the crashed server pool to the new pool, but if the type of the pooled class is not the same, you will get a ClassCastException exception.

14.9 Problem determination and troubleshooting

If you encounter problems using object pool service and you suspect that the problem source is somewhere within the service then you can additionally use the following trace string to monitor what is happening in the execution time:

```
com.ibm.ws.asynchbeans.pool.*=all=enabled
```

14.10 Performance monitoring

By default, performance monitoring within the object pool service is not enabled. Follow the steps below to enable it:

1. Launch the Administrative Console and log in.
2. Select the application server of which object pool you want to monitor, switch to the Configuration or the Runtime tab (only if the process is running) and

click **Performance Monitoring Service**. The difference is that changing the property on the runtime will already affect the currently running application server process, while changing the property on the configuration level will take effect with the application server's next startup.

3. If you do not want to monitor all the WebSphere components, you should not set the initial specification level to Standard, but only change PMI enablement for the object pool service. Choose **Custom initial specification**, and locate the objectPoolModule property, which by default is set to N (none).
4. Change the objectPoolModule property value to H (high). See Figure 14-3 on page 557.
5. Apply the changes and save the configuration.
6. After the configuration, you can monitor the object pool service through your performance viewer client.

Tip: All the object pool service performance monitoring variables belong to the High group of the PMI specification levels.

The following object pool service runtime behavior can be monitored using WebSphere's Performance Monitoring Infrastructure (PMI):

- Objects created: Total number of new objects created.
- Objects allocated: The number of objects requested from the pool.
- Objects returned to pool: The number of objects returned to the pool.
- Idles object in pool: Average number of idle object instances in the pool.

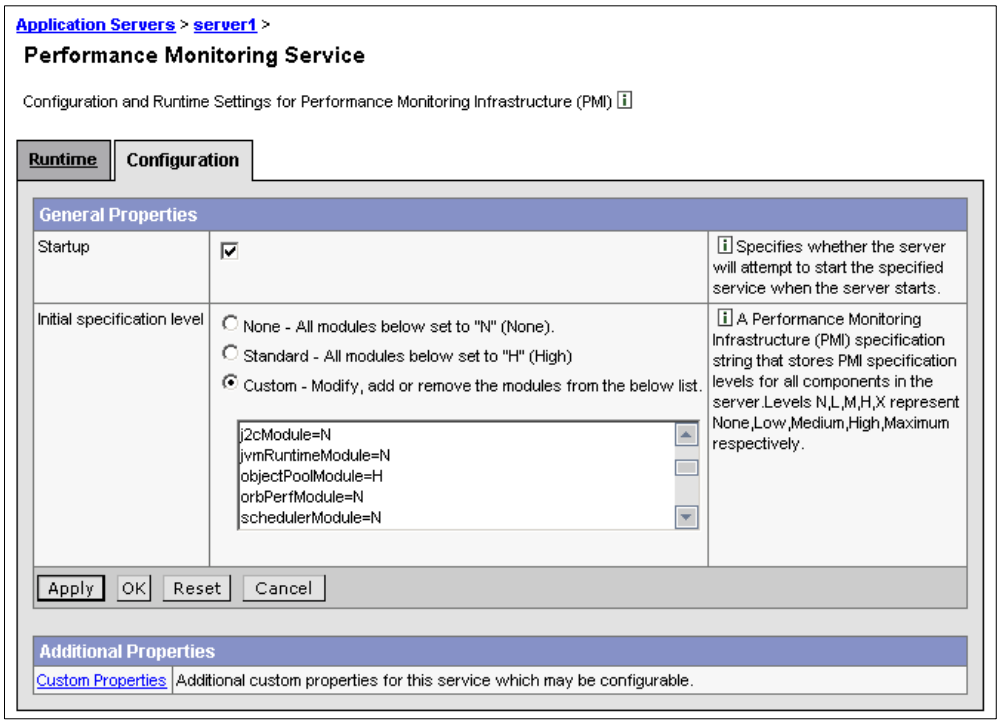


Figure 14-3 PMI configuration for object pool service

Important: There is no PMI available when using custom object pools.

15

Shared Work Area service

A Shared Work Area is a WebSphere Application Server Enterprise Programming Model Extension (PME) allowing you to pass user-defined information in a J2EE environment via a Java context.

This chapter discusses the usage of the Shared Work Area service in J2EE application and more specifically in WebSphere.

15.1 Planning

In the process of developing software applications, the need to pass data between application components is often a fundamental requirement. A significant part of this is contextual information such as security information, transaction context, locale information and business state data. When developing distributed applications, potentially spanning multiple logical and physical tiers, the process of passing context data around can be quite challenging. As a particular user request flows from the originating client through one or more application tiers, the information has to be always available to the current component for processing. Even if the information is not needed for a particular component, it must be preserved and propagated to components further down the process flow in case they need the information to complete their jobs.

The scenario in this chapter can be best explained using an example. In a typical application, a browser-initiated request might call a servlet, which then calls a session EJB that calls multiple Entity EJBs. In this example, the servlet, session EJB, and Entity EJB would all need access to security information, transactional information, and possibly other business information. Fortunately, in case of J2EE, the underlying protocol RMI/IIOP has the ability to implicitly carry private context information on the thread of execution without the developer having to write any code. This is how security and transaction contexts are propagated from an EJB client (a servlet in this example) to each EJB that it calls, as well as to each EJB that those EJBs call, whether the EJBs are on the same physical tier or separate ones.

This takes care of a large part of the “standard” contextual data that needs to flow from tier to tier, but what about the business data that might be necessary or at least convenient to have available to every component? For instance, when a user initiates a session it might be beneficial to look up some profile information and make that available to all the components that are called by the application. There are a couple of alternatives to achieve this. One would be to add a “custom profile” key or object to every method call so the providing component has access to the profile data. This would have to be done even in cases where the component did not need the data, but other components further down the line might need it to complete their functions. This alternative would have several negative side effects, including making every method call more complex, increasing maintenance complexity, and reducing the reuse potential of many EJBs. Clever design and/or use of a custom framework might reduce the severity of these side effects, but they would still pose significant challenges. In the case where components have been purchased from a third party, adapting pre-written components in this fashion might be literally impossible without participation from the component vendor.

A better alternative would be to have the underlying middleware “plumbing” carry the client profile information the same way that it carries security and transactional information. This provides a much simpler programming model as well as much more flexibility and maintainability.

The WebSphere Enterprise Shared Work Area service provides a simple, flexible solution to the problem outlined above. Using this service, developers can easily create a work area, insert data into it, and make remote invocations to EJBs. The work area will be propagated with each remote method invocation in the same way that the security and transaction contexts are. The receiving component may use the data, ignore it, or add more to it. If the receiving component calls methods on other components, the work area data will flow along to those components as well. When the original client is done with the work area, it terminates it.

A work area is defined as a set of properties, each of which contains a key (which uniquely identifies the element), a value (the actual data that needs to be propagated), and a mode (which indicates whether the data can be modified or deleted by downstream components). Work areas can also be nested to provide control over data visibility and even override specific properties.

Work areas are not only interesting to corporate customers, but can be very much interesting to Independent Software Vendors who need to build customizable applications and need solution to carry customized data along that might not be known by many of the components at development time.

15.2 Design

Work areas can hold any kind of information, and they can hold an arbitrary number of individual pieces of data. Each piece of data is represented by a property that consists of a key-value-mode triplet. The key-value pair represents the information contained in the property. The key is a name by which the associated value is retrieved. The mode determines whether the property can be removed or modified. Each piece of data can be accessed separately, so if the type or length of the data changes, only those methods that actually use that particular piece of data are affected by the change.

The Shared Work Area service has an important characteristic: the data can only be passed one way, from the caller to the remote method. If the remote method makes changes to the data, the changes are never seen by the caller. A remote method can be another servlet or a method in the remote interface of an EJB. Even if the servlet or bean resides in the same JVM, the call is still considered remote. It is important to remember this characteristic when designing applications using the Shared Work Area service.

There are two prime considerations in deciding whether to pass specific information explicitly as an argument or implicitly by using a work area:

- Pervasiveness

  How much of the given information is used in your application overall? Is it used in a majority of the methods in an application? The Shared Work Area service is best used to store data that is required by many parts of the application. If we pass this data as properties, method signatures would quickly become cluttered. Even methods that do not need the data may have to pass it along to other methods that require it. Also, if future enhancements require additional data to be passed or the type or length of the data must change, all methods that use or have to pass this data would have to change. By using shared work areas, only those methods that actually use the data would have to handle it. Because a shared work area is not explicitly passed, there are no extra properties for methods that do not need the data. In this sense, a work area is similar to the security or transaction context in WebSphere Application Server, but in contrast to those two, it is generalized and exposed for use in your application.

- Size

  Is it reasonable to send the information even when it will not be used? The size of information used with shared work areas is important from a performance point of view. Since we cannot always predict and control the size programmatically, WebSphere's implementation has built in some configurable limitations for the size of information stored in a shared work area.

15.3 Development

There is no special support for Shared Work Area services in WebSphere Studio IE. You only have to make sure that acwa.jar and distexcep.jar are in the application's Java classpath.

Important: In order to successfully compile your application and generate deploy code, acwa.jar and distexcep.jar files must be added to Java build path. In WebSphere Studio IE, the files are in the <*WebSphere_Studio_IE_root*>\WAS_EE_V5\lib directory. In WebSphere Enterprise V5, the files are in the <*WebSphere_root*>\lib directory.

15.3.1 Work Area API

The Work Area API is contained in com.ibm.websphere.workarea package. For more details about the API check complete description of Work Area API in the WebSphere Enterprise InfoCenter:

http://publib7b.boulder.ibm.com/wasinfo1/en/info/ee/javadoc/ee/com/ibm/websphere/workarea/package-summary.html

In the following sections, we describe parts of WebSphere Enterprise's work area implementation.

UserWorkArea

This is the interface that defines the work area class. It consists of the following methods:

- begin() and complete(): Start and terminate a work area in current thread.
- set(): Define or modify a property in the work area.
- get(): Get the defined properties.
- getName(): Return the name of a specified property.
- retrieveAllKeys(): Return all defined properties for the work area of caller's thread.
- remove(): Terminate the work area that was created (if any) in the caller's thread.

PropertyModeType

By this class you define permissions for each property in a work area. Possible values are:

- Normal

  You can modify, override (by a new one with the same name in a nested work area) and remove the property.

- Read-only

  The value of read-only property cannot be modified. However you can remove the property (only from the thread that created its work area).

- Fixed

  Property cannot be removed, but it can be modified and overridden.

- Fixed read-only

  Property cannot be removed, nor overridden, nor modified.

Modifying properties within a work area

For a particular work area, only the originator thread (the thread that created and "owns" the given work area) can modify, remove, or change mode of properties created in that work area.

Nested work areas

A nested work area is a work area created "over" another work area. Let's say you have a thread that got a work area from a remote caller. When your thread creates another work area, we call the new work area to be nested. Then using work area in the current thread, you will see all the properties. The originating work area, plus the properties from the newly created nested work area, will appear as one flat work area. The remote caller will not be aware of any nested work areas, since the originating thread will not see anything from the newly created nested work area.

In contrast, when you make a remote call to another thread from the current thread (the one that inherited a work area and then created a nested work area over), the called thread will get both work areas and again it will appear like one flat work area.

Nested work areas are also used to override the properties from the originating work areas. Since you cannot change properties in a work area if you are not the originator who created the work area, the only chance is to create a nested work area with the same property. And in the nested work area, you can only create the same property if the inherited property permits it, which it can do if its mode is not any of the read-only modes.

15.3.2 Steps for using the Shared Work Area service

The following steps have to be done in your application when programming for work areas.

Find the Shared Work Area service

The object work area is exposed in the JNDI name space. Nothing has to be configured in WebSphere. By default it is enabled and programmatically you can access it by doing a lookup on a context to find the Shared Work Area service. See Example 15-1.

Example 15-1 Shared Work Area service lookup

```
import com.ibm.websphere.workarea.*;
import javax.naming.InitialContext;
...
initialContext = new InitialContext();
workArea = (UserWorkArea)
```

```
initalContext.lookup("java:comp/websphere/UserWorkArea");
```

The Shared Work Area service is available within the WebSphere J2EE server environment and also within J2EE client container. In both cases the same procedure is used to get the service, it depends if you want to create or to use existing work areas within given thread just find the service.

Create a work area

When you have the Shared Work Area service you can create and associate a new work area for the current thread of execution. This is done by invoking the begin() method, as shown in Example 15-2.

Example 15-2 Create a work area

```
import com.ibm.websphere.workarea.*;
...
//find work area service
...
//create a new work area with name ACompanyWorkArea in the current thread
workArea.begin("ACompanyWorkArea");
```

This code creates a new work area named ACompanyWorkArea in the thread that invokes it. There are no restrictions on the name, except it has to be a non-null value.

If there are already existing work areas associated with the thread, a nested work area will be created. Refer to “Nested work areas” on page 564 for more details.

Use work area

So far, we have found a Shared Work Area service and defined work areas. The next step is to put properties with values into the work area. Remember, each property is represented by a name, value, mode triplet.

Set user defined properties

In Example 15-3 on page 566, we put two properties into the work area of the current thread: a property named productID and its value is represented with our own ProductProperties class. You can put whatever you want in the work area value as long as your user-defined class implements the java.io.Serializable interface. Also when we set the property, we did not specify its mode. By default the property’s mode is set to Normal.

Additionally we put a property named customerName with a value of John and also set the property to be Fixed (that is, it cannot be overridden by another nested work area) and Read-only (its value cannot be modified).

Example 15-3 Set user defined property

```
import com.ibm.websphere.workarea.*;
...
//find work area service
...
//define and init product properties
ProductProperties productProps = new ProductProperties();
//set user defined properties
ProductProperties productID = new ProductProperties();
workArea.set(“productProperties”, productProps);
workArea.set(“customerName”, “John”, PropertyModeType.fixed_readonly);
```

Important: Your classes must implement java.io.Serializable if you want to put them into the work area property.

Get defined properties

When we need to get a property from a work area, we simply invoke a get() method supplying the property name. Work area service will start searching on current threads' work area and nested areas. See Example 15-4.

Example 15-4 Get user defined properties

```
import com.ibm.websphere.workarea.*;
...
//find work area service
...
ProductProperties productProps = (ProductProperties)
workArea.get(“productProperties”);
```

Modify properties

When we want to modify the property, we simply invoke the set() method again with the same property name. You can only do that in the originator thread (the thread that created the work area). If you try to modify it in a “remote” thread (which got the context from the caller, the originating thread), you will get NotOriginator exception.

If you want to modify a property of the work area in a non-originator thread, the only way is to create another nested work area with the same property and this is only possible when the original property allows it to be overridden, if its mode is not read-only.

Important: For the given work area you can only modify the properties whose mode is either Normal or Fixed. And you can only modify them from the thread where the work area was created - the originator thread.

Change property mode

The mode of the property can only be changed from the originating thread, the thread that creates the property. In a non-originating thread, you can create a nested work area with the same property that would have other modes. Permissible mode changes depend on the original mode. See Table 15-1.

Table 15-1 Permissible mode changes in a nested work area

Mode in the workarea created by originating thread	Possible mode change in nested work area
Normal	All modes
Read-only	No change allowed
Fixed read-only	No change allowed
Fixed	Fixed, Fixed read-only

Remove properties

You can also remove properties already created. See Example 15-5. Only the originating thread, the thread that created it, can remove it. Otherwise you will get a NonOriginator exception. If the property's mode is set to Fixed, then you cannot remove it, not even from an originator thread. You will get a PropertyFixed exception.

Example 15-5 Get user defined properties

```
import com.ibm.websphere.workarea.*;
...
//find work area service
...
ProductProperties productProps = (ProductProperties)
workArea.remove(“productProperties”);
```

Terminate work area

When you no longer need a work area, you should terminate it. See Example 15-6. Again, you can only terminate a work area that was created in the current thread. If the current thread just has a work area inherited from a remote caller, then it cannot terminate it. It can only terminate its work areas and nested work areas.

Example 15-6 Terminate work area

```
import com.ibm.websphere.workarea.*;
...
//find work area service
...
```

```
//use work area
...
//terminate work area
workArea.complete();
```

15.4 Unit test environment

To use work areas, there is nothing special to configure in the WebSphere Studio IE test environment. The work area service is enabled by default.

15.5 Sample application

In this section, we will create a sample application, which consists of a stand-alone Java client named WorkAreaTestClient running in the J2EE client container, and of a remote EJB named ACompanyWorkAreaBean, running in WebSphere Application Server Enterprise V5. In the work area properties, we put the ComputerProperties class. This is a custom class that implements java.io.Serializable, so we can use it as a value in properties.

In this sample, we have a large company with some intranet applications. The employees can use some J2EE stand-alone application that is a client for some internal applications. There are many different computer systems from which the client is used and for statistical reasons or evaluation we want to collect the data about the client computer systems: processor type, OS type, and number of processors. These all are stored in the ComputerProperties class. Every client stores this class in a work area property, which is then propagated to the servers.

Prerequisites

The following prerequisites have to be met before starting with the sample application:

- For testing purposes, a WebSphere Enterprise test server needs to be created in WebSphere Studio IE.
- The developed EJB is going to be part of the ACompany project's ACompanyEJB module. It can reside in any EJB module you create in your project.

Create the application client project

The first step is to create an application client project:

1. In WebSphere Studio IE, select **File** -> **New** -> **Application Client Project**.

2. Select **Create J2EE 1.3 Application client project**. Enter the project name: `WorkAreaTestClient`. Select this project to be under **Existing Enterprise Application** and select **ACompany** for the application. Click **Finish**.

   If you do not have the ACompany project already created, you should create another project that will contain the ACompanyWorkAreaTest EJB.

3. In the J2EE Navigator view, right-click the **WorkAreaTestClient** project, then select **Properties**. In the Properties for WorkAreaTestClient window, switch to the Java Build Path and select the **Libraries** tab.

4. In the Libraries tab choose **AddVariable**, double-click the **WAS_EE_V5**, go to the lib directory and choose **acwa.jar** and **distexcep.jar**.

5. Apply the changes.

Create WorkAreaTestClient application client class

To add the main class into the application client project, follows these steps:

1. In the J2EE Navigator view, right-click the **appClientModule** in ObjectPtest, then select **New** -> **Class**.

2. In New Java Class window, enter `WorkAreaTestClient` as the value for the Name property. Leave all the other properties with their default values. Click **Finish**.

3. Copy the code from Example 15-7 into the WorkAreaTestClient.java class.

Example 15-7 WorkAreaTestClient.java

```
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import com.acompany.ejbs.*;
import com.ibm.websphere.workarea.*;

public class WorkAreaTest {
   public static void main(String[] args) {
   UserWorkArea workArea=null;
   ACompanyWorkAreaTest remoteEJB=null;
   ComputerProperties compProps_0 = new ComputerProperties("CSIR", "XIA", 1);
   ComputerProperties compProps_1 = new ComputerProperties("LETNI", "SWODNIW", 1);
   ComputerProperties compProps_2 = new ComputerProperties("ALOROTOM", "XINU", 2);
   ComputerProperties compProps_3 = new ComputerProperties("CRAPS", "SIRALOS", 3);
   //Find work area service
   try {
      InitialContext initalContext = new InitialContext();
      workArea = (UserWorkArea) initalContext.lookup("java:comp/websphere/UserWorkArea");
      ACompanyWorkAreaTestHome home = (ACompanyWorkAreaTestHome) PortableRemoteObject.narrow(
      initalContext.lookup("java:comp/env/ejb/ACompanyWorkAreaTest"),
         ACompanyWorkAreaTestHome.class);
      remoteEJB = home.create();
```

```
   } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
   }
   //create a work area
   workArea.begin("originatorWorkArea");
   System.out.println("WorkAreaTest: workArea "+workArea.getName()+" created\n");
   //set the properties
   try{
      workArea.set("clientComputerPropertiesNormal",compProps_0,PropertyModeType.normal);
      workArea.set("clientComputerPropertiesReadOnly",compProps_1,PropertyModeType.read_only);
      workArea.set("clientComputerPropertiesFixedNormal",
         compProps_2,PropertyModeType.fixed_normal);
      workArea.set("clientComputerPropertiesFixedReadOnly",
         compProps_3,PropertyModeType.fixed_readonly);
      } catch(PropertyReadOnly e) {
         e.printStackTrace();
      } catch(Exception e) {
         e.printStackTrace();
      }
      //do some tests on this work area remotely
      try{
         remoteEJB.changeInherited();
         remoteEJB.changeNested();
      } catch(Exception e) {
         e.printStackTrace();
      }
      //try to overwrite read only property
      try {
         workArea.set("clientComputerPropertiesReadOnly",compProps_2);
      } catch(PropertyReadOnly e) {
         System.out.println("WorkAreaTest: as expected got PropertyReadOnly when trying to
overwrite read only property\n");
      }
      catch(Exception e) {
         e.printStackTrace();
      }

      //try to overwrite fixed property
      try{
         workArea.set("clientComputerPropertiesFixedNormal",compProps_2);
         System.out.println("WorkAreaTest: as expected we could overwrite fixed normal
property from local work area\n");
      } catch(Exception e) {
         System.out.println("WorkAreaTest: unexpected exception");
         e.printStackTrace();
      }
      //complete work area
      workArea.complete();
```

```
   }
}
```

Create ComputerProperties application class

To add the main class into the application client project, do the following:

1. In J2EE hierarchy window, right-click the **ACompanyEJB** module in EJB Modules.
2. Select **New** -> **New** -> **Class**. In the New Java Class window, enter `ComputerProperties` as the value for the Name property. Uncheck Public Static Void Main(String args[]). Leave all the other properties at their default values. Click **Finish**.
3. Copy the code from Example 15-8 into the ComputerProperties.java class.

Example 15-8 ComputerProperties.java source

```
package com.acompany.ejbs;
public class ComputerProperties implements java.io.Serializable {
   protected String processorType;
   protected String osType;
   protected Integer numberOfProcessors;
   //default constructor
   public ComputerProperties() {
      this.processorType = new String("ABAKUS");
      this.osType = new String("OS4");
      this.numberOfProcessors = new Integer (1);
   }
   //another constructor
   public ComputerProperties(String pTyp, String osTyp, int nOfPr) {
      this.processorType = new String(pTyp);
      this.osType = new String(osTyp);
      this.numberOfProcessors = new Integer (nOfPr);
   }
   //setter method for proccessorType
   public void setProcType(String pTyp) {
      this.processorType = new String(pTyp);
   }
   //setter method for osType
   public void setOsType(String osTyp) {
      this.osType = new String(osTyp);
   }
   //setter method for numberOfProcessors
   public void setNumOfProc(int nOfPr) {
      this.numberOfProcessors = new Integer (nOfPr);
   }
   //getter method for proccessorType
   public String getProcType() {
```

```
        return(this.processorType);
    }
    //getter method for osType
    public String getOsType() {
        return(this.osType);
    }
    //getter method for numberOfProcessors
    public int getNumOfProc() {
        return(this.numberOfProcessors.intValue());
    }
}
```

The class is just a simple class that contains two String attributes, one for OS and one processor type, and an Integer for number of processor. Additionally it contains a few setter and getter methods for the attributes.

Change the main class of the application client

You have to change the specified main class of the client so that the J2EE client container will know which class to run when starting the application. Our main class is ObjectTest.java. The client container starts the program by starting its main() method.

1. In the J2EE Navigator view, select **ObjectPtest -> appClientModule -> META-INF** and double-click the **MANIFEST.MF** file.
2. In the Dependencies tab at the bottom, enter `WorkAreaTestClient` as the value for the Main-Class property.
3. Save and close the file.

Create ACompanyWorkAreaTest bean

To create ACompanyWorkAreaTest bean and all the necessary environment, do the following:

1. In the J2EE hierarchy window, right-click the **ACompanyEJB** module in the EJB Modules, then select **New** -> **Enterprise Bean**.
2. In the first Enterprise Bean Creation window, make sure that **ACompanyEJB** is selected for the project, and click **Next**.
3. Configure the first Enterprise Bean Creation window, as shown in Figure 15-1 on page 573, and click **Next**.

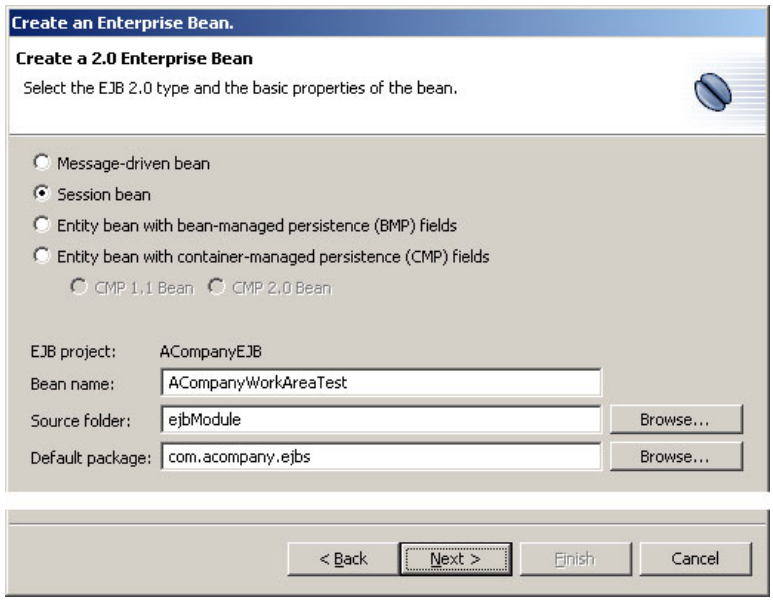


Figure 15-1 Creating ACompanyWorkAreaTest bean, first window

4. Configure the second Enterprise Bean Creation window, as shown in Figure 15-2 on page 574, and click **Finish**.

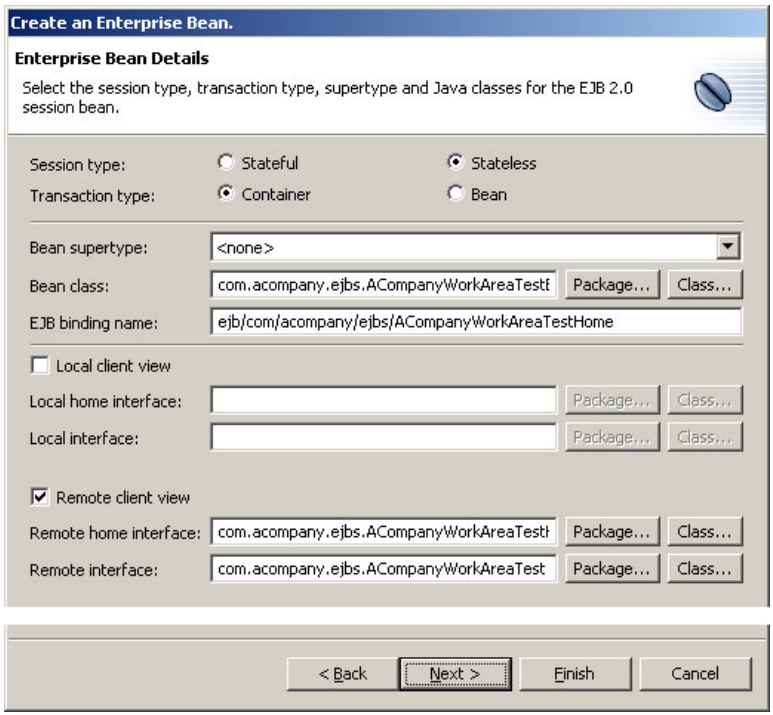


Figure 15-2 Creating ACompanyWorkAreaTest bean, second window

Adding code for the ACompanyWorkAreaTest remote interface

Open the ACompanyWorkAreaTest.java and copy the code from Example 15-9.

Example 15-9 ACompanyWorkAreaTest.java

```
package com.acompany.ejbs;
public interface ACompanyWorkAreaTest extends javax.ejb.EJBObject {
   public void changeInherited() throws java.rmi.RemoteException;
   public void removeInherited() throws java.rmi.RemoteException;
   public void changeNested() throws java.rmi.RemoteException;
}
```

Adding code for the ACompanyWorkAreaTest bean

Open ACompanyWorkAreaTestBean.java and copy the code from Example 15-10.

Example 15-10 ACompanyWorkAreaTestBean.java

```
package com.acompany.ejbs;
import javax.naming.InitialContext;
```

```
import com.ibm.websphere.workarea.*;
public class ACompanyWorkAreaTestBean implements javax.ejb.SessionBean {
   UserWorkArea workArea;
   ComputerProperties compProp = new ComputerProperties("ADA", "DONUX", 2);
   private javax.ejb.SessionContext mySessionCtx;
   public javax.ejb.SessionContext getSessionContext() {
      return mySessionCtx;
   }

   public void setSessionContext(javax.ejb.SessionContext ctx) {
      mySessionCtx = ctx;
   }

   public void ejbCreate() throws javax.ejb.CreateException {
      try {
         InitialContext initalContext = new InitialContext();
         //Find (inherited) work area
         workArea = (UserWorkArea) initalContext.lookup("java:comp/websphere/UserWorkArea");
         System.out.println("ACompanyWorkAreaTestBean: found workarea: " +
            workArea.getName());
         //list all the properties
         String[] properties = workArea.retrieveAllKeys();
         if (properties != null) {
            for (int i = 0; i < properties.length; i++)
               System.out.println("ACompanyWorkAreaTestBean: found " + properties[i] + "
                  property in workarea:" + workArea.getName());
         } else
            System.out.println("ACompanyWorkAreaTestBean: no properties found in workarea =" +
               workArea.getName());
      } catch (javax.naming.NamingException e) {
         System.out.println("ACompanyWorkAreaTestBean: lookup failed");
         e.printStackTrace();
      }
   }

   public void ejbActivate() {}

   public void ejbPassivate() {}

   public void ejbRemove() {}

   public void changeInherited() {
      ComputerProperties compProp = new ComputerProperties("NEUMMAN", "DONUX", 3);
      //try to modify normal property
      try {
         workArea.set("clientComputerPropertiesNormal", compProp);
      } catch (NotOriginator e) {
         System.out.println();
         System.out.println("ACompanyWorkAreaTestBean: as expected got NonOriginator
```

```
            exception");
         System.out.println("when trying to modify a normal property from " +
            workArea.getName());
      } catch (Exception e) {
         System.out.println("This exception was not expected !!!");
         e.printStackTrace();
      }
      //try to remove fixed normal property
      try {
         workArea.remove("clientComputerPropertiesNormal");
      } catch (NotOriginator e) {
         System.out.println();
         System.out.println("ACompanyWorkAreaTestBean: as expected got NonOriginator
            exception");
         System.out.println("when trying to remove normal property from " +
            workArea.getName());
      } catch (Exception e) {
         System.out.println("This exception was not expected !!!");
         e.printStackTrace();
      }
   }

   public void changeNested() {
      //Create a new nested work area; work area scope has now changed
      workArea.begin("nestedWorkArea");
      System.out.println();
      System.out.println("ACompanyWorkAreaTestBean: created new nested work area");
      //try to override read-only property
      try {
         System.out.println();
         System.out.println("ACompanyWorkAreaTestBean: trying to override read only property
            in" + workArea.getName());
         workArea.set("clientComputerPropertiesReadOnly", compProp);
      } catch (PropertyReadOnly e) {
         System.out.println("ACompanyWorkAreaTestBean: as expected got PropertyReadOnly
            exception");
         System.out.println("when trying to override a read-only property");
      } catch (Exception e) {
         System.out.println("This exception was not expected !!!");
         e.printStackTrace();
      }
      // overriding a fixed normal property
      try {
         System.out.println();
         System.out.println("ACompanyWorkAreaTestBean: overriding a fixed normal property");
         workArea.set("clientComputerPropertiesFixedNormal", compProp);
         System.out.println("ACompanyWorkAreaTestBean: as expected we could override a
            property");
```

```
         ComputerProperties props = (ComputerProperties)
workArea.get("clientComputerPropertiesFixedNormal");
         System.out.println();
         System.out.println("ACompanyWorkAreaTestBean: clientComputerPropertiesFixedNormal:
            procType=" + props.getProcType());
         System.out.println("ACompanyWorkAreaTestBean: clientComputerPropertiesFixedNormal:
            osType = " + props.getOsType());
         System.out.println("ACompanyWorkAreaTestBean: clientComputerPropertiesFixedNormal:
            numOfProc=" + props.getNumOfProc());
      } catch (java.lang.Exception e) {
         System.out.println("This exception was not expected !!!");
         e.printStackTrace();
      }
      //try to remove normal property - but now the scope is changed because of new work area
      try {
         workArea.remove("clientComputerPropertiesNormal");
      } catch (NotOriginator e) {
         System.out.println();
         System.out.println("ACompanyWorkAreaTestBean: as expected we got NotOriginator
            exception");
         System.out.println("when trying to remove normal property from " +
            workArea.getName());
      } catch (Exception e) {
         System.out.println("This exception was not expected !!!");
         e.printStackTrace();
      }
      //complete nested work area
      try {
         workArea.complete();
      } catch (Exception e) {
         System.out.println("This exception was not expected !!!");
         e.printStackTrace();
      }
   }
}
```

Use the application client

At the end, build the project and test the client, export the project into an EAR file, and use the **`launchClient`** command to run the project in J2EE client container.

1. Select the **ACompanyClient** project, and from the menu bar select **Project -> Rebuild Project**.
2. From the menu, select **Export**. The export wizard starts. Select **EAR file** on the first window, then click **Next**. Select **ACompany** as a project you want to export. Enter the file name to which you want to export the project, for example `C:\SG246932\ACompany.ear`. Click **Finish**.

3. Open a terminal or command prompt window. Change the directory where WebSphere's launchClient.bat resides in your system.
4. Make sure that the WebSphere Enterprise test server is running.
5. The client takes no arguments. Here is an example of how to run our client:

   ```
   launchClient C:\SG246932\ACompany.ear -CCjar=WorkAreaTestClient.jar
   -CCBootstrapHost=localhost -CCBootstrapPort=2809 -CCverbose=true
   ```

   The output you should get from the server is shown in Example 15-11 and the output you should get from the client is shown in Example 15-12 on page 579.

Example 15-11 Server system output when testing sample application

```
HttpTransport A SRVE0171I: Transport https is listening on port 9,043.
SchedulerServ I SCHD0001I: The Scheduler Service has started.
ConnectionFac I J2CA0107I: Component-managed authentication alias not specified for connection
factory or datasource BPEDataSourceCloudscape.
RMIConnectorC A ADMC0026I: RMI Connector available at port 2809
WsServer      A WSVR0001I: Server server1 open for e-business
SystemOut     O ACompanyWorkAreaTestBean: found workarea: originatorWorkArea
SystemOut     O ACompanyWorkAreaTestBean: found clientComputerPropertiesReadOnly property in
workarea:originatorWorkArea
SystemOut     O ACompanyWorkAreaTestBean: found clientComputerPropertiesNormal property in
workarea:originatorWorkArea
SystemOut     O ACompanyWorkAreaTestBean: found clientComputerPropertiesFixedReadOnly property
in workarea:originatorWorkArea
SystemOut     O ACompanyWorkAreaTestBean: found clientComputerPropertiesFixedNormal property in
workarea:originatorWorkArea
SystemOut     O
SystemOut     O ACompanyWorkAreaTestBean: as expected got NonOriginator exception
SystemOut     O when trying to modify a normal property from originatorWorkArea
SystemOut     O
SystemOut     O ACompanyWorkAreaTestBean: as expected got NonOriginator exception
SystemOut     O when trying to remove normal property from originatorWorkArea
SystemOut     O
SystemOut     O ACompanyWorkAreaTestBean: created new nested work area
SystemOut     O
SystemOut     O ACompanyWorkAreaTestBean: trying to override read only property
innestedWorkArea
SystemOut     O ACompanyWorkAreaTestBean: as expected got PropertyReadOnly exception
SystemOut     O when trying to override a read-only property
SystemOut     O
SystemOut     O ACompanyWorkAreaTestBean: overriding a fixed normal property
SystemOut     O ACompanyWorkAreaTestBean: as expected we could override a property
SystemOut     O
SystemOut     O ACompanyWorkAreaTestBean: clientComputerPropertiesFixedNormal: procType=ADA
SystemOut     O ACompanyWorkAreaTestBean: clientComputerPropertiesFixedNormal: osType = DONUX
SystemOut     O ACompanyWorkAreaTestBean: clientComputerPropertiesFixedNormal: numOfProc=2
```

Example 15-12 Client output when testing sample application

```
C:\Program Files\WebSphere\AppServer\bin>launchClient C:\ACompany.ear
-CCjar=WorkAreaTestClient.jar -CCBootstrapPort=2809 -CCverbose=true
IBM WebSphere Application Server, Release 5.0
J2EE Application Client Tool
Copyright IBM Corp., 1997-2002
WSCL0012I: Processing command line arguments.
WSCL0001I: Command line, property file, and system property arguments resolved to:
        File to launch          = C:\ACompany.ear
        CC Property File        = null
        Client Jar File         = WorkAreaTestClient.jar
        Alternate DD            = null
        BootstrapHost           =
        BootstrapPort           = 2809
        Trace enabled           = false
        Tracefile               = null
        Init only               = false
        Classpath Parameter     = null
        Security Manager        = disable
        Security Manager Class  = Not used. -CCsecurityManager=disable
        Security Manager Policy = Not used. -CCsecurityManager=disable
        Exit VM                 = false
        Soap Connector Port     = null
        Application Parameters  =
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0025I: Binding EJB reference object:
           JNDI name: ejb/ACompanyWorkAreaTest ==>
ejb/com/acompany/ejbs/ACompanyWorkAreaTestHome @ corbaloc:iiop:localhost:2809
           Description:
WSCL0031I: The object was bound successfully.
...
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
WSCL0014I: Invoking the Application Client class WorkAreaTest

WorkAreaTest: workArea originatorWorkArea created

WorkAreaTest: as expected got PropertyReadOnly when trying to overwrite read only property

WorkAreaTest: as expected we could overwrite fixed normal property from local work area
```

If you check through the code and the outputs, you find the following:

1. In the beginning, the client finds the Shared Work Area service and the remote EJB. The client starts a work area named originatorWorkArea and creates properties for it, each with a different mode and using ComputerProperties classes as a property value.

2. The client calls the changeInherited() remote method on the ACompanyWorkAreaTest EJB that is running on the server. At this time, the client's work area gets propagated to the EJB. In the create() method, we can find the work area and print out the properties available.
3. The server EJB tries to modify a normal property using a set() method. Since the server did not create and put the property into the (inherited) work area, we receive a NotOriginator exception, as expected, and we intercept it.
4. The server EJB tries to remove a property. Since the server did create and put the property into the (inherited) work area, we get a NotOriginator exception, as expected, and we intercept it.
5. The client calls the changeNested() remote method on the ACompanyWorkAreaTest EJB. The Work area gets propagated. Execution moves to the server. In this method call, the server starts with creating another (nested) work area.
6. The method tries to override a read-only property and as expected we get a ReadOnly exception.
7. The code successfully overrides a fixed normal property. Therefore, we see a new value in the nested work area under this property. The original property remains unchanged on the client side. It does not see the nested work area.
8. The server completes (closes) the work area created in the changeNested() call and exits, and the execution returns to the client. Note that after we complete the work area, the overridden property is gone.
9. The client tries to overwrite a read-only property. As expected, we receive and intercept a PropertyReadOnly exception.
10. The client tries to overwrite a fixed property. It succeeds as expected.
11. Finally, the code completes the work area and exits.

15.6 Configuration

This section explains the configurational procedures for the the Shared Work Area service and its dependencies. Handling and managing shared work areas is all done programmatically from the J2EE application. In administering terms, only a few properties of the Shared Work Area service can be configured.

15.6.1 Shared Work Area service configuration

For the Shared Work Area service configuration, do the following:

1. Launch the Administrative Console and select **Servers -> Application Servers**. Select the application server where you want to configure the Shared Work Area service.
2. In the Configuration tab, click **Work Area Service**. You will get the work area configuration window as shown in Figure 15-3.

   Here you can enable or disable the Shared Work Area service. By default it is enabled. The service starts during WebSphere startup. If you have it disabled and then enable it while the server is running, the server must be restarted to start the service.

   Additionally, you can set the maximum send and the maximum receive size (in bytes) of a single work area.

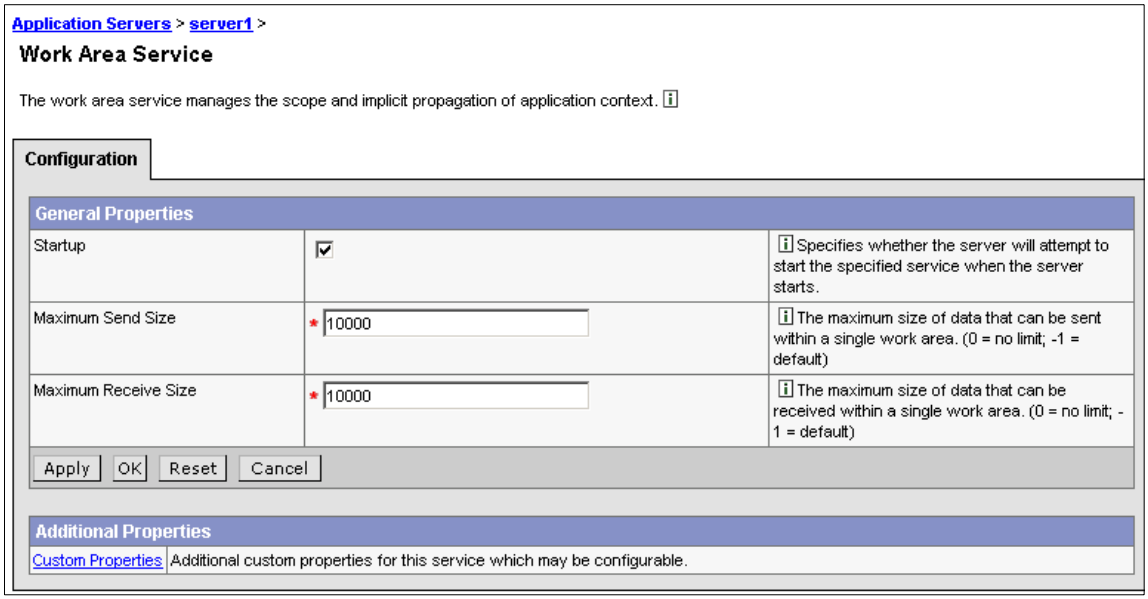


Figure 15-3 the Shared Work Area service configuration

15.6.2 Shared work area client properties

When using shared work areas from a client running in a J2EE client container, you should be able to specify some properties affecting the behavior. In this case the properties are specified by using the “-D” JVM directives. You need to edit the

launchClient script and add the property to the Java invocation line. The following properties are available:

- -Dcom.ibm.websphere.workarea.maxSendSize

  This sets the maximum send size of a single shared work area. The property type is integer and has a default value of 32767 bytes.

  An example of specifying this parameter would be:

  ```
  -Dcom.ibm.websphere.workarea.maxSendSize=10000
  ```

- -Dcom.ibm.websphere.workarea.enabled

  This enables or disables the use of Shared Work Area within your client application. The property type is boolean and has a default value of TRUE, which means the work area is enabled by default.

15.7 Problem determination and troubleshooting

If you encounter problems using the Shared Work Area service and you suspect that the problem source is somewhere within the service, then you can also use the following trace string to monitor what is happening in the execution time:

```
com.ibm.ws.workarea.*=all=enabled
```

16

Internationalization (i18n) service

The Internationalization service provides a mechanism for propagating locale and time zone information from clients to servers and between server components. This information can be used by server application components to customize the results according to the client locale and time zone. The transfer of the locale and time zone information is done transparently by the Internationalization (i18n) service. This chapter covers when to use the Internationalization service, how to develop the application that uses this service, and configuration and troubleshooting of the service.

16.1 Planning

This chapter describes the value of the Internationalization service, the problems of existing solutions for internationalization, then introduces the WebSphere Enterprise Internationalization service.

Internationalization (i18n)

An application that can present information to users according to regional cultural conventions is said to be *internationalized*. The application can be configured to interact with users from different localities in culturally appropriate ways. In an internationalized application, a user in one region sees error messages, output, and interface elements in the requested language. Date and time formats, as well as currencies, are presented appropriately for users in the specified region. A user in another region sees output in the conventional language or format for that region.

Internationalization of an application is driven by two variables: the time zone and the locale. The time zone indicates how to compute the local time as an offset from a standard time, such as Greenwich Mean Time (GMT). The locale is a collection of information about language, currency, and the conventions for presenting such information as dates. In a localized application, the locale also indicates the message catalog from which an application is to retrieve message strings. A time zone can cover many locales, and a single locale can span time zones. With both time zone and locale, the date, time, currency, and language for users in a specific region can be determined.

Why use the Internationalization service?

In a distributed client/server environment, application processes may run on different machines configured to different locales corresponding to different culture conventions. They may also be located across geographical boundaries. With the advent of Internet-based business computational models, such as e-commerce, more and more clients and servers will operate in different locales and geographical regions.

Internationalization techniques have traditionally been expensive and difficult to implement, so they have been applied only to major development efforts. However, given the rise in distributed computing and in the use of the World Wide Web, application developers have been pressured to internationalize a much wider variety of applications. This requires making internationalization techniques much more accessible to application developers.

Computers located in different locales

Client and server processes can run on computers that have a different locale setting. For example, a Spanish client may invoke a business method on an object that resides on an American server. Some business methods can be locale-sensitive. For example, a business method may return a sorted list of strings; the Spanish client will expect that list to be sorted according to the Spanish collating sequence, not in the server's English collating sequence. Since data retrieval and sorting procedures run on the server, the locale of the client has to be available in order to perform a legitimate sort. A server may also have to return strings containing date, time, currency, or exception messages formatted according to the client's locale.

Computers located in different time zones

Client and server processes can execute in geographical locations having different time zones. For example, suppose a vendor makes the claim that orders received before 2:00 PM will be processed by 5:00 PM the same day. The times given, of course, are in the time zone of the server that is processing the orders. It is important to know the client's time zone in order to give customers in other time zones the correct times for same-day processing. Other time zone-sensitive operations include time stamping messages, order tracking, transaction completion time, and estimated shipment arrival times.

16.1.1 The traditional solutions and the limitations

The conventional solution for solving locale and time zone mismatch problems is to pass one or more extra parameters on all business methods necessary for conveying either the client's locale or time zone to the server. Although the technique is simple, it has serious limitations:

- Parameter lists become longer.
- Extra parameters may need to be added to all methods to pass the information through the call chain even if they do not use the information.
- Adding extra parameters to deployed applications is inherently error-prone.
- It may be impossible to add parameters to the applications that are not easily modified, such as legacy applications.

16.1.2 The Internationalization service solution

The WebSphere Internationalization service solves the locale and time zone mismatch problems without the traditional limitations. The locale list and time zone are managed as a unit, referred to as an internationalization context. The Internationalization service manages the distribution of internationalization context across the various components of Enterprise applications, including Java

client applications, EJBs, JSPs, and servlets. The server-side components can use the internationalization context API to access the distributed internationalization context and then localize computations to the locale or time zone of the client-side components.

16.2 Design

The service associates an internationalization context with each thread of execution in an application. When a client-side program invokes a remote business method, the Internationalization service obtains the context associated with the current thread and attaches it to the outgoing request. On the server side, the Internationalization service detaches the caller's context from the incoming request and associates it with the remote business method thread. The service propagates this context on subsequent remote business method invocations to pass the context of the original request down the call chain.

16.2.1 Internationalization context

An internationalization context is a distributable collection of internationalization information containing an order list, or chain, of locales and a single time zone, where the locales and time zone are instances of Java SDK types, java.util.Locale and java.util.TimeZone. A locale chain is ordered according to the user's preference.

- java.util.Locale:

  The Locale object is defined by a language and optional country and variant. An application can use Locale to properly format messages or sort string lists, for example, by passing it to the locale-sensitive methods of other Java SDK objects, such as java.text.NumberFormat, java.text.DateFormat or java.text.Collator.

- java.util.TimeZone:

  The TimeZone represents a time zone offset from GMT. An application can use TimeZone to perform time-sensitive computations. For example, the constructors of java.util.Calendar accept a TimeZone parameter that configures the resulting Calendar objects to compute time according to the supplied time zone and daylight savings rules.

Note: The Internationalization service does not support the time zone types other than java.util.SimpleTimeZone. Unsupported time zone types map to the default time zone of the JVM when supplied to internationalization context API methods.

The Internationalization service manages and makes available two varieties of internationalization context: the caller context representing the caller's localization environment, and the invocation context representing the local environment under which a business method executes. The server application components use elements of the caller and invocation internationalization contexts to appropriately tailor locale-sensitive and time zone-sensitive computations.

Caller context

Caller internationalization context contains the local chain and time zone received on incoming EJB business method and servlet service method invocations. It is the internationalization context propagated from the calling process. Use caller context elements within server application components to localize computations to the calling component. Caller context is read-only and can be accessed by all application components by using the Internationalization interface of the internationalization context API.

Caller context is computed in the following manner:

- On an EJB business method or a servlet service method and servlet service method invocation, the Internationalization service extracts the internationalization context from the incoming request and scopes this context to the method as the caller context.
- On the servlets, the caller locales are retrieved from the Accept-Language header field in the HTTP request. This field depends the language setting of the Web browser. However, on the servlets, the caller time zone is always GMT because there is no space to put the time zone information in the HTTP request.

Invocation context

The invocation internationalization context contains the locale chain and time zone under which EJB business methods and servlet service methods execute. It is managed by either the hosting container or the application component, depending on the applicable internationalization policy. On outgoing business method requests, it is the context that propagates to the target process. Use invocation context elements to localize computations under the specified settings of the current application component.

The invocation context is computed in the following manner. On an incoming business method or servlet service method invocation, the Internationalization service queries the associated context management policy. If the policy is container-managed internationalization (CMI), the container scopes the context designated by the policy to the invocation. Otherwise the policy is application-managed internationalization (AMI), and the container scopes an

empty context to the invocation that can be altered by the method implementation.

The application components can access invocation context elements through both the Internationalization and InvocationInternationalization interfaces of the internationalization context API. The invocation context elements can be overwritten under the application-managed internationalization policy only.

On an outgoing business method request, the service obtains the currently scoped invocation context and attaches it to the request. This outgoing exported context becomes the caller context of the target invocation. When supplying the invocation context elements, either for export on outgoing requests or through the API, the Internationalization service always provides the most recent element set using the API.

Note: For any missing or null context element, the service inserts the corresponding default element of JVM (for example, java.util.Locale.getDefault() and java.util.TimeZone.getDefault()). The default elements of JVM are determined by the settings of the operating system.

16.2.2 Internationalization type

Every server application component that runs on WebSphere Enterprise has one internationalization type setting. By this setting, the server container decides that the invocation internationalization context is managed by the application component or by the hosting J2EE container.

The server application components can be deployed to use one of the following types of internationalization context management:

- Application-managed internationalization (AMI)
- Container-managed internationalization (CMI)

Application-managed internationalization (AMI)

Under the AMI deployment policy, component developers assume complete control over the invocation internationalization context. AMI components can use the internationalization context API to programmatically set invocation context elements.

The AMI components are expected to manage the invocation context. Invocations of AMI components implicitly run under the default locale and time zone of the hosting JVM. If the invocation context elements are not set by using the API, the container gets the JVM default values when accessed through the API or when exported on business methods. To export context elements other than the JVM defaults, AMI servlets, AMI EJB and EJB client application must

overwrite invocation elements using the internationalization context API. To continue propagating the context of the calling process, AMI servlets and AMI EJBs have to use the API to transfer the caller context elements to the invocation context.

Container-managed internationalization (CMI)

Under the CMI, the Internationalization service collaborates with the Web and EJB containers to set the invocation internationalization context for the servlets and EJBs. The service sets invocation context according to the container internationalization attribute of the policy associated with a servlet (service method) or an EJB business method.

The methods within CMI components can obtain elements of the invocation context using the internationalization context API, but cannot set them. Any attempt to set invocation context elements within CMI components results in a java.lang.IllegalStateException.

Internationalization attribute

A CMI policy includes a container internationalization attribute that indicates which internationalization context the container is to scope to an invocation. The main field of the container internationalization attribute is the Run-as field. The Run-as field specifies one of three types of invocation context that a container can scope to a method. For the servlet service and EJB business methods, the container constructs the invocation internationalization context according to the Run-as field and associates this context to the current thread before delegating to the method's implementation. The invocation context types specifiable with the Run-as field are following:

- Caller

  The container invokes the method under the internationalization context of the calling process. Select run as caller when you want the invocation to execute under the invocation context of the calling process.

- Server

  The container invokes the method under the default locale and time zone of the JVM. Select run as server when you want the invocation to execute under the invocation context of the JVM.

- Specified

  The container invokes the method under the internationalization context specified in the attributes. Select run as specified when you want the invocation to execute under the custom invocation context specified in the

policy, then provide the custom context elements by completing the Locales and Time zone ID fields:

– Locales field

The Locales field specifies an ordered list of locales that the container scopes to an invocation. You can put one or more locales in this field. A locale presents a specific geographical, cultural, or political region and contains three fields:

- Language code

  The language code is one of the lowercase, two-character codes defined by ISO-639. However, the language code is not restricted to ISO codes and is not a required field. A valid locale has to specify a language code if it does not specify a country code.

- Country code

  The country code is one of the uppercase, two-character codes defined by ISO-3166. However, the country code is not restricted to ISO codes and is not a required field. A valid locale has to specify a country code if it does not specify a language code.

- Variant

  The variant is a vender-specific code. The variant is not a required field and serves only to supplement the language and country code files according to application-specific or platform-specific requirements.

– Time zone ID field

The Time zone ID field specifies a shorthand identifier for a time zone that the container scopes to an invocation. A time zone represents a temporal offset and computes daylight savings information. A valid ID indicates any time zone supported by the Java 2 SDK type, java.util.TimeZone. Specifically, a valid ID is any of the IDs in the list of the time zone IDs returned by method java.util.Timezone.getAvailableIds(), or a custom ID having the form GMT[+|-]hh[[:]mm]. For example, America/Los_Angeles, GMT-08:00 are valid time zone IDs.

By default, invocations of servlet service methods and EJB business methods implicitly run as Caller.

Applicable settings

The applicable settings of the internationalization type depend on the component type. The following list shows the applicable settings for each type of components:

- The servlets, session beans and message-driven beans can be deployed as AMI or CMI, but not both. CMI is the default.

- The internationalization type of the entity beans is CMI and it cannot be configured.
- The EJB client applications do not have an internationalization type setting, but are implicitly AMI.

16.3 Development

In this section, we show you how to use the internationalization context API. To develop the application that uses the internationalization context API, you have to put the i18nctx.jar file in your classpath. This file is in the <*WebSphere_root*>/lib directory.

16.3.1 The internationalization context API

The applications use the API to access and manage internationalization context. Three interfaces are provided by the com.ibm.websphere.i18n.context package for this purpose:

- com.ibm.websphere.i18n.context.UserInternationalization
- com.ibm.websphere.i18n.context.Internationalization
- com.ibm.websphere.i18n.context.InvocationInternationalization

The UserInternationalization interface

The UserInternationalization interface provides a factory for obtaining the internationalization context API objects. These objects give access to the desired type of an internationalization context (caller or invocation contexts). This interface is shown in Example 16-1.

Example 16-1 The UserInternationalization interface

```
public interface UserInternationalization {
   public Internationalization getCallerInternationalization();
   public InvocationInternationalization getInvocationInternationalization();
}
```

The UserInternationalization interface defines two methods:

- Internationalization getCallerInternationalization()

  This method returns an object implementing the Internationalization interface. This interface allows read-only access to the caller context. If the service is disabled, the method throws a java.lang.IllegalStateException.

- InvocationInternationalization getInvocationInternationalization()

  This method returns an object implementing the InvocationInternationalization interface. This interface allows read and write access to the invocation context according to the internationalization context management policies. If the service is disabled, the method throws a java.lang.IllegalStateException.

The Internationalization interface

The Internationalization interface allows read-only access to the internationalization context. This interface is shown in Example 16-2.

Example 16-2 Internationalization interface

```
public interface Internationalization {
   public java.util.Locale[] getLocales();
   public java.util.Locale getLocale();
   public java.util.TimeZone getTimeZone();
}
```

The Internationalization interface provides three methods for read-only access to the internationalization context elements:

- java.util.Locale[] getLocales()

  This method returns the array of locales associated with the current thread. If the array of locales is null, the method returns an array containing the default locale of the process associated with the execution of this method. The method is useful if the first locale in the array is not recognized or not supported by the server. In this case, a different locale from the array could be selected.

- java.util.Locale getLocale()

  This method returns the first element from the array of locales associated with the current thread. If the array of locales is null, the method returns the default locale of the process associated with the execution of this method.

- java.util.TimeZone getTimeZone()

  This method returns the java.util.SimpleTimeZone object associated with the current thread. If the time zone is null, the method returns the default time zone of the process associated with the execution of this method.

The InvocationInternationalization interface

The InvocationInternationalization interface allows read and write access to the invocation internationalization context. Use this interface to read or modify the invocation internationalization context. The server component can access the

invocation internationalization context if its internationalization type is AMI. This interface is shown in Example 16-3.

Example 16-3 The InvocationInternationalization interface

```
public interface InvocationInternationalization extends Internationalization {
   public void setLocales(java.util.Locale[] locales);
   public void setLocale(java.util.Locale locale);
   public void setTimeZone(java.util.TimeZone timeZone);
   public void setTimeZone(String timeZoneId);
}
```

Since the InvocationInternationalization interface extends the Internationalization interface, all methods from the Internationalization interface are available in the InvocationInternationalization interface. In addition, the InvocationInternationalization interface defines four more methods:

- void setLocales(java.util.Locale[] locales)

  This method sets the array of locales associated with the current thread to the supplied array of locales. The supplied locale array can be null or have a length greater than or equal to zero. When the supplied locale array is null or has a length of zero, the Internationalization service sets the array of locales to an array length of 1 containing the default locale of the process associated with the execution of this method. Null entries can exist within the supplied array of locale. However, on the remote method invocation, the Internationalization service substitutes all null elements with the default locale of the default locale of the process associated with the execution of this method.

- void setLocale(java.util.Locale locale)

  This method sets the array of locales associated with the current thread to an array of length 1 containing the supplied locale. If the supplied locale is null, the service sets the array of locales to an array of length 1 containing the default locale of the process associated with the execution of this method.

- void setTimeZone(java.util.TimeZone timeZone)

  This method sets the invocation time zone associated with the current thread to the supplied time zone. If the supplied time zone is null or not an instance or subclass of java.util.SimpleTimeZone, the service sets the time zone to the default time zone of the process associated with the execution of this method.

- void setTimeZone(String timeZoneId)

  This method sets the invocation time zone associated with the current thread to java.util.SimpleTimeZone with the supplied time zone ID. The general format for the time zone ID is either country/city or GMT[+|-]hh[[:]mm]. For example, the time zone ID for the U.S. Pacific time zone is

America/Los_Angeles or GMT-08:00. If the supplied time zone ID is null or unsupported, the Internationalization service sets the time zone to a time zone having an ID of GMT and the default offset of the process is associated with the execution of this method. The use of three-letter time zone IDs other than GMT is deprecated. A list of supported time zone IDs can be obtained by using the java.util.TimeZone.getAvailableIds() method. See the Java 2 SDK API documentation for more information about the time zone IDs.

16.3.2 Using the Internationalization service

This section shows you how to use the Internationalization service by using a sample application. This application consists one stateless session bean and one EJB client. On the stateful session bean, we explain how to retrieve the caller internationalization context. On the EJB client, we explain how to retrieve and set the invocation internationalization context.

To use the internationalization context, we have to:

1. Bind to the Internationalization service.
2. Use the UserInternationalization interface to retrieve the API objects that afford access to the desired internationalization context types.
3. Access the desired context type.

Once the context is accessed, we can use it with the time zone and locale-sensitive operations.

Retrieving and Using the caller Internationalization context

Using the ejbCreate() method, the Internationalization service is bound. It is looked up from the JNDI repository. Once the Internationalization service is bound, we can retrieve the caller internationalization context. From the caller internationalization context, we can retrieve the caller locale and time zone and use them for the time zone and locale-sensitive operations. There is no need to pass the locale and time zone as a parameter to the method. The sample codes are shown in Example 16-4.

Example 16-4 Retrieving and using the caller internationalization context

```
package com.ibm.itso.i18n;

import java.text.*;
import java.util.*;
import javax.ejb.EJBException;
import javax.naming.*;
import com.ibm.websphere.i18n.context.*;

public class I18nHelloBean implements javax.ejb.SessionBean {
```

```
   private UserInternationalization i18nService = null;
   .....
   public void ejbCreate() throws javax.ejb.CreateException {
      try {
         InitialContext ctx = new InitialContext();
         i18nService =
            (UserInternationalization) ctx.lookup(
               "java:comp/websphere/UserInternationalization");
      } catch (NamingException ne) {
         throw new EJBException(ne);
      }
   }
   .....
   public String greeting() {
      Internationalization callerI18n =
         i18nService.getCallerInternationalization();
      Locale callerLocale = callerI18n.getLocale();
      TimeZone callerTimeZone = callerI18n.getTimeZone();
      DateFormat df =
         DateFormat.getDateTimeInstance(
            DateFormat.FULL, DateFormat.FULL, callerLocale);
      df.setTimeZone(callerTimeZone);
      ResourceBundle resource =
         ResourceBundle.getBundle("com.ibm.itso.i18n.myResource",
callerLocale);
      Object[] obj = { df.format(new Date())};
      return new MessageFormat(resource.getString("greetStr")).format(obj);
   }
}
```

Retrieving and setting invocation internationalization context

The Internationalization service is bound to the JNDI. Then the invocation internationalization context is retrieved from the UserInternationalization interface. On the EJB client, we can set the locale and time zone to the invocation internationalization context. This context is passed to the server-side components and retrieved as the caller internationalization context. The sample code is shown in Example 16-5.

Example 16-5 Retrieving and setting the invocation internationalization context

```
package com.ibm.itso.i18n;

import java.util.*;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import com.ibm.websphere.i18n.context.*;

public class I18nHelloClient {
```

```
    ....
    public void callGreeting(Locale locale, TimeZone timeZone) {
       try {
          InitialContext ctx = new InitialContext();
          UserInternationalization i18nService =
             (UserInternationalization) ctx.lookup(
                "java:comp/websphere/UserInternationalization");
          InvocationInternationalization invI18n =
             i18nService.getInvocationInternationalization();
          invI18n.setLocale(locale);
          invI18n.setTimeZone(timeZone);

          I18nHelloHome home =
             (I18nHelloHome) PortableRemoteObject.narrow(
                ctx.lookup("java:comp/env/ejb/I18nHello"),
                I18nHelloHome.class);
          System.out.println(home.create().greeting());
       } catch (Exception ex) {
          ex.printStackTrace();
       }
    }
}
```

16.3.3 Enhanced Internationalization Service

The enhanced Internationalization service extends the Internationalization service to support the internationalization on the Web Services applications by using the same concepts. It is provided as the technology preview.

The enhanced service transparently propagates internationalization context over the requests originating from the J2EE Web Services clients. On an outgoing Web Services request, the service creates a SOAP header block containing the invocation context associated to the current thread and this SOAP representation is then inserted into the outgoing request.

When a request arrives at the server side, the service scopes the propagated internationalization context, referred to as the caller context, to the invocation of the stateless session bean enabled as a Web service. It also scopes an invocation context to the invocation in case the internationalization context management policies are CMI and run as Caller.

To perform the localizations, Web services-enabled EJBs will obtain the elements of either context using the internationalization context API and utilize them within locale- or time zone-sensitive operations.

The Internationalization service is bound to the JNDI. Then invocation internationalization context is retrieved from the UserInternationalization interface. On the Web Services client, we can set the locale and time zone to the invocation internationalization context. This context is passed to the server side components and retrieved as the caller internationalization context. The sample code is shown in Figure 16-6.

Example 16-6 The Web Services client using the enhanced Internationalization service

```
package com.ibm.itso.i18n;

import java.util*;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import javax.xml.namespace.QName;
import com.ibm.websphere.i18n.context.*;

public class I18nHelloServiceClient {
   ........
   public void callGreeting(Locale locale, TimeZone timeZone) {
      try {
         InitialContext ctx = new InitialContext();
         UserInternationalization i18nService =
            (UserInternationalization) ctx.lookup(
               "java:comp/websphere/UserInternationalization");
         InvocationInternationalization invI18n =
            i18nService.getInvocationInternationalization();
         invI18n.setLocale(locale);
         invI18n.setTimeZone(timeZone);

         I18nHelloSEIService service =
            (I18nHelloSEIService) ctx.lookup(
               "java:comp/env/service/I18nHelloSEIService");
         QName portQName =
            new QName("http://i18n.itso.ibm.com", "I18nHelloSEI");
         I18nHelloSEI port =
            (I18nHelloSEI) service.getPort(portQName, I18nHelloSEI.class);

         System.out.println(port.greeting());
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
}
```

To set up the enhanced Internationalization service, see “Install Enhanced Internationalization Service Technology Preview” on page 624.

Note: Use this technology preview if you want to gain experience with the latest standards or if you are beginning a development project that will not be deployed until full support is available.

16.4 Unit test environment

There is no tooling support and no configuration support in WebSphere Studio Application Developer Integration Edition. By default the Internationalization service is disabled. If you are planning to test the Internationalization service on the unit test environment, you have to enable it. The steps to enable it are as follows:

1. Open the server configuration.
2. Switch to the Configuration tab and make sure that the Administrative Console is enabled.
3. Start the test server, then right-click the server and select **Run administrative console**.

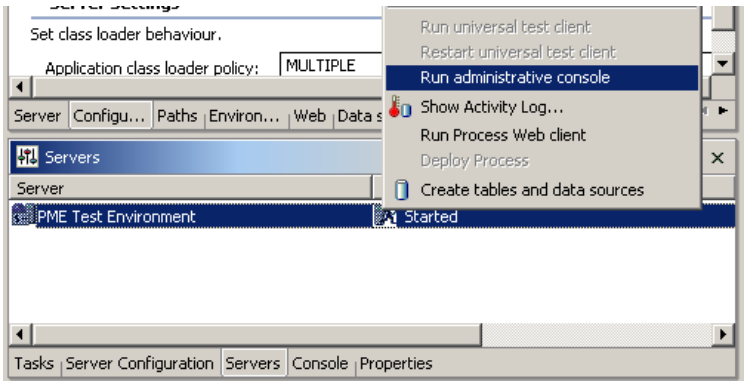


Figure 16-1 Run the Administrative Console on WebSphere Studio IE

4. When the Administrative Console comes up, follow the steps in 16.8, “Configuration” on page 622.
5. Import the SchedulerCalendars.ear from the <*WebSphere_Studio_IE_root*>\runtimes\ee_v5\installableApps directory. Use the project name: `SchedulerCalendars`.
6. Open the properties for the Calendar EJB project and add the scheduler-client.jar to the Java Build Path. The library is under the WebSphere Enterprise \lib directory. You can use the WAS_EE_V5 variable.

7. Select the Calendars EJB project in the navigator view, then select **Select Project -> Rebuild Project** from the menu.
8. Add the application to the server configuration. You will have to stop the test server to perform this step.

16.5 Assembly

You can specify the internationalization type and attribute by using the Application Assembly Tool. This section shows you how to set them. These steps are optional. If you don't set them, the default settings are CMI and run as Caller.

16.5.1 Specify the internationalization type

The servlets, session bean, and message-driven bean have an internationalization type setting that specifies whether internationalization context is managed by the application component or by its hosting J2EE container during invocation of their respective life cycles and business methods.

Setting the internationalization type for EJB

The steps to set the internationalization type for an EJB are the following:

1. Start the Application Assembly Tool.
2. Open your J2EE archive file.
3. Select the application name and click **EJB Modules.** Select the EJB module name and click **[Session Beans | Message Driven Beans]** and select the EJB name in the left-hand pane.
4. Click the **WAS Enterprise** tab in the right-hand pane.
5. Select **Container** for CMI or **Application** for AMI in the Internationalization Type field.

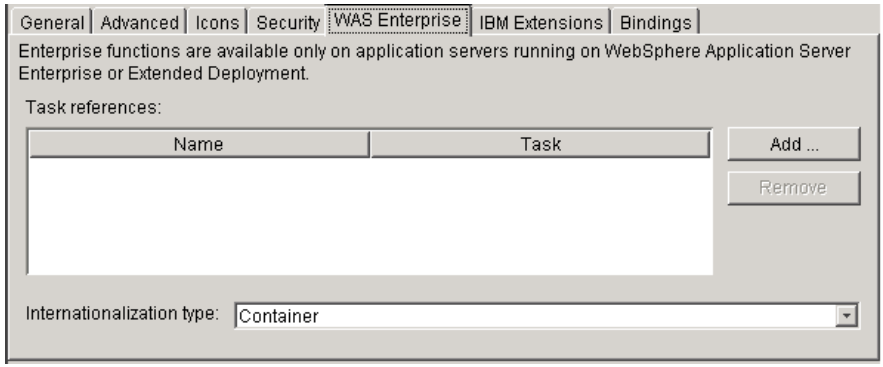


Figure 16-2 Setting internationalization type for EJB

6. Click **Apply**.

Setting the internationalization type for servlet

The steps to set the internationalization type for a servlet are as follows:

1. Start the Application Assembly Tool.
2. Open your J2EE archive file.
3. Select the application name and click **Web Modules**. Select the Web module name and click **Web Components** and select the Servlet name in the left-hand pane.
4. Click the **WAS Enterprise** tab in the right-hand pane.
5. Select **Container** for CMI or **Application** for AMI in the Internationalization Type field.

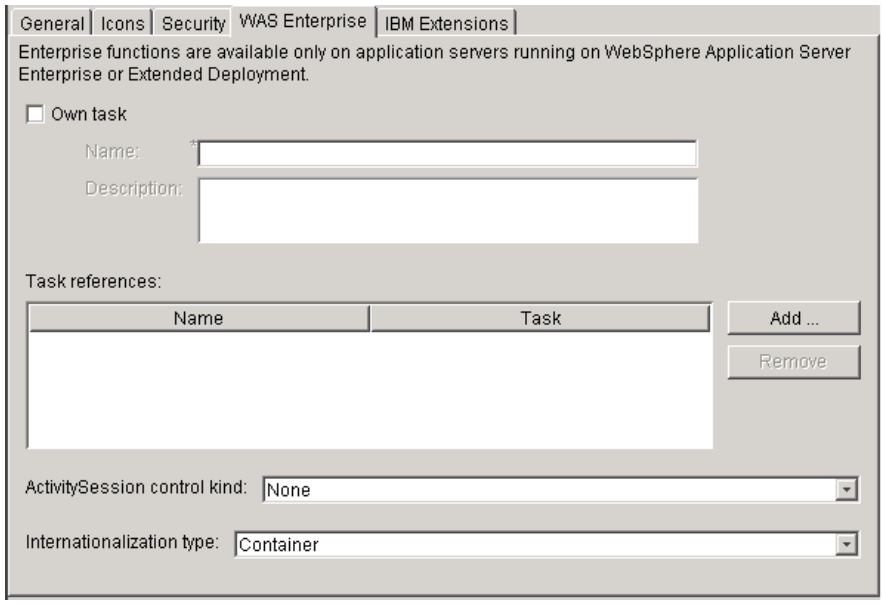


Figure 16-3 Setting the internationalization type for servlet

6. Click **Apply**.

16.5.2 Specify the container internationalization attribute

You can specify the container internationalization attribute for the CMI servlets and CMI EJBs. The applicable value of internationalization attribute is Caller, Server or Specified. If the value is Specified, the custom internationalization attributes can be specified.

Named container internationalization attributes can be associated with sets of servlets or with sets of EJB business methods.

Setting the internationalization attribute for EJBs

The steps to set the internationalization attribute for EJBs are the following:

1. Start the Application Assembly Tool.
2. Open your J2EE archive file.
3. Select the application name and click **EJB Modules.** Select the EJB modules name and click **Internationalization** in the left-hand pane.
4. To configure a new attribute, right-click **Internationalization** and select **New**. Otherwise, skip to step 5.

a. On the New Container Internationalization Attribute window, enter a description that uniquely identifies the policy in the Description field.

b. Click **Add by Methods**. On the All Methods window, select one or more methods to which the attribute will apply and click **OK** to exit the window. The selected methods will appear in the Methods list.

c. Click **OK**.

5. Select a named container internationalization attribute from the Description list. The fields of the selected attribute are displayed.
6. If desired, re-enter a description that uniquely identifies this attribute.
7. Click **Add by Methods**. On the All Methods window, select one or more methods to which the attribute will apply and click **OK** to exit the window. The selected methods will appear in the Methods list.
8. Select **Caller**, **Server** or **Specified** in the Run-as field.

   If you select **Specified**, follow the steps below:

   a. Enter a description of the specified context. This field is optional.

   b. Complete the Timezone field:

      i. Enter a description of this time zone.

      ii. Enter the ID of this time zone. For more detais on the time zone ID, see “Internationalization attribute” on page 589.

   c. Complete the Locales field:

      i. Click **Add**.

      ii. On the Add Locales window, enter a description, the language code, the country code and any variant. For details on locale, see “Internationalization attribute” on page 589.

      iii. Click **OK** to exit the window.

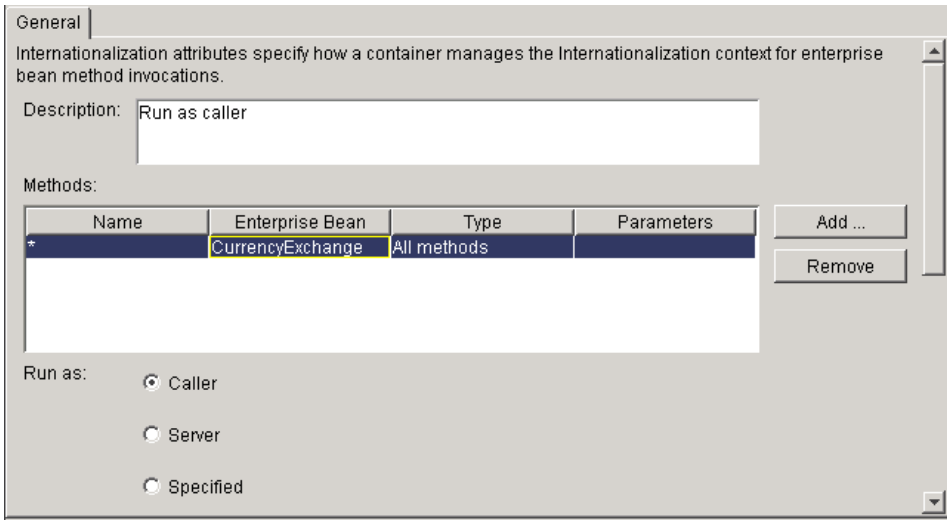


Figure 16-4 Setting the internationalization attribute for EJB

9. Click **Apply**.

Setting the internationalization attribute for servlets

The steps to set the internationalization attribute for servlets are the following:

1. Start the Application Assembly Tool.
2. Open your J2EE archive file.
3. Select the application name and click **Web Modules.** Select the Web modules name and click **Internationalization** in the left-hand pane.
4. To configure a new attribute, right-click **Internationalization** and select **New**. Otherwise, skip to step 5.
   a. On the New Container Internationalization Attribute window, enter a description that uniquely identifies the policy in the Description field.
   b. Click **Add by Web Components**. On the Add Servlets window, select one or more servlets to which the attribute will apply and click **OK** to exit the window. The selected servlets will appear in the Web Components list.
   c. Click **OK**.
5. Select a named container internationalization attribute from the Description list. The fields of the selected attribute are displayed.
6. If desired, re-enter a description that uniquely identifies this attribute.

7. Click **Add by Web Components**. On the Add Servlets window, select one or more servlets to which the attribute will apply and click **OK** to exit the window. The selected servlets will appear in the Web Components list.
8. Select **Caller**, **Server** or **Specified** in the Run-as field.

   If you select **Specified**, follow the steps below:

   a. Enter a description of the specified context. This field is optional.

   b. Complete the Timezone field:

      i. Enter a description of this time zone.

      ii. Enter the ID of this time zone. For details on the time zone ID, see “Internationalization attribute” on page 589.

   c. Complete the Locales field:

      i. Click **Add**.

      ii. On the Add Locales window, enter a description, the language code, the country code and any variant. For details on the locale, see “Internationalization attribute” on page 589.

      iii. Click **OK** to exit the window.

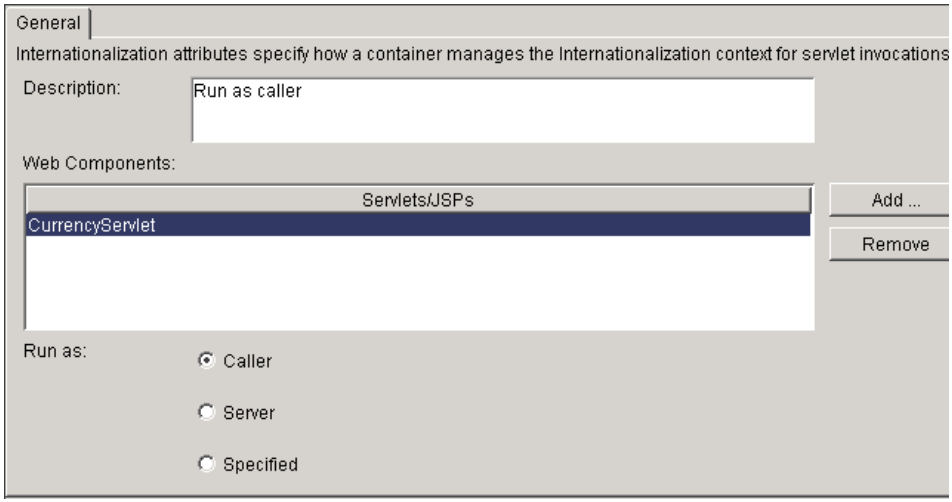


Figure 16-5 Setting the internationalization attribute for servlet

9. Click **Apply**.

16.6 Sample scenario for the EJB client

In the following text we demonstrate the use of i18n on our sample scenario. This sample shows how the Internationalization service works between EJBs and EJB clients.

16.6.1 Description

Briefly, we will schedule a task and it will be scheduled differently according to the time zone of the client that schedules a task. In this scenario we want to scheduler a task remotely.

Let's say that the task is an update of stock values and we want to update the values before stock trading begins.

Now, let's say this is a worldwide application and that we have a stock market in Frankfurt that opens at 9 AM local time and that we have a stock market in New York that opens at 8 AM local time. And, we want to use the same stand-alone client to schedule a task for updating the stock values, and we want to use the same commands or procedure to do it.

So, the internationalization context from the client will be propagated right to the scheduler proxy, which will decide which calendar to use for scheduling the task according to the client's time zone.

The following parts are involved:

- SchedulerClient

  This is a simple stand-alone client that runs in J2EE client container. We can use it for schedule tasks, list tasks, delete tasks and cancel tasks on a local or remote WebSphere server. As an argument, it also includes the time zone; thus we can simulate that a client is in a different time zone.

- SchedulerProxy

  SchedulerProxy is a stateless EJB that serves as a proxy between the scheduler client and Scheduler service.

  When scheduling a task it uses a user-defined calendar.

- ACompanyCalendar

  ACompanyCalendar is a user-defined scheduler calendar. What is important here is that its name identifier is the client time zone ID. So, we will use different calendars according to the client time zone ID.

16.6.2 Prerequisites

Before starting development, make sure that the following prerequisites are met:

- The scheduled task we use here is ACompanyTask handler, which was already developed for sample scenarios in Chapter 13, “Scheduler service” on page 509. It also uses some Java proxy classes for the Catalog process. For this sample scenario, it is really not necessary to have a “real task” defined. Therefore, if you skipped the scheduler sample scenario, you can just create a simple TaskHandler EJB with an empty process() method and use it as a task handler when defining and scheduling a task.

  Note: For the i18n demonstration, it is not necessary to define the ACompanyTask handler. You can use any task handler bean just as a placeholder.

  However, if you want to use our ACompanyTask, make sure that you successfully use the scheduler sample scenario together with all its prerequisites.
- For testing our sample with WebSphere Studio IE, there has to be an instance of WebSphere V5.0 Enterprise server created. For details refer to “Configuring the test environment” on page 670.
- The i18nctx.jar file has to be added to the Java build path of the EJB module that will contain the Startup Bean. This will be the ACompanyEJB module. If you are using WebSphere Studio IE, select this module, then right-click **Properties**, and go to the Java Build Path tab. Then choose **AddVariable**, select **WAS_EE_V5**, go to the lib directory, and choose **startupbean.jar**. Apply the changes.
- This sample requires the scheduler calendar services. The test server has to run the SchedulerCalendar.ear application. For more information, refer to 16.4, “Unit test environment” on page 598.

16.6.3 Develop

In this section, we will create and add the code for the sample scenario.

Create application client project

First, you have to create an application client project, as follows:

1. In WebSphere Studio IE, select **File** -> **New** -> **Application Client Project**. Then select **Create J2EE 1.3 Application client project**. Enter `ACompanyClient` as the project name. In the Enterprise application project field, select **Existing** and select **ACompany** for the existing project, and click

Next. On the next window, check **ACompanyEJB.jar** in the Available dependent JARs field. Click **Finish**. The new application client project is created.

2. In the J2EE Navigator view, right-click the newly created **ACompanyClient** project. Select **Properties** and you will get the Properties for ObjectPtest window. Go to the Java Build Path and select the **Libraries** tab.
3. In the Libraries tab, choose **AddVariable**, select **WAS_EE_V5**, double-click it, go to lib directory, and choose **i18nctx.jar**. Apply the changes.

Create SchedulerClient application client

To add the main class into application client project, do the following:

1. Still using the J2EE navigator view, in ObjectPtest project, select **appClientModule**, right-click it and select **New** -> **Class**. In the New Java Class window enter `SchedulerClient` as the value for the Name property. Also, enter `com.acompany.client` as the value for the Package property. Leave all the other properties at their default values. Click **Finish**.
2. Now, copy the code shown in Figure 16-7 into the SchedulerClient.java class.

Example 16-7 SchedulerClient.java

```
package com.acompany.client;

import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import com.acompany.ejbs.SchedulerProxy;
import com.acompany.ejbs.SchedulerProxyHome;
import com.ibm.websphere.i18n.context.InvocationInternationalization;
import com.ibm.websphere.i18n.context.UserInternationalization;

public class SchedulerClient {

   public static void main(String[] args) {

      SchedulerClient schedulerClient = new SchedulerClient();

      switch (args.length)
      {
      case 3:
      case 4:
         schedulerClient.callTask(Integer.parseInt(args[0]), args[1], args[2], args[3]);
         break;
      case 2:
         if (args[0].equalsIgnoreCase("cancelTask"))
         {
            schedulerClient.cancelTask(args[1], false);
            break;
```

```
        }
        else if (args[0].equalsIgnoreCase("deleteTask"))
        {
           schedulerClient.cancelTask(args[1], true);
           break;
        }
        if (args[0].equalsIgnoreCase("listTask"))
        {
           schedulerClient.listTask(args[1]);
           break;
        }
     default:
        System.out.println("SchedulerClient repeat taskName deltaInterval [timezoneID]");
        System.out.println("SchedulerClient cancelTask taskId");
        System.out.println("SchedulerClient deleteTask taskId");
        System.out.println("SchedulerClient listTask taskId");
        System.exit(-1);
     }
     System.exit(0);
  }

  public void callTask(

     int repeat,
     String taskName,
     String deltaInterval,
     String timeZoneId) {
     try {
        InitialContext ctx = new InitialContext();
        SchedulerProxy scheduler;
        SchedulerProxyHome home =
           (SchedulerProxyHome) PortableRemoteObject.narrow(
              ctx.lookup("java:comp/env/SchedulerProxy"),SchedulerProxyHome.class);
        scheduler = home.create();

        if (timeZoneId != null) {
           UserInternationalization i18nService =
              (UserInternationalization) ctx.lookup(
                 "java:comp/websphere/UserInternationalization");
           InvocationInternationalization invI18n =
              i18nService.getInvocationInternationalization();
           invI18n.setTimeZone(timeZoneId);
        }
        scheduler.scheduleTask(repeat, taskName, deltaInterval);
     } catch (Exception ex) {
        ex.printStackTrace();
     }
  }
```

```
    public void cancelTask(

       String taskId,
       boolean purge){
       try {
          InitialContext ctx = new InitialContext();
          SchedulerProxy scheduler;
          SchedulerProxyHome home =
             (SchedulerProxyHome) PortableRemoteObject.narrow(
                ctx.lookup("java:comp/env/ejb/SchedulerProxyHome"),
                SchedulerProxyHome.class);
          scheduler = home.create();

          scheduler.cancelTask(taskId, purge);
       } catch (Exception ex) {
          ex.printStackTrace();
       }
    }

       public void listTask(

       String taskId)
       {
       try {
          InitialContext ctx = new InitialContext();
          SchedulerProxy scheduler;
          SchedulerProxyHome home =
             (SchedulerProxyHome) PortableRemoteObject.narrow(
                ctx.lookup("java:comp/env/ejb/SchedulerProxyHome"),
                SchedulerProxyHome.class);
          scheduler = home.create();

          scheduler.listTask(taskId);
       } catch (Exception ex) {
          ex.printStackTrace();
       }
    }
}
```

In the code, you can see that in our stand-alone Java client we have four main methods: scheduleTask(), cancelTask(), deleteTask(), listTask(). What the client does is to find the SchedulerProxy EJB and then calls its remote method passing the right arguments.

Regarding i18n service, everything is done in the callTask() method. First we get the service via a JNDI lookup and then we get the invocation internationalization context. Then we set its time zone to the time zone specified by command-line arguments. Thus by setting the client time zone, we simulate different time zones

in which a client might be located. In a real-world application, you certainly won't do that since you usually want to set it to the system default.

Add a resource environment reference

Since a local reference to the SchedulerProxy EJB is used in SchedulerClient.java, you must add a resource environment reference, as follows:

1. Open the Client Deployment Descriptor of the ACompanyClient project. Go to References tab and click **Add**.
2. Select **EJB reference** and click **Next**. Now, enter `SchedulerProxy` in the Name field. Then on the Link: click **Browse** and in the Link reference selection window, select the **SchedulerProxy** bean under the ACompany project, and click **OK**. Click **Finish**.
3. Select the created resource environment reference and in WebSphere Bindings in the JDNI name field, make sure that ejb/com/acompany/ejbs/SchedulerProxyHome is entered.

Create SchedulerProxy bean

In the following section, we describe how to create SchedulerProxy bean and the necessary environment.

Create the bean

To create the bean, do as follows:

1. In the J2EE hierarchy window, go to EJB Modules and right-click the **ACompanyEJB** module. Select **New** -> **Enterprise Bean**.
2. In the first Enterprise Bean Creation window, make sure that **ACompanyEJB** is selected for the project, and click **Next**.
3. Configure the first Enterprise Bean Creation window as shown in Figure 16-6 on page 611. Click **Next**.

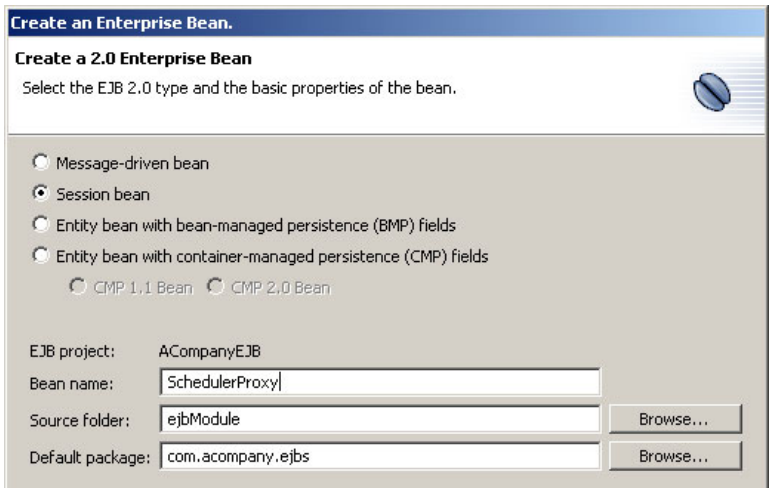


Figure 16-6 Creating SchedulerProxy bean, first window

4. Configure the second Enterprise Bean Creation window as shown in Figure 16-7. Click **Finish**.

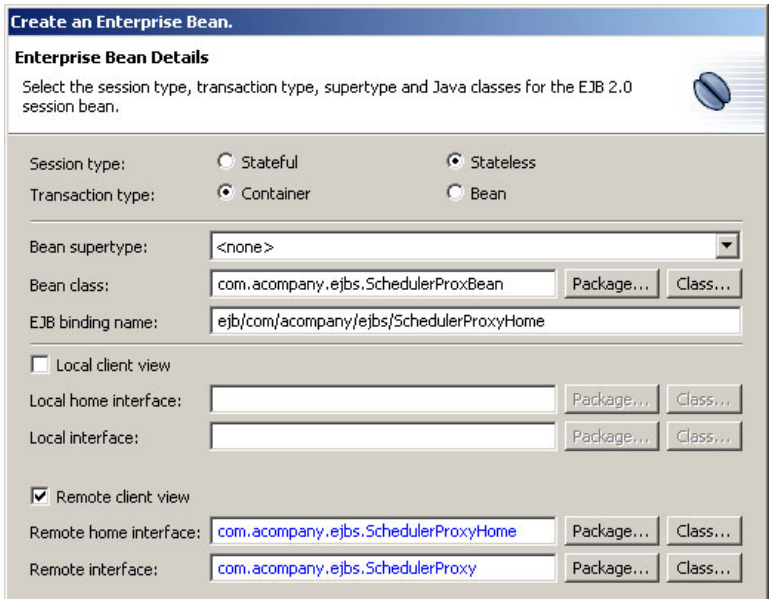


Figure 16-7 Creating SchedulerProxy bean, second window

Add resource environment reference

Since we use local EJB references for the ACompanyCalendar bean, for the ACompanyTask bean and for the Scheduler service in the Scheduler proxy bean, you have to add resource references to the Deployment Descriptor of EJB module, as follows:

1. Open the Deployment Descriptor of the ACompanyEJB project by double-clicking **ACompanyEJB** in the EJB modules directory using the J2EE Hierarchy view. Go to the References tab, select the **SchedulerProxy** bean and click **Add**.
2. Select **EJB reference** and click **Next**. Now, enter `LocalCalendarReference` in the Name field. Then on the Link: click **Browse** and in the Link reference selection window, select the **ACompanyCalendar** bean under the ACompany project, and click **OK**. Click **Finish**.
3. Select the created resource environment reference and in WebSphere Bindings in the JNDI name field, make sure that `ejb/com/acompany/ejbs/ACompanyCalendarHome` is entered.

> **Note:** If you use your own task handler bean, you have to modify steps 4 and 5 accordingly.

4. Again select the **SchedulerProxy** bean and click **Add.** Select **EJB reference** and click **Next**. Now, enter `LocalTaskHandlerReference` in the Name field. Then on the Link: click **Browse** and in the Link reference selection window, select the **ACompanyTask** bean under the ACompany project, and click **OK**. Click **Finish**.
5. Select the created resource environment reference and in WebSphere Bindings in the JDNI name field, make sure that `ejb/com/acompany/ejbs/ACompanyTaskHome` is entered.
6. Again select the **SchedulerProxy** bean and click **Add**. Select **Resource environment reference** and click **Next**. Now, enter `ACompanyScheduler` in the Name field, and enter `com.ibm.websphere.scheduler.Scheduler` in the Type field. Click **Finish**.
7. Select the created resource environment reference and in WebSphere Bindings in the JNDI name field, enter `ACompanyScheduler` for the value. Save the changes.

Adding code for the SchedulerProxy remote interface

Open the SchedulerProxy.java file and copy the code shown in Example 16-8 on page 613.

Example 16-8 SchedulerProxy.java

```
package com.acompany.ejbs;
/**
 * Remote interface for Enterprise Bean: SchedulerProxy
 */
public interface SchedulerProxy extends javax.ejb.EJBObject
{

   public boolean scheduleTask(int repeats, String taskName, String deltaInterval) throws
java.rmi.RemoteException;
   public void cancelTask(String taskId, boolean purge) throws java.rmi.RemoteException;
   public void listTask(String taskId) throws java.rmi.RemoteException;
}
```

Adding code for the SchedulerProxy bean

Open theSchedulerProxyBean.java and copy the code shown in Example 16-9.

Example 16-9 SchedulerProxyBean.java

```
package com.acompany.ejbs;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import com.ibm.websphere.i18n.context.Internationalization;
import com.ibm.websphere.i18n.context.UserInternationalization;
import com.ibm.websphere.scheduler.BeanTaskInfo;
import com.ibm.websphere.scheduler.Scheduler;
import com.ibm.websphere.scheduler.TaskHandlerHome;
import com.ibm.websphere.scheduler.TaskStatus;

public class SchedulerProxyBean implements javax.ejb.SessionBean {
   private javax.ejb.SessionContext mySessionCtx;
   /**
    * getSessionContext
    */

   private Scheduler scheduler = null;
   private InitialContext initialContext = null;
   private UserInternationalization i18nService = null;

   public javax.ejb.SessionContext getSessionContext() {
      return mySessionCtx;
   }
   /**
    * setSessionContext
    */
   public void setSessionContext(javax.ejb.SessionContext ctx) {
      mySessionCtx = ctx;
   }
```

```
   /**
    * ejbCreate
    */
   public void ejbCreate() throws javax.ejb.CreateException {
      try {
         initialContext = new InitialContext();
         System.out.println("SchedulerProxyBean.scheduleTask: Getting the Scheduler");
         scheduler = (Scheduler) initialContext.lookup("java:comp/env/ACompanyScheduler");
         //Get the i18Service
         i18nService = (UserInternationalization)
initialContext.lookup("java:comp/websphere/UserInternationalization");
      } catch (javax.naming.NamingException e) {
         System.out.println("SchedulerProxyBean: lookup failed");
         e.printStackTrace();
      }

   }
   /**
    * ejbActivate
    */
   public void ejbActivate() {
   }
   /**
    * ejbPassivate
    */
   public void ejbPassivate() {
   }
   /**
    * ejbRemove
    */
   public void ejbRemove() {
   }

   public void cancelTask(String taskId, boolean purge) {
      try {
         scheduler.cancel(taskId, purge);
      } catch (Exception e) {
         System.out.println("SchedulerProxyBean: cancelTask failed");
         e.printStackTrace();
      }
   };

   public void listTask(String taskId) {
      TaskStatus status;
      try {
         status = scheduler.getStatus(taskId);
         System.out.println("SchedulerProxyBean.listTask: Task ID is: " + status.getTaskId());
         System.out.println("SchedulerProxyBean.listTask: The name of the Task is: " +
status.getName());
```

```
         System.out.println("SchedulerProxyBean.listTask: Task created " +
status.getTimeCreated());
         System.out.println("SchedulerProxyBean.listTask: Task status is: " +
status.getStatus());
         System.out.println("SchedulerProxyBean.listTask: Next fire time " +
status.getNextFireTime());

      } catch (Exception e) {
         System.out.println("SchedulerProxyBean: listTask failed");
         e.printStackTrace();
      }
   };

   public boolean scheduleTask(int repeats, String taskName, String deltaInterval) {
      return (scheduleTaskInt(0, repeats, 0, taskName, deltaInterval));
   };

   private boolean scheduleTaskInt(long startInterval, int repeats, long repeatInterval,
String taskName, String deltaInterval) {
      boolean result = false;
      //UserInternationalization i18nService = null;

      try {
         //Get Time zone ID - ISO 3166
         Internationalization clientI18n = i18nService.getCallerInternationalization();
         String clientTimeZoneID = clientI18n.getTimeZone().getID();

         // Get the task handler which will be called when the task runs.
         System.out.println("SchedulerProxyBean.scheduleTask: Getting the task handler");

         Object taskHandlerObj =
initialContext.lookup("java:comp/env/LocalTaskHandlerReference");
         TaskHandlerHome taskHandlerHome = (TaskHandlerHome)
PortableRemoteObject.narrow(taskHandlerObj, TaskHandlerHome.class);

         // Create our Schedule Task Info for our ACompanyTask task handler
         System.out.println("SchedulerProxyBean.scheduleTask: Creating the task");

         BeanTaskInfo taskInfo = scheduler.createBeanTaskInfo();
         taskInfo.setTaskHandler(taskHandlerHome);
         taskInfo.setStartTimeInterval(startInterval + "minutes");

         if (deltaInterval.length() != 0)
            taskInfo.setRepeatInterval(deltaInterval);
         else
            taskInfo.setRepeatInterval(repeatInterval + "minutes");

         taskInfo.setNumberOfRepeats(repeats);
         taskInfo.setName(taskName);
```

```
         //set calendar bean and specify calendar identifer according to client's time zone
id, if not CET the default calendar will be used
         System.out.println("SchedulerProxyBean.scheduleTask: clientTimeZone =" +
clientTimeZoneID);

         taskInfo.setUserCalendar("java:comp/env/LocalCalendarReference", clientTimeZoneID);

         //set the name of the task which is associated to a TaskID
         TaskStatus status = scheduler.create(taskInfo);

         System.out.println("SchedulerProxyBean.scheduleTask: Task submitted");
         System.out.println("SchedulerProxyBean.scheduleTask: Task ID is: " +
status.getTaskId());
         System.out.println("SchedulerProxyBean.scheduleTask: The name of the Task is: " +
status.getName());
         System.out.println("SchedulerProxyBean.scheduleTask: Task status is: " +
status.getStatus());
         result = true;

      } catch (Exception e) {
         System.out.println("SchedulerProxyBean.scheduleTask: failed");
         e.printStackTrace();
      }
      return (result);
   }
}
```

As you can see in the code on SchedulerProxy's creation in ejbCreate() method, we do a lookup for the Scheduler service and the i18n service.

The main method is schedulerTaskInt(). Here we get the client's internationalization context and then specify the calendar that will be used with the scheduled task according to the client internationalization context. In our case, our calendar name is "Europe/Brussels" . Since the name is also used to identify time zone in the internationalization context, we just pass the client's time zone ID to our user-defined calendar bean as the calendar name.

We then schedule the task. If you are more interested in scheduling procedures, they are the same as used with Scheduler sample scenario. See "Sample scenario" on page 517.

When the Scheduler service schedules a task, it calls applyDelta() method of the calendar bean specified for the task. In our case, we used our calendar bean. It then determines how to schedule the task according to the client time zone. For details about the calendar, see "User-defined calendar" on page 522.

Use the application client

Finally you can build the project and test the client. We will export the project into the EAR file and use the **launchClient** command to run the project inthe J2EE client container. Follow these steps:

1. Select the **ACompanyClient** project, and from the menu bar select **Project -> Rebuild Project**. Click **Export**, select the EAR file and click **Next**. Select **ACompany** as a project you want to export. Enter the file name to which you want to export the project. We entered `C:\ACompany.ear`. Click **Finish**, and check if the project was exported to where you specified.
2. In Windows, open a Command Prompt window. If you don't have launchClient.bat in your system path, move to the location where it is. You can use the one that comes with WebSphere Studio IE test environment. In this case it is in the <*WebSphere_Studio_root*>\runtimes\ee_v5\bin directory. Or if you have WebSphere V5.0 Enterprise installed, you can use its launchClient. In that case, it is in the <*WebSphere_root*>\bin directory.
3. Make sure that the WebSphere Application test server is running.
4. SchedulerClient recognizes one of the following syntax forms:
   - SchedulerClient repeat taskName deltaInterval timezoneID

     This form is used for scheduling task. It takes the following arguments:

     repeat — How many times the scheduled task will repeat.

     deltaInterval — The time interval when the task will be scheduled. For the this demonstration of the i18n service, you have to use the beginTrade value as the deltaInterval.

     timezoneID — The time zone ID under which the client will "pretend" to be running. For the demonstration of the i18n service, you can use "Europe/Brussels" as the value of timezoneID.
   - SchedulerClient cancelTask taskId

     This form is used for canceling tasks. You simply specify the ID of the task you want to cancel.
   - SchedulerClient deleteTask taskId

     This form is used for deleting tasks. You simply specify the ID of the task you want to delete.
   - SchedulerClient listTask taskId

     This form is used for listing task information. You simply specify the ID of the task you want to see status.
5. Here is an example how to use our client:

```
launchClient C:\ACompany.ear -CCjar=ACompanyClient.jar
-CCclasspath="C:\program files\websphere\appserver\lib\i18nctx.jar"
```

```
-CCBootstrapHost=localhost -CCBootstrapPort=2809 -CCverbose=true
SchedulerClient 1 ATaskName beginTrade Europe\Brussels
```

where `BootstrapPort` and `BootstrapHost` are the bootstrap port and the host name or IP of the WebSphere Application Server on which you will run the client. In our case, this is WebSphere Studio IE test environment server. Check the bootstrap port in your configuration.

This command will schedule a task. Give it a name (ATaskName), and specify 1 repetition. It will also set the Europe\Brussels calendar to “begin trade”, which means it will schedule the update task to happen Mon-Fri at 8AM, the beginning of the trading day.

Another example would be:

```
launchClient C:\ACompany.ear -CCjar=ACompanyClient.jar
-CCclasspath="C:\program files\websphere\appserver\lib\i18nctx.jar"
-CCBootstrapHost=localhost -CCBootstrapPort=2809 -CCverbose=true
SchedulerClient 1 ATaskName beginTrade Pacific/Honolulu
```

In this case a different calendar will take effect and as a consequence, the task will be scheduled to happen Mon-Fri at 9AM.

16.7 Sample scenario for the Web client

In this section, we describe how the Internationalization service works in the sample scenarios. This sample shows you how the Internationalization service works for the Web clients, such as Web browsers.

Users prefer user interfaces that are localized for their language, and many Web sites provide the localized user interfaces. The preferred languages are usually a part of the configuration of the Web browser. In this section, we show you how to provide the localized Web UI by using the Internationalization service. To access to the localized UI, follow these steps:

1. Invoke a Web browser and open the following page:

   ```
   http://<your server name>:<your server port>/approval/index.jsp
   ```

   You will see the UI where the local language is English.

Figure 16-8 The default UI

2. Add Korean and Japanese to the language setting of the Web browser. For example, click **Tools -> Internet Options -> Languages** if you are using Microsoft® Internet Explorer.
3. Access the same page again. This time, you will see the UI where the language is Japanese.

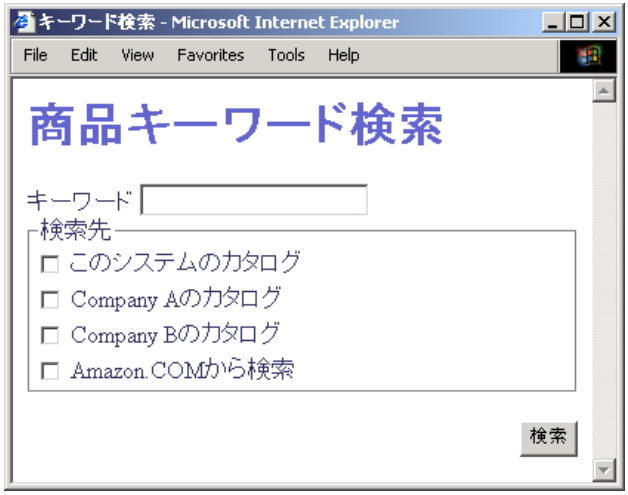


Figure 16-9 The Japanese UI

In this application, the default UI is in English. If no languages are specified, the default English UI appears. In addition, a Japanese UI is provided, but the

Korean UI is not. Therefore, when the user specifies Korean and Japanese as the preferred languages, the Japanese UI will be displayed even if the priority of the Korean language is higher than Japanese.

16.7.1 Implementation details

When the index.jsp is accessed, this JSP looks for the localized page by using I18nUtils and forwards the request to the page.

The Java source codes, JSPs, and the Deployment Descriptors are in the ACompanyApprovalWeb project. To compile this project, the i18nctx.jar has to be in your classpath, and asyncbeans.jar and wsexception.jar also have to be in your classpath, because this project has a dependency on Asynchronous Beans.

I18nUtils

When the instance of I18nUtils is created, UserInternationalization is looked up from the JNDI repository and kept for future use.

In the getLocalizedFileName method, the user's preferred languages are retrieved as the caller locales from the internationalization context. Then it determines the file name from the specified base name, specified extension and one of the locales, and checks if the file exists. For example, when the base name is "/query", the extension is ".html" and the locales are Korean(ko) and Japanese(ja), it looks for "/query_ko.html" and "/query_ja.html" as the localized files. If the file exists, it returns the file name. If no files exist, it returns the default file name, which is the base name plus the extension.

Example 16-10 I18nUtils

```
package com.acompany.approval;

import java.io.File;
import java.util.Locale;
import javax.naming.*;
import javax.servlet.ServletContext;
import com.ibm.websphere.*;

public class I18nUtils {
   private UserInternationalization i18nService = null;
   ......
   public I18nUtils() {

      try {
         InitialContext ctx = new InitialContext();
         i18nService =
            (UserInternationalization) ctx.lookup(
               "java:comp/websphere/UserInternationalization");
```

```
            } catch (NamingException ne) {
                ne.printStackTrace();
            }
        }

        public String getLocalizedFileName(
            String baseName,
            String extension,
            ServletContext context) {
            String resultFile = null;
            Internationalization callerI18n =
                i18nService.getCallerInternationalization();
            Locale[] locales = callerI18n.getLocales();

            for (int i = 0; i < locales.length; i++) {
                String fileName = baseName;
                String language = locales[i].getLanguage();
                if (language != null && language.length() != 0) {
                    fileName = fileName + "_" + language;
                    String country = locales[i].getCountry();
                    if (country != null && country.length() != 0) {
                        fileName = fileName + "_" + country;
                    }
                }
                fileName = fileName + extension;
                String path = context.getRealPath(fileName);
                File file = new File(path);
                if (file.exists()) {
                    return fileName;
                }
            }
            return baseName + extension;
        }
    }
```

16.7.2 Configuration and requirements

To run this application, you need to have an installation of the WebSphere Enterprise Version 5 configured with the Additional Enterprise Extensions option. After you install WebSphere Enterprise Version 5, you must configure it to start up the Internationalization service.

16.8 Configuration

The default setting of the Internationalization service is disabled. If you are planning to use this service, you must enable it. To enable the Internationalization service, do the following:

1. Start the WebSphere Administrative Console, then log on.
2. Click **Servers -> Application Servers**.
3. Select the server name.
4. Click **Internationalization Service** in the Additional Properties.
5. Check **Startup** and click **Apply**.

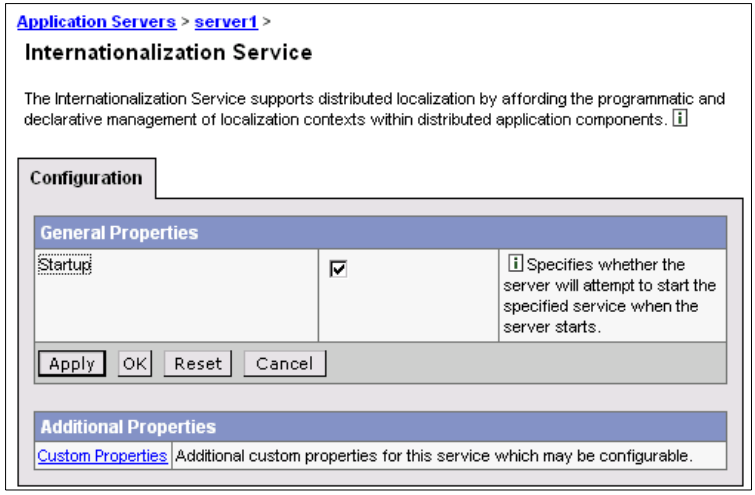


Figure 16-10 Enable the Internationalization service on Administrative Console

6. Save the configuration and restart the server.

You can also enable the Internationalization service by using the **wsadmin** command. Enter the following commands to wsadmin and restart the server:

Example 16-11 Enable the Internationalization service by wsadmin

```
set x [$AdminConfig list I18NService]
$AdminConfig modify $x { { enable true } }
$AdminConfig save
exit
```

16.9 Deployment

There are no settings at deployment time. If you need to set the internationalization type and attributes, use the Application Assembly Tool. For more details, see “Assembly” on page 599.

16.10 Problem determination and troubleshooting

This section shows you the problem determination steps. First, we show the possible reasons for some exceptions and the solutions:

- javax.naming.NameNotFoundException

  If the Internationalization service does not properly initialize, the JNDI lookup on the UserInternationalization URL throws a javax.naming.NameNotFoundException. Make sure the Internationalization service is installed correctly and configured to start up. To configure to start up the Internationalization service, see “Configuration” on page 622.

  When the Internationalization service initialize properly, you will see the following message in SystemOut.log in <*WebSphere_root*>/logs/<server name> directory.

Example 16-12 The Internationalization service initializing message

```
[4/3/03 16:36:01:997 CST] 6daa4389 I18nService   I I18N0009I: The
Internationalization service is created on server1.
[4/3/03 16:36:01:997 CST] 6daa4389 I18nServiceSe I I18N0009I: The
Internationalization service is enabled on server1.
[4/3/03 16:36:03:429 CST] 6daa4389 I18nService   I I18N0009I: The
Internationalization service is initialized on server1.
```

- java.lang.IllegalStateException

  java.lang.IllegalStateException is thrown whenever a server application component whose internationalization type is set to container-managed internationalization (CMI) attempts to set the invocation context. This is a violation of the CMI policy, under which servlets and EJBs cannot modify their invocation internationalization context.

  If you need to modify the invocation internationalization context, modify the internationalization type to application-managed internationalization (AMI) by using the Application Assembly Tool. For more details, see “Assembly” on page 599.

If you still have a problem, turn on the trace service to get more information about the Internationalization service. To have your application server emit the trace statements for the Internationalization service, specify the appropriate trace

string to the server's diagnostic trace service. The steps for this task are as follows:

1. Start the WebSphere Administrative Console, then log on.
2. Click **Servers -> Application Servers**.
3. Select the server name.
4. Click **Diagnostic Trace Service** in the Additional Properties.
5. Select the **Enable Trace** check box.
6. In the Trace Specification field, type the following (no spaces and no line breaks):

   ```
   com.ibm.ws.i18n.context.*=all=enabled:com.ibm.websphere.i18n.context.*=all=enabled
   ```

7. Click **Apply**.
8. Save the configuration and restart the server.

16.11 Install Enhanced Internationalization Service Technology Preview

If you are planning to use the Enhanced Internationalization Service, you must install the Enhanced Internationalization Service Technology Preview. This technology preview requires the installation of WebSphere Enterprise Version 5 configured with the Additional Enterprise Extensions option and the installation of Web Services Technology Preview. For more details about the installation of WebSphere Enterprise Version 5, see Appendix A, "Installation and configuration" on page 653. For more details about the Web Services Technology Preview, refer to the following resources:

- Web Services Technology Preview

  http://www7b.software.ibm.com/wsdd/downloads/web_services.html

- *WebSphere Version 5 Web Services Handbook,* SG24-6891

The steps for installing the Enhanced Internationalization Service Technology Preview are as follows:

1. Download the Technology Preview from:

   http://www7b.software.ibm.com/wsdd/downloads/wasee5/ei.html

2. Unzip the downloaded file into a temporary directory.
3. Change the directory to the i18nctx subdirectory of the temporary directory.

4. Launch the installer. You must specify the WebSphere Enterprise Version 5 installed directory as the argument.

   ```
   installI18nTP C:/WebSphere/AppServer
   ```

5. Click **Next** on the Welcome window.
6. Click **OK** to accept the license agreement.
7. Verify the WebSphere Enterprise Version 5 installed directory, which is shown in the Directory Name field, and click **Next** to continue.
8. Click **Next** to install the features shown.
9. Click **Finish** to complete the installation.

17

WebSphere Enterprise runtime

This chapter discusses the WebSphere runtime environment, including generic functions and features, some architecture considerations and configuration issues.

Most of the runtime environment discussions apply to the base application server, which is covered in the following Redbooks:

- *IBM WebSphere Application Server V5.0 System Management and Configuration*, SG24-6195
- *IBM WebSphere V5.0 Performance, Scalability and High Availability,* SG24-6198

17.1 Introduction

WebSphere Application Server Version 5.0 is IBM's implementation of the J2EE (Java 2 Enterprise Edition) platform, conforming to the Java 2 Platform Enterprise Edition (J2EE) Specification, V1.3.

In this chapter, we discuss the runtime architecture and components of three different WebSphere Application Server packages:

- IBM WebSphere Application Server V5, referred to as WebSphere base.
- IBM WebSphere Application Server Enterprise V5, referred to as WebSphere Enterprise.
- IBM WebSphere Application Server Network Deployment V5, referred to as Deployment Manager.

  A Network Deployment environment consists of multiple application server nodes grouped into a single administrative domain.

There are also other WebSphere Application Server packages, but they are not relevant here. If you want more information about different packages, see the redbook *IBM WebSphere Application Server V5.0 System Management and Configuration,* SG24-6195.

17.2 Architecture

This chapter will focus on architectural differences introduced by WebSphere Enterprise. In addition we briefly discuss each of the WebSphere Enterprise Extensions.

From the runtime architectural point of view, we discuss two different WebSphere versions: base and Enterprise.

WebSphere Enterprise provide additional support for solving “large-scale” software development problems in the areas of:

- Function (sometimes called Extensions): New APIs providing increased power for the J2EE programmer
- Quality of Service: Transparently making applications more scalable, manageable and higher performing (sometimes called Optimizations).
- Extensibility: Plug-in and SPIs required for large, complex heterogeneous environments.

17.2.1 WebSphere Application Server V5 base

Figure 17-1 illustrates the basic architectural layout of WebSphere Application Server V5 base.

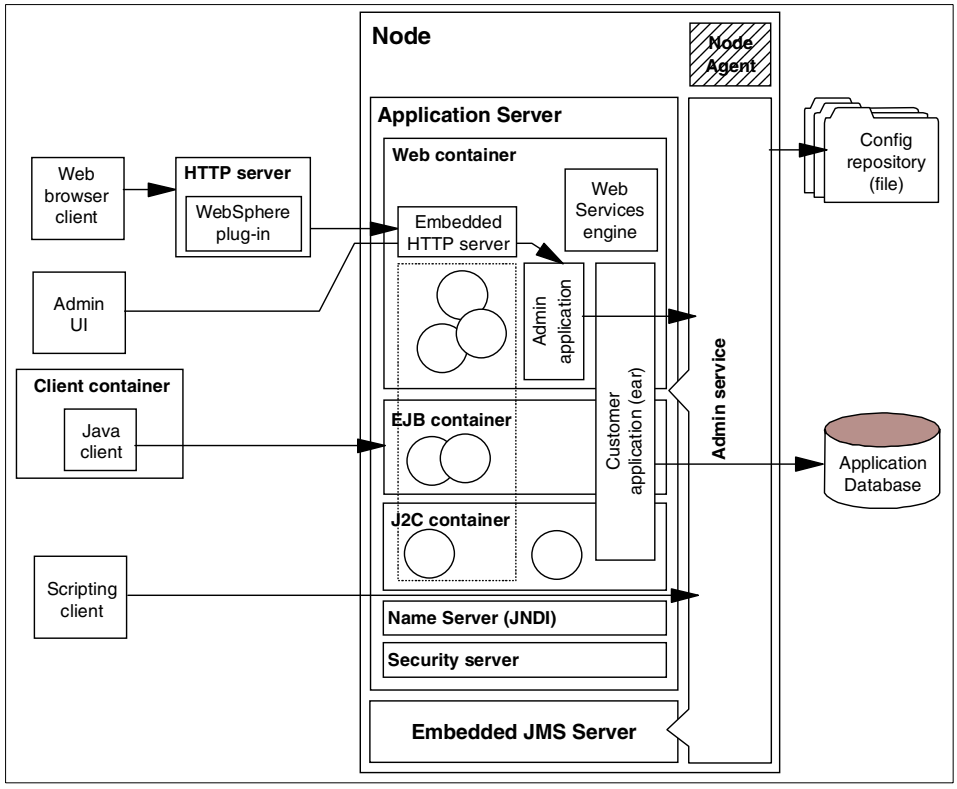


Figure 17-1 WebSphere Application Server V5 base architecture

In the following sections, we briefly describe the parts of the application server shown in the diagram.

Node

A node is a logical grouping of WebSphere managed server processes that share common configuration and operational control. In the base configuration, each application server is responsible for its own configuration in the configuration repository.

Node Agent

The Node Agent is a process running on each node connected to a cell. It is present in every WebSphere Application Server installation, but it is only activated when the server connects to the cell.

The process handles Deployment Manager and node communications. Its responsibilities include administration tasks and process management tasks.

HTTP Server with WebSphere plug-in

The HTTP Server is the Web server, the primary access point to the Web applications. A Web server can serve static content to the clients, and can route specific requests to application servers.

In order to pass an application server request, the Web server has a plug-in installed that routes the request to the right application server.

Embedded HTTP server

The application server also has to act as a Web server to serve content to clients using HTTP protocol. In this case the content can be either static or dynamic, although application servers practically serve dynamic content.

The embedded Web server is part of the Web container. Each application server has one Web container. Therefore each application server has an embedded server instance running and listening on a distinct port.

Application server

The application server is the primary component of WebSphere. It runs in a Java Virtual Machine (JVM), providing the runtime environment for the application's code. The application server provides containers that specialize in enabling the execution of specific Java application components. There are three containers in the application servers:

- Web container
- EJB container
- J2C container

Web browser clients connect to the Web container through the HTTP server and Web server plug-in to access the dynamic content. Stand-alone Java applications, either J2EE application clients or thin Java clients, connect to the EJB container to access EJBs, and invoke methods through RMI/IIOP.

Web container components can use EJB resources within the application logic. Application servers can access a shared database for storing data. The application server provides other services besides the containers:

- Object Request broker (ORB)
- Name service (JNDI)
- Security service (JAAS and Java 2 security)
- Admin service (JMX)
- Trace service

- Performance Monitoring Interface (PMI)
- Transaction management
- Messaging interfaces (JMS)
- E-mail interfaces (JavaMail)
- Database connection (JDBC) and connection pooling

Web container

The Web container processes servlets, JSP files and other types of server-side includes. Each Web container automatically contains a single session manager.

When handling servlets, the Web container creates a request object and a response object, then invokes the servlet service method. The Web container invokes the servlet's destroy method when appropriate and unloads the servlet, after which the JVM performs garbage collection.

The Web container runs an embedded HTTP server for handling HTTP(S) requests from external Web server plug-ins or Web browsers.

A Web container configuration provides information about the application server component that handles servlet requests forwarded by the Web server. Each application server runtime has one logical Web container, which can be modified but not created or removed. The administrator specifies Web container properties including:

- Default virtual host.
- Session management properties.
- Number and type of connections between Web server and Web container.
- Port(s) on which the Web container listens for incoming HTTP(S) requests.

EJB container

The EJB container provides all the runtime services needed to deploy and manage Enterprise Java Beans (EJBs). It is a server process that handles requests for both session and entity beans.

The enterprise beans (inside EJB modules) installed in an application server do not communicate directly with the server. Instead, the EJB container provides an interface between the EJBs and the server. Together, the container and the server provide the bean runtime environment.

The container provides many low-level services, including threading and transaction support. From an administrative viewpoint, the container manages data storage and retrieval for the contained beans. A single container can host more than one EJB JAR file.

J2C container

The Java Connector Architecture (JCA) container is a component provided by WebSphere Application Server, into which JCA resource adapters from EIS vendors can be plugged-in, configured and used by JCA compliant applications.

For information on JCA containers, see the J2EE Connector specification at:

http://java.sun.com/j2ee/connector/

Admin service

The Admin service runs within each server JVM. In the base configuration, the Admin service runs in the application server. In the Network Deployment configuration, each of the following servers host an Admin service:

- Deployment Manager
- Node Agent
- Application server
- JMS server

The Admin service provides the necessary functions to manipulate configuration data for the server and its components. The configuration is stored in a repository, a set of XML files stored in the server's file system.

Application servers are attached to nodes and nodes belong to a cell in the Network Deployment environment. In this environment the Deployment Manager is responsible for managing all the application servers in the cell, which means that the administrator has access to multiple application servers under one user interface through the Deployment Manager.

The Admin service running in a particular server is responsible only for that server.

Admin services has a course-grained security control and filtering functionality, providing different levels of administration to certain users or groups.

Admin application

The Admin application is a special EAR installed on the application server instance. This EAR only has a Web module. The application performs administration tasks for the application server, based on user interactions from the Web.

Admin UI

The Admin UI (user interface) is a Web browser-based interactive graphical user interface. Users can access the Admin application with a Web browser and perform administration tasks.

Scripting client (wsadmin)

The scripting client is a text-based, command-line administration client for WebSphere Application Servers. The client supports multiple scripting languages.

Embedded JMS Server

The embedded WebSphere JMS provider uses a JMS server to implement the integrated messaging functions. It supports point-to-point and publish/subscribe styles of messaging and is integrated with the transaction management service.

The JMS server is used for:

- Support of message-driven beans.
- Messaging within a WebSphere cell.

In the base configuration, the JMS server runs in the same JVM as the application server. In the Network Deployment configuration, the JMS server is separated from the application server and runs in a separate dedicated JVM.

Name server (JNDI)

Each application server JVM hosts a name service that provides a Java Naming and Directory Interface (JNDI) name space. The service is used to register all EJBs and J2EE resources (JMS, J2C, JDBC, URL, JavaMail) hosted by the application server.

The JNDI implementation in WebSphere Application Server V5 is built on top of a Common Object Request Broker Architecture (CORBA) naming service (CosNaming).

Security server

Each application server JVM hosts a security service that uses the security settings held in the configuration repository to provide authentication and authorization functionality.

Security issues are not covered in this redbook. For information about WebSphere security, see the redbook *IBM WebSphere V5.0 Security Handbook*, SG24-6573.

Web Services engine

The Web services engine does not really stand as a separate component. The application server implements numerous APIs for additional services. Web services are provided as a set of APIs in cooperation with the J2EE applications.

WebSphere's Web services engine is based on AXIS, and implements the following specifications:

- SOAP (Simple Object Access Protocol)

  A protocol that defines the messaging between Objects. It is based on the XML and XML Schema specification.

- WSDL (Web Services Description Language)

  Describes the services that can be located and used by applications.

- UDDI (Universal Description, Discovery and Integration)

  Enables an application to find services on the network published by service brokers.

- WSIF (Web Services Invocation Framework)

  A tool that provides a standard API for invoking services described in WSDL, no matter how or where the services are provided. The architecture allows new bindings to be added at runtime.

The Web services support provided in WebSphere Application Server V5 is not covered in this redbook. For detailed information on the Web services support in WebSphere Application Server V5, see the redbook *WebSphere Version 5 Web Services Handbook*, SG24-6891.

17.2.2 WebSphere Application Server Enterprise

Figure 17-2 on page 635 depicts the basic architectural layout of WebSphere Application Server Enterprise V5.

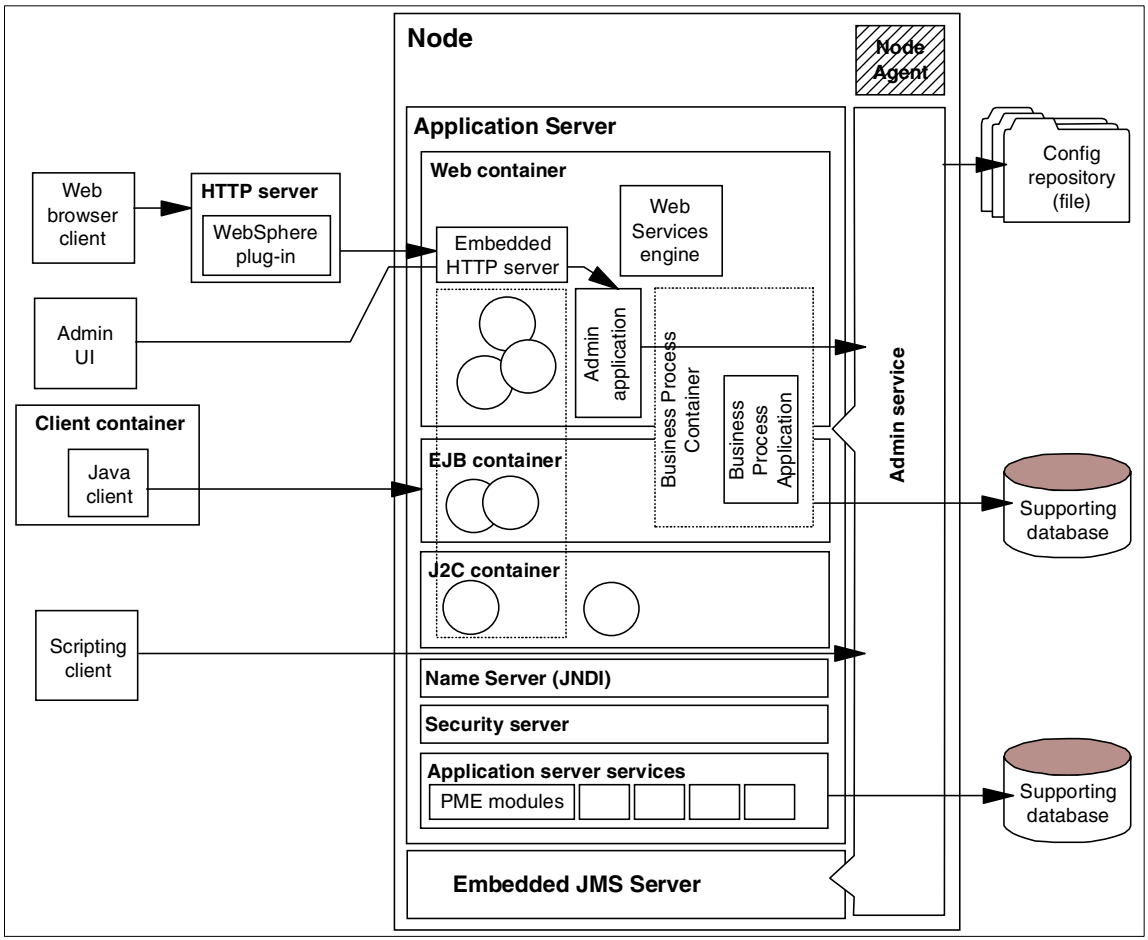


Figure 17-2 WebSphere Application Server Enterprise V5 architecture

This redbook does not go into detail about the components that exist in WebSphere Application Server base or Network Deployment. This section focuses on those components that are provided by the WebSphere Enterprise environment.

Business Process container

The Business Process Container is a new type of container for the application server. It is indicated as an enterprise application overlapping the Web and EJB container. The container is really a special enterprise application and not an application server implementation of a new container. Because the application server is still a J2EE application server, the container implementation does not

violate the specification. The administration process hides these details from the user, so it really looks and works like an individual container.

The Business Process applications, developed by WebSphere Studio IE, are also deployed as enterprise applications (.ear). The WebSphere Enterprise administration makes sure that the processes are deployed into the process container. The Business Process applications usually have three modules:

- Process EJB module, implementation of the process flow.
- Process Web client, implementation of the GUI for the process.
- Flow archive (.far), a package of all the flow descriptions implemented in the application. The flow descriptions are stored in .fdml format. This module is automatically generated by WebSphere Studio IE and stored under the root EAR module as a packaged file.

Special Enterprise Archive

As mentioned before, the Business Process Container is a special enterprise application:

- It is loaded first as an enterprise application.
- The class loader can see other enterprise application modules, because the container has to interact with the process applications.

The business process applications are also somewhat special. They are loaded before other enterprise applications.

Important: There is an important consideration regarding classloading the business process container and related applications. Consider the following scenario, based on our sample:

- ACompany's business logic is implemented as a set of EJBs under the ACompany EAR, packaged into the ACompanyEJB.jar.
- ACompany's business processes are under the ACompanyProcess EAR.

The following steps explain the problems we can experience:

1. The business processes have to use the business logic calling methods on the EJBs. When exporting ACompanyProcess from Studio, the EAR will include the ACompanyEJB.jar as part of the package.
2. Exporting ACompany from Studio will put the ACompanyEJB.jar in the .ear again. It is quite common that we have the same code packaged and deployed multiple places instead of just one, although it is not recommended and should be avoided.
3. After deploying the ACompany.ear and the ACompanyProcess.ear on the server, for example a staging or test server, the application works fine.
4. Let's make some modifications to the business model, change some of the EJB code that is used in the business process.
5. Export and redeploy the ACompany.ear application.
6. We will find the application not working or not working exactly as we expected, no matter how and what we change in the code.

The problem is with the duplication of the ACompanyEJB.jar. Since the business process applications are loaded first and the ACompanyProcess has the old ACompanyEJB.jar, the process container will use that package instead of our new one.

The solution to the problem is to redeploy not only the enterprise application with the business model, ACompany.ear, but to redeploy the business process also, ACompanyProcess.ear.

Supporting database

The business process container requires a database to persist the process templates, instances, and states. The database is registered in WebSphere as a data source. Therefore, any WebSphere supported database can be used.

WebSphere Enterprise provides scripts for different database products to configure the database for Process Choreographer.

For more information about Process Choreographer configuration, refer to Chapter 5, “Process Choreographer runtime environment” on page 171.

Application server services

The WebSphere Application Server architecture is designed in a way that new application server services can be easily developed and plugged into the existing server architecture.

The Programming Model Extensions in WebSphere Enterprise are implementations of service extensions. The extensions are then installed on top of the base application server. These services provide those additional functions and features that we know as Programming Model Extensions (PME) in WebSphere Enterprise V5.

Services, PMEs, Enterprise Applications

The question arises: how can one use the services provided in WebSphere Enterprise? Think about the base services provided by the application server, such as Web container, EJB container, and Embedded HTTP Server. These services are either accessed by using a specific API, for example a servlet API or EJB API, or the service interacts directly with the deployed code and content, for example serving HTML files.

The services in WebSphere Enterprise work the same way. Some services can be accessed using the provided API, for example Scheduler service, Internationalization (I18N) service, and Shared Work Area service. Other services work behind the scenes and use the Deployment Descriptors, or extensions in the descriptors, for example Last Participant Support. Again other services require a specific enterprise application that provides access to the services to other applications, for example Dynamic Query.

Figure 17-3 on page 639 summarizes how a custom enterprise application can access the enterprise services.

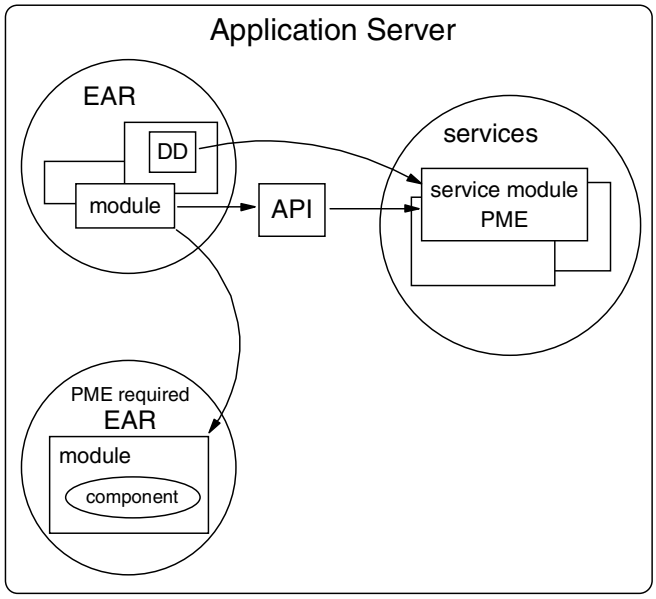


Figure 17-3 Using the application server services

Table 17-1 is a list of the extensions and the provided method of accessing them.

Table 17-1 Accessing PMEs in WebSphere Enterprise

Extension	API access	Deployment Descriptor	Special EAR provided
Process Choreographer	X (for the client)		X (application itself is a special EAR)
Extended Messaging	X (Wizards provided in Studio)		
Asynchronous Beans	X		
Application Profiling, Access Intent		X	
Business Rule Beans	X		X
Dynamic Query	X		X
Startup Beans	X		

Extension	API access	Deployment Descriptor	Special EAR provided
Scheduler service	X		
ActivitySession	X		
Last Participant Support		X	
Object pools	X		
Shared Work Area	X		
Internationalization (I18N) service	X		
CORBA support	X		

Supporting database

Some of the Programming Model Extensions require a supporting database to persist states for different objects and services. Extensions that require a supporting database:

- Process Choreographer (see the previous section)
- Business Rule Beans
- Scheduler service

Process Choreographer

Process Choreographer administration involves the following tasks:

- Installing and configuring the Business Process Container, including the supporting database and the message provider
- Deploying processes
- Managing processes: starting, stopping
- Configuring the staff support

For more information about the administration tasks related to this extension, refer to Chapter 5, “Process Choreographer runtime environment” on page 171.

Extended Messaging

Extended Messaging administration involves the following tasks:

- Configuring the application server resources for Extended Messaging
- Creating and configuring the message provider

For more information about the administration tasks related to this extension, refer to 6.5, “Configuration” on page 258.

Asynchronous Beans

Asynchronous Beans require that WorkManager resource(s) be configured for the application server.

For more information about the administration tasks related to this extension, refer to 7.7, “Configure” on page 329.

Application Profiling and Access Intent

Application Profiling and Access Intent are defined at development time. There are no deployment or administration tasks related to this feature.

Transactional services

Transactional services cover multiple features in WebSphere Enterprise. The administration tasks apply to some of these features:

- Enabling Last Participant Support for an application
- Configuring ActivitySession for the application server

For more information about the administration tasks related to this extension, refer to Chapter 9, “Transactional Services” on page 385.

Business Rule Beans

Business Rule Beans administration involves the following tasks:

- Creating and administering the supporting database for the rules
- Managing the rules using the Rule Management application

For more information about the administration tasks related to this extension, refer to 10.6, “Deployment” on page 448 and 10.6.1, “Running the Rule Management Application” on page 449.

Dynamic Query

Dynamic Query administration involves the following tasks:

- Installing the supporting enterprise application (query.ear)
- Changing the application server’s class loader policy

For more information about the administration tasks related to this extension, refer to 11.6, “Configuration” on page 482 and 11.7, “Deployment” on page 483.

Startup Beans

Startup Beans do not require any specific administration for the application server.

The order of loading applications and modules can be specified, and it may have an impact on the Startup Beans, for example making sure that certain resources from applications are available by the time a Startup Bean fires.

Scheduler service

Scheduler service administration involves the following tasks:

- Creating and administering the supporting database
- Configuring the WorkManager service
- Configuring the Scheduler resources
- Scheduler may require that you install a supporting application for SchedulerCalendars.ear

For more information about the administration tasks related to this extension, refer to 13.8, “Configuration” on page 527.

Object pools

Object pools administration involves the following tasks:

- Configuring (enable/disable) the Object pool service for the application server.
- Configuring the Object pool resources.

For more information about the administration tasks related to this extension, refer to 14.7, “Configuration” on page 552.

Shared Work Area

The Shared Work Area service can be configured for the application server.

For more information about the administration tasks related to this extension, refer to 15.6, “Configuration” on page 580.

Internationalization service

The Internationalization (i18n) service can be configured (enable/disable) for the application server.

For more information about the administration tasks related to this extension, refer to 16.8, “Configuration” on page 622.

17.3 Administration

The most important rule for WebSphere Application Server Enterprise when using Network Deployment Manager is that all application servers in the cell must be Enterprise application servers. The different editions of the application server (Express, base, Enterprise) cannot mix in one cell under Network Deployment.

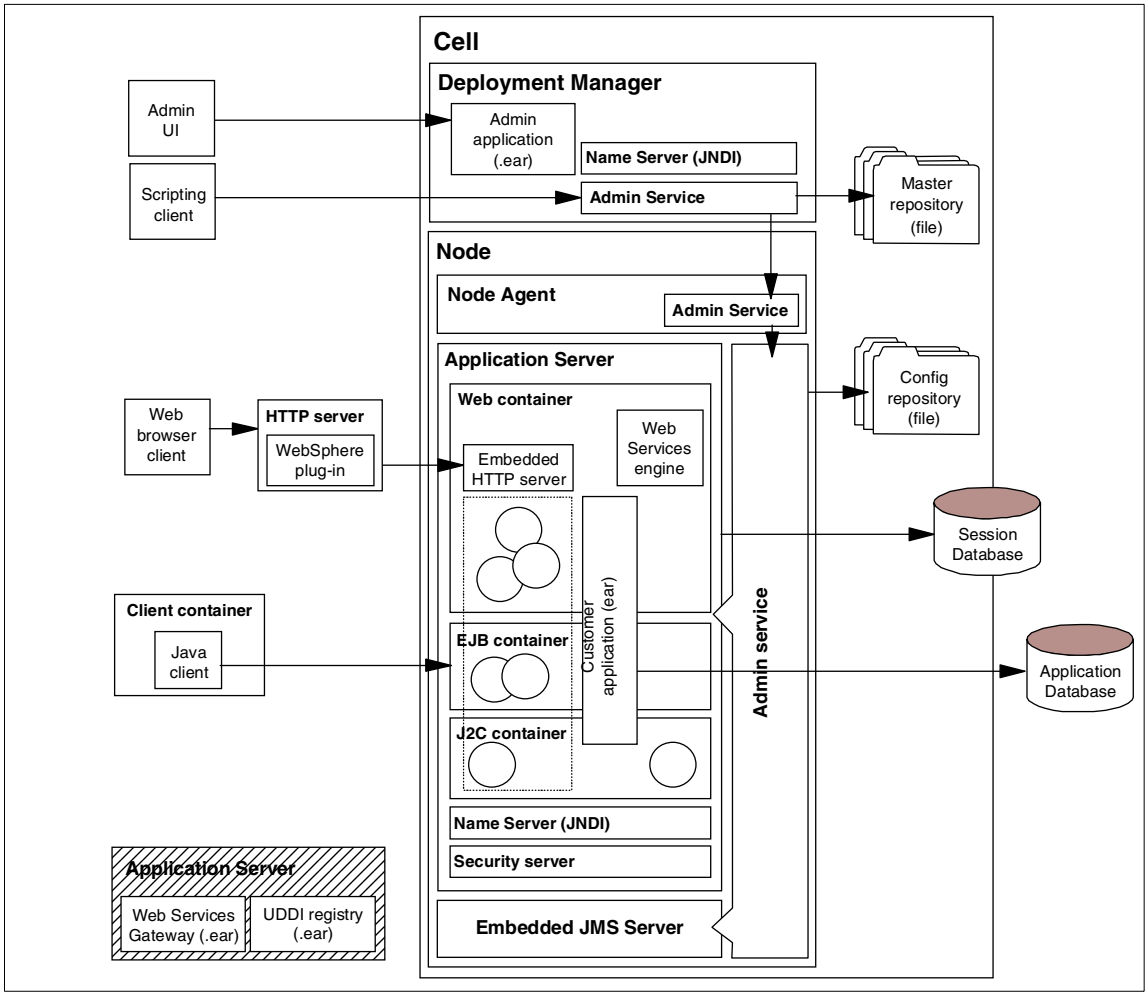


Figure 17-4 WebSphere Application Server (base) in a cell using Network Deployment Manager

Install Administrative Console on a new server

The preferred way of managing multiple application servers is to install the WebSphere Network Deployment application and use the Deployment Manager to manage the application servers.

Although there are situations where more than one application server instance is required but you may not want to use the Deployment Manager. These cases are: development and testing environments, staging servers, and hosting servers to provide individual application servers for multiple users.

This section provides information about the installation of the Administrative Console for a new application server, including an explanation of some minor difficulties with the installation.

First, we need to understand how WebSphere Enterprise is organized and what happens to the Administrative Console when we install the product.

WebSphere Application Server Enterprise is a collection of application server extensions called Programming Model Extensions (PME). The extensions are installed on top of the base application server. This architecture ensures the compatibility of applications and it also ensures the J2EE compatibility.

At the beginning of the installation, the features and functions selected are the Programming Model Extensions, and only the selected extensions will be installed with the product. The modularization of the product also applies to the Administrative Console. Only those administration modules are installed where the pertinent extension is selected for installation.

For example, if you want to install the Asynchronous Beans and Business Rule Beans extensions, the installation process installs only these two extensions for the application server and it installs only the two pertinent packages for the Administrative Console.

The problem with installing a new Administrative Console is that the provided .ear file from the installableApps directory is only the base application without extensions. The install process does not remember the selected extensions, so it does not install them. The following description provides step-by-step instructions on how to install a new Administrative Console.

Note: At this point we assume that the default server, server1 is configured for the WebSphere Enterprise application server.

The steps below will create a new application server instance also. If you already have the server skip those steps.

1. Make sure that WebSphere Application Server is stopped.
2. Install the interim fix for WebSphere Application Server Enterprise V5 PQ70707. Put the PQ70707.jar in a directory where you can place your interim fixes.

   Note: This problem is solved in WebSphere Application Server Enterprise V5.0.2.

3. Open a command window and set the JAVA_HOME environment variable:

   ```
   set JAVA_HOME=<WebSphere_Root>\java
   ```

   Go to <*WebSphere_root*>\WAS50_efix and run **efixWizard.bat**.
4. With the wizard, select the directory where you can find your interim fixes, then install the PQ70707 interim fix. At the end, close the efixWizard.
5. Start the WebSphere Application Server (server1).
6. Export the Administrative Console application from server1. Open a command window, go to <*WebSphere_root*>\bin, then issue the following command:

   ```
   EARExpander.bat -ear ..\installableApps\adminconsoleE.ear -operationDir
   "<WebSphere_root>\installedApps\<your_Nodename>\adminconsole.ear"
   -operation collapse
   ```

   Where you have to replace <*WebSphere_root*> and <*your_Nodename*> according to your environment.

   Note: There is an E at the end of the adminconsoleE.ear file you export. Make sure you do not overwrite the original adminconsole.ear file.

   At the end close the command window.
7. Launch the Administrative Console and log in.
8. Select **Servers -> Application Servers**, then click **New** to create a new server.
9. Provide the name for the server: `ACompanyServer`.
10. Once the creation of the server is done, save the configuration for WebSphere.
11. Select **Environment -> Virtual Hosts**.
12. Select **admin_host**, then **Host Aliases** on the next page.
13. Click **New** to add a new entry.
    - Host name: *

– Port: 9091

Add another entry:

– Host name: *
– Port: 9044

14. Go back to the Virtual Hosts and select **default_host**, then **Host Aliases**.
15. Add the following entry:

    – Host name: *
    – Port: 9081

    and another one:

    – Host name: *
    – Port: 9444

    Save the configuration for WebSphere.
16. We need to install the Administrative Console for the new server. Select **Applications -> Install New Application**.
17. Browse for the adminconsoleE.ear under the <*WebSphere_root*>\installableApps directory.
18. Step through the installation process. In Step 3, make sure that you assign the application to the ACompanyServer application server.
19. Once the installation is done, save the configuration for WebSphere.
20. We need to stop server1 and start the ACompanyServer.

    Close the WebSphere Administrative Console, then use the following commands (names are case sensitive):

    ```
    stopserver server1
    startserver ACompanyServer
    ```
21. Open the WebSphere Administrative Console, then log in.
22. Check for the extensions in the Administrative Console.

17.4 Workload management

Discussing workload management usually means focusing on how to, within a given runtime environment, divide the load on the resources that are used by an application. By resources we refer to system memory, CPU time, input/output processing, and communications. In our case the runtime environment is WebSphere Application Server Enterprise as a J2EE application server with additional programing and runtime extensions.

In the following sections, we start with a quick overview of WebSphere Application Server base workload management capabilities and then continue with the differences that are particular to enterprise extensions. The focus is just on the application server, although cloning or using multiple instances of other resources such as a Web server and databases is supported with WebSphere Application Server. The recommended resources for more details about the possibilities are:

- *IBM WebSphere V5.0 Performance, Scalability and High Availability,* SG24-6198
- WebSphere Application Server V5 InfoCenter

17.4.1 Scalability and high availability basics

This section is a quick overview of the scalability and high availability basics that can be applied to WebSphere Enterprise.

Figure 17-5 illustrates the basic configuration for a single application server.

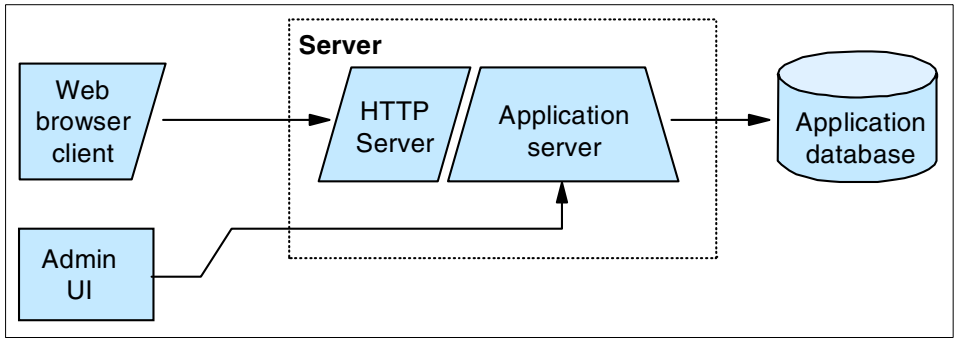


Figure 17-5 Basic configuration, without scalability and high availability

Vertical scaling and cloning

Vertical scaling means installing or having more than one instance of application server on the same machine to get better utilization of its resources. Typically, a multiprocessor machine with enough memory and I/O capabilities is used in this situation. It is meant to have an J2EE application spread over vertical cloned application servers, so the application's request throughput and response time will be better, but you can also have independent J2EE applications among the vertical cloned application server. In either case, machine resources are exploited better, but where we spread the same application, there must be some additional services available to spread and coordinate the workload of one application over multiple J2EE application servers. In this case, it is these services to which we refer when we speak about the workload management

capabilities of WebSphere Application Server. In order to use them you have to create a WebSphere cluster and assign application servers to the cluster.

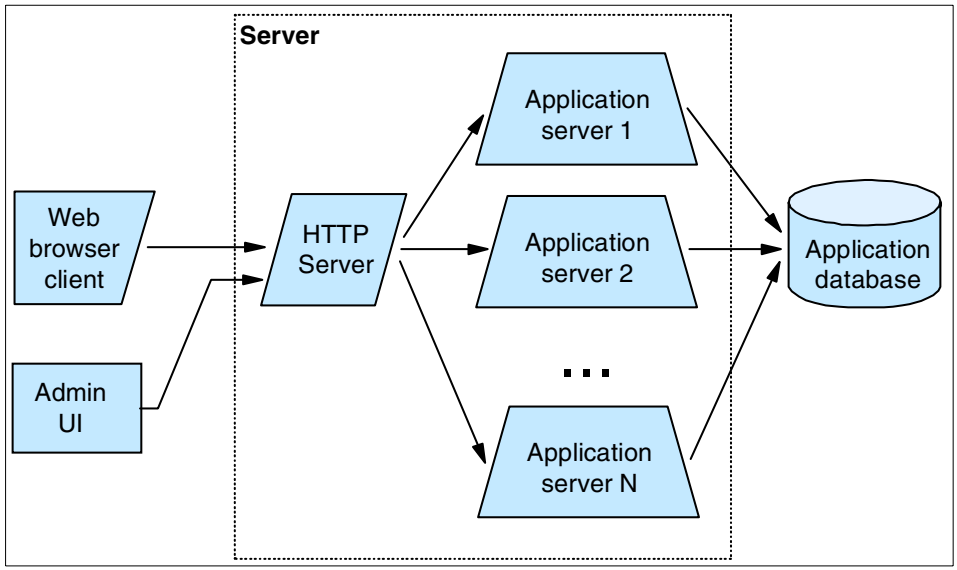


Figure 17-6 Vertical scaling and cloning

Horizontal scaling and cloning

Horizontal scaling means having application server instances installed over more than one machine, because the J2EE application we use is large and thus would take too much resources from one machine, that is, a single machine would be underpowered to run a given application. So the aim is the same as before: to spread the application over more J2EE application servers. In this case the same workload management services are used as before.

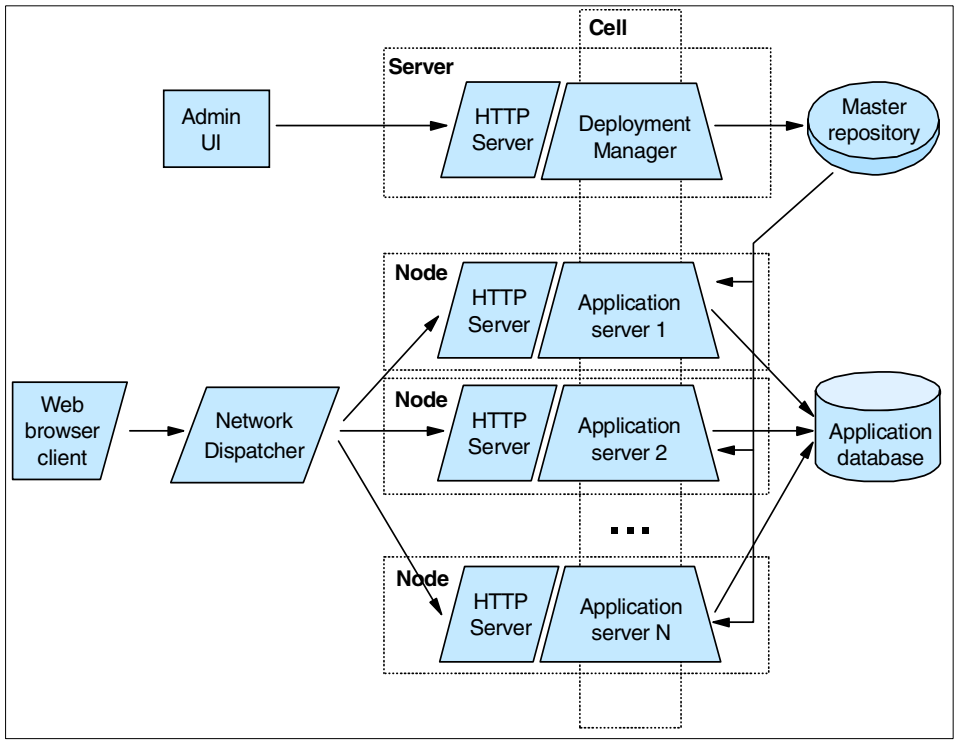


Figure 17-7 Horizontal scaling and cloning

17.5 Where to find more information

- *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195
- *WebSphere Application Server V5 for iSeries™: Installation, Configuration, and Administration*, SG24-6588
- *IBM WebSphere V5.0 Performance, Scalability, and High Availability: WebSphere Handbook Series*, SG24-6198
- *IBM WebSphere V5.0 Security WebSphere Handbook Series*, SG24-6573
- WebSphere InfoCenter at:

  `http://publib7b.boulder.ibm.com/webapp/wasinfo1/index.jsp?deployment=Enterprise&lang=en`

  then select **System Administration**.

Part 3

Appendixes

A

Installation and configuration

This appendix provides an explanation of the procedures for installing, configuring, and verifying IBM WebSphere Application Enterprise.

This chapter concentrates on the decisions and tasks associated with WebSphere Application Server Enterprise. If you need information about the installation of WebSphere Application Server and WebSphere Application Server Network Deployment, refer to the redbook *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series,* SG24-6195.

Planning for installation

There are several ways to install WebSphere Application Server Enterprise. The simplest way is the umbrella installation, which installs both the WebSphere Application Server base product and the Enterprise product during the same installation procedure. You can also add the Enterprise product to the existing base product.

On the umbrella installation, only the base features that Enterprise product requires are installed. The installer installs the base product without the following features:

- IBM HTTP Server
- Web server plug-ins
- Performance and analysis tool

If you must install these features or you want to customize the base product, you can install the base product before installing the Enterprise product. The installation image for the base product is available in the directory WAS50.

If you plan to install the Enterprise product on a multinode environment, you must install Enterprise product on each federated node in the cell and install the Enterprise product on the Deployment Manager node as well. IBM WebSphere supports a cell of homogeneous nodes. If you are planning to use the Enterprise extensions on any application server node in the cell, you must install the Enterprise product on all nodes, including the Deployment Manager node.

The Enterprise Administrative Console extensions are not installed on a federated node. If you later decide to remove the node from the cell, you must uninstall and reinstall the Enterprise product on the node to get the Enterprise Administrative Console extensions.

Installations

If you are planning to install an umbrella installation, see “Install Enterprise and base at the same time” on page 655. If you already have the existing base product or you want to customize the base product before installing the Enterprise product, see “Install Enterprise to the existing base” on page 658.

If you are planning to install the Enterprise product to the existing multinode environment, see “Install Enterprise to the existing Network Deployment” on page 661 to install on the Deployment Manager node, and see “Install Enterprise to the existing base” on page 658 to install on the application server node.

Install Enterprise and base at the same time

WebSphere Application Server Enterprise requires the WebSphere Application Server base product. If the base product is not installed, the Enterprise installer works as an umbrella installation that installs both the base product and the Enterprise product.

Installation steps

The installation steps are the following:

1. Log on as an administrator user in the local server domain (not part of a Windows domain).
2. Insert the IBM WebSphere Application Server Enterprise V5 CD into the CD-ROM drive.
3. Using the Windows Explorer, switch to the \nt directory on the CD. Double-click **LaunchPad.bat** to start the installation.
4. Select a language for the LaunchPad and click **OK**.
5. Review the Readme file and installation guides.
6. Click **Install the product**. Then the installer starts.
7. Select a language for the installer and click **OK**.
8. On the Welcome window, click **Next** to continue.
9. The software license agreement window will appear. Read the agreement and check **I accept the terms in the license agreement**, then click **Next** to continue.
10. Select **Typical** if you want to install Embedded Messaging, Business Rule Beans, Extended Messaging, Dynamic Query, Additional Enterprise Extensions, Scheduler and Asynchronous Beans, Process Choreographer, and their samples. Select **Custom** if you want to customize your installation.
11. If you select **Custom**, select the features to install. For guidance on which features to install, see Table A-1 on page 656.

Table A-1 Enterprise features and feature dependencies

Components	Dependency			Feature description
	Embedded Messaging	Additional Enterprise Extensions	Scheduler and Asynchronous Beans	
Embedded Messaging				Installing and configuring a JMS provider
Server and Client				
Message-driven Bean Samples				
Business Rule Beans				Externalizing business rules with Business Rule Beans
Business Rule Beans Sample				
Extended Messaging				Using Extended Messaging in applications
Extended Messaging Sample	X			
Dynamic Query				Using the Dynamic Query service
Dynamic Query Sample				
Additional Enterprise Extensions				Using the following functions in applications ► Internationalization service ► Application Profiling ► Object pools ► Startup Beans ► WorkArea service ► ActivitySession service ► Last Participant Support
Additional Enterprise Extension Sample				
Scheduler and Asynchronous Beans				Using Scheduler service and Asynchronous Beans
Scheduler and Asynchronous Beans Sample	X			

Components	Dependency			Feature description
	Embedded Messaging	Additional Enterprise Extensions	Scheduler and Asynchronous Beans	
Process Choreographer		X	X	Using Process Choreographer
Process Choreographer Sample				
Configure sample BPE container	X			
CORBA C++ SDK				Implementing CORBA applications
Interface Repository Support (requires DB2)				
CORBA C++ SDK Sample				
JavaDoc				Installing WebSphere Application Server Release 5 API document in JavaDoc format

12. The next window asks for the install directories for IBM WebSphere Application Server and Embedded Messaging. Verify the install directories and click **Next** to continue.

13. The next window asks the node name and host name for this installation. The node name must be unique within a cell. Verify them and click **Next** to continue.

14. The next window will allow you to choose to run WebSphere Application Server as a service. Windows services can be automatically started, and startup and recovery operations can be configured. Check if you want to elect it as a service and click **Next** to continue.

15. The last window before the installation begins shows a summary of all of the choices. Click **Next** to begin the installation.

16. At first, the WebSphere Application Server base product and interim fixes are installed. Then the installer will begin the progress of installing Enterprise features and samples that were selected during the installation.

17. The last step of the installation gives you the opportunity to register the product. Click **Next** to continue.
18. Click **Finish** to complete the installation.
19. After completing the installation, the First Steps windows comes up. From this window, you can start the server and verify the installation.

Verify the installation

To verify Enterprise functions, run the samples. If you did not install samples associated with the extensions you are verifying, return the installation and use the **Custom** installation type to add additional samples. The steps to verify Enterprise extensions are the following:

1. Start the server using one of these methods.
   a. Select the **Start the server** option on the First Steps window.
   b. Issue the command **<WAS_ROOT>\bin\startServer server1** from a command window.
2. Run the Sample Gallery using one of these methods:
   a. Select the **Samples Gallery** option on the First Steps window.
   b. Invoke a Web browser to access the following:

      ```
      http://localhost:9080/WSsamples/en/index.html
      ```

   c. Select **Start -> Programs -> IBM WebSphere -> Application Server v5.0 -> Samples Gallery**.
3. Follow the links and instructions on this page to access samples of Enterprise extensions.

Running the sample successfully indicates that the functionality of the extension is fully operational.

Note: On the First Steps window, there is one menu, that is the Verify Installation menu. On this menu, only the base functions are verified.

Install Enterprise to the existing base

If you already installed the base product, you can add the Enterprise extensions to it. When you invoke the installer, the installer detects the existing products and shows you a list of them. Check **Add to the existing copy of WebSphere Application Server, v5.0** and select one of the base installation instances on the window shown in Figure A-1 on page 659.

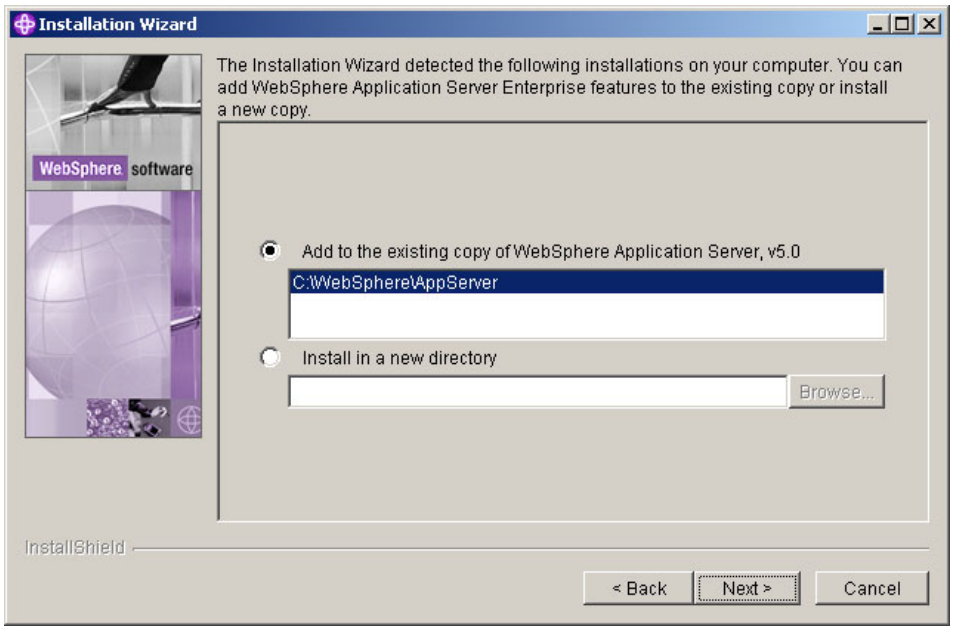


Figure A-1 The window to select the existing base product

After you select the existing base product, you will see the list of the Enterprise extensions. If you did not install the base features that the selected Enterprise extensions requires, the installer will install them automatically. Table A-2 shows you the dependencies between the base features and the Enterprise extensions.

Table A-2 Enterprise features and product feature dependencies

Enterprise Feature	Base product feature						
	Application Server	Admin Scripting	Administrative Console	Application Assembly Tool	Application Server sample	Embedded Messaging	Message-driven bean sample
Business Rule Beans	X	X	X				
Business Rule Beans sample					X		
Extended Messaging	X					X	
Extended Messaging sample					X		X

Enterprise Feature	Base product feature						
	Application Server	Admin Scripting	Administrative Console	Application Assembly Tool	Application Server sample	Embedded Messaging	Message-driven bean sample
Dynamic Query	X						
Dynamic Query sample					X		
Additional Enterprise Extensions	X		X	X			
Additional Enterprise Extension sample					X		
Scheduler and Asynchronous Beans	X						
Scheduler and Asynchronous Beans sample					X		
Process Choreographer	X	X	X				
Process Choreographer sample					X		
Configure a sample BPE container						X	
CORBA C++ SDK							
Interface Repository Support (Requires DB2)							
CORBA C++ SDK sample				X			
JavaDoc							

If you did not install the Embedded Messaging and select to install it or it is required by the other features, you need to specify the directory that the Embedded Messaging is installed.

The installer will install the required base features, apply the interim fixes that are required by Enterprise extensions, then install Enterprise features.

Verify the installation

The way to verify the installation is the same as umbrella installation. See “Verify the installation” on page 658.

Install Enterprise to the existing Network Deployment

On the multinode environment, you must install the Enterprise product on all the nodes in the cell and Deployment Manager node. This section covers how to install the Enterprise product to the Deployment Manager node where the Network Deployment product is installed. To install the Enterprise product on the application node, see “Install Enterprise to the existing base” on page 658.

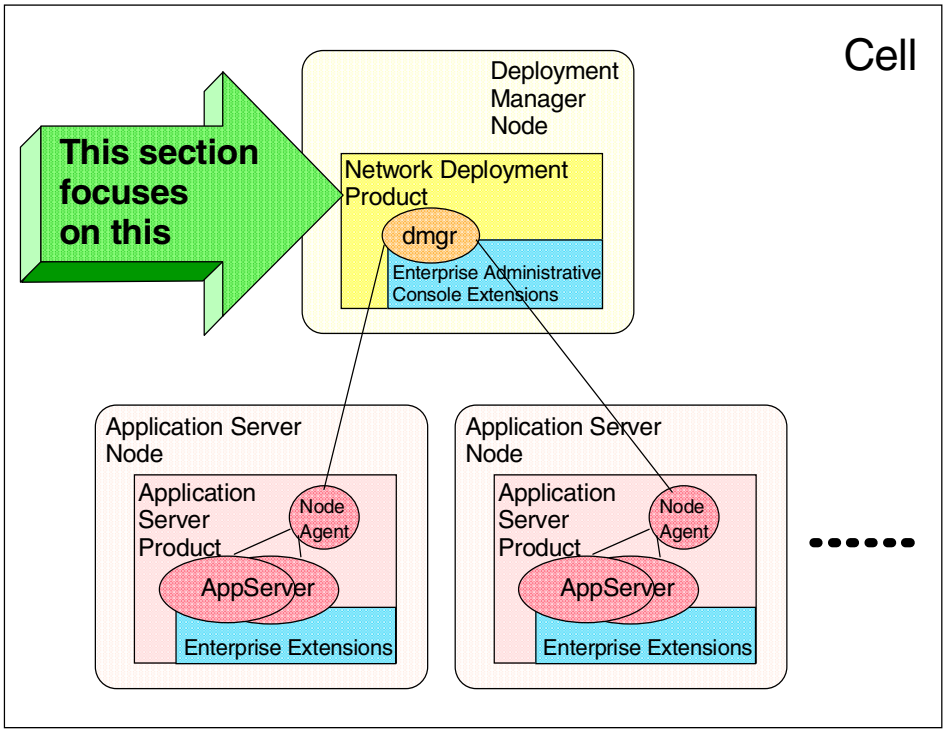


Figure A-2 The multinode environment

There is no way to install the Network Deployment product and the Enterprise product at the same time. You must install the Network Deployment product before you install the Enterprise product.

When you invoke the installer, the installer detects the existing products and shows a list of them. Check **Add to the existing copy of WebSphere**

Application Server Network Deployment, v5.0 and select one of the base installation instance on the window shown in Figure A-3.

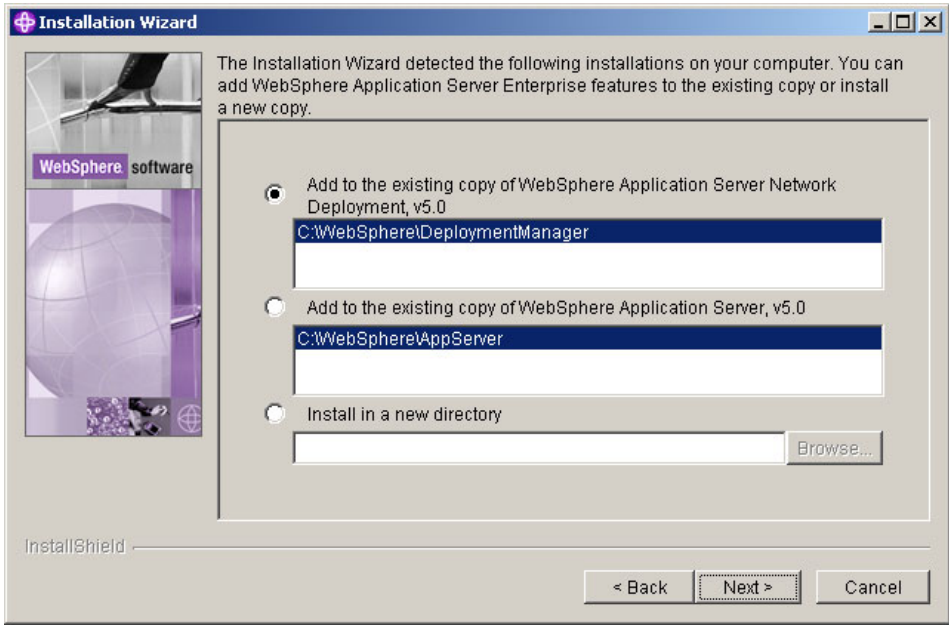


Figure A-3 The window of select the existing Network Deployment product

Only administrative extensions that extend the Administrative Console and the scripting facilities are installed on the Deployment Manager. There are no choices to select the features. When installing on the existing Network Manager node, the installer automatically installs the Administrative Console extensions feature. No base features and no Enterprise extensions are installed on the Network Manager node.

Verify the installation

On the Network Deployment, only the extension of the Administrative Console is installed. To verify the installation, start the Deployment Manager and invoke a Web browser and open the Administrative Console by accessing the following:

```
http://localhost:9090/admin/
```

You will find some additional resources for the Enterprise extensions.

Troubleshooting the installation

If you received some errors during the installation or the verification, check the log files. Table A-3 shows you where the log files are located.

Table A-3 installation log files

Component		Log Filename
base product		
	<WAS_ROOT>\log	
	Application Server	log.txt
	Embedded messaging feature installation log	mq_install.log
	Embedded messaging feature configuration log	createMQ.<node>.server1.log
	Default Application	installDefaultApplication.log
	Samples Gallery	installSamples.log
	Administrative Console	installAdminConsole.log
Enterprise product		
	<WAS_ROOT>\log	
	Enterprise	log.txt PMEinstallSummary.log WAS.PME.install.log
	Administrative Console	installAdminConsole.log
	<WAS_ROOT>\log\pme	
	Enterprise	PluginProcessorInst.log PMEinstallApps.log

You can also find the error logs for each Enterprise component in <*WebSphere_root*>\logs\pme\PluginProcessor and the log files for interim fixes in <*WebSphere_root*>\logs\pmeefixlogs.

If the installation logs do not contain enough information to determine the cause of the problem, turn on tracing. The installer will report the stdout and stderr logs to the console window by adding the `-is:javaconsole` option to the **`Install`** command:

```
Install -is:javaconsole
```

To capture additional information to a log, add `-is:log <filename>` to the **`Install`** command:

```
Install -is:log C:\temp\install.log
```

Configuration

To configure the WebSphere Application Server Enterprise, invoke a Web browser and open the Administrative Console by accessing the following:

```
http://localhost:9090/admin/
```

For detailed configurations for each Enterprise extensions, see the configure part of each runtime chapters.

If you are planning to use external data sources, the following information is available to help you:

- Go to the InfoCenter at `http://publib7b.boulder.ibm.com/wasinfo1/index.jsp` and click **Resources -> Data access -> Accessing data from applications -> Data sources**.

If you are planning to use WebSphere MQ as the JMS provider, the following information is available to help you:

- *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series,* SG24-6195
- Go to the InfoCenter at `http://publib7b.boulder.ibm.com/wasinfo1/index.jsp` and click **Resources -> Messaging -> Using JMS and messaging in applications -> Administering WebSphere JMS support -> Installing and configuring a JMS provider -> Installing WebSphere MQ as the JMS provider**.

If you are planning to turn on security, the following information is available to help you:

- *IBM WebSphere V5.0 Security WebSphere Handbook Series*, SG24-6573
- Go to the InfoCenter at `http://publib7b.boulder.ibm.com/wasinfo1/index.jsp` and click **Security**.

If you are planning to make a multinode environment, the following information is available to help you:

- *WebSphere V5.0 Applications: Ensuring High Performance and Scalability*, SG24-6198

B

Sample scenario

This appendix is a detailed description of the sample application that is provided with this redbook.

The sample application is based on the sample scenario introduced in Chapter 3, “Sample scenario” on page 25. The application implements some of the functions that are necessary to show the functions and features in WebSphere Enterprise V5.

Sample application

The sample application is a partial implementation of the sample scenario used in this book. The main objective of the sample is to show the functions and features specific to WebSphere Enterprise V5. This appendix walks through the application showing the technical details, in order to get a better understanding.

User registry

You can use either the operating system (OS) or LDAP user registry. To make it simpler we recommend that you use the local OS for the sample application.

Create the users listed in Table B-1 in your user registry.

Table B-1 Users

username	password
wsadmin	passw0rd
dbuser	passw0rd
bpedbuser	passw0rd
jmsuser	passw0rd
client01	passw0rd

If you are using Windows, make sure that the Administrator has the “Act as part of the operating system” privilege.

Database

The following steps are required to create and populate the database for the sample application. It is the same for the base and the extended sample application. Some database items are only used in the extended sample.

1. Open a command window or a terminal window with the DB2 environment.
2. Run the **`setup.bat`** script from the database directory. It creates the database, creates the database tables, and populates the database.
3. Make sure that the script ran successfully. You can check the responses from the individual commands or you can open a DB2 Control Center and check for the generated database and database tables.
4. Close the command or terminal window.

Development environment

The instructions below explain how to set up the development test environment for the sample application. There are two versions of the sample application shipped together with the book. One is the base version, without extensions. The other is a more complex one with several extensions to the original code.

Important: The instructions in this section apply to the base sample with no extensions.

Start WebSphere Studio Application Developer Integration Edition. In order to have the security working with the test server, we need to start the application under a local operating system user, for example Administrator. We also need to specify the location of the workspace for the workbench. The command to start up WebSphere Studio IE with the required settings is:

```
runas /user:Administrator "<WebSphere_Studio_IE_root>\wsappdevie.exe -data
<workspace_location>"
```

Where <*workspace_location*> is the directory for the workspace, for example `C:\SG246932\ACompanyBase`.

Note: The administrator has to have the “Act as part of the operating system” user right assigned.

Importing the sample application

Follow these steps to import the sample application:

1. Open or switch to the Resource perspective in WebSphere Studio IE.
2. Select **File -> Import** from the menu.
3. Select **Existing Project into Workspace** on the Import window, then click **Next**.
4. Browse for the ACompany folder under the directory where you have the project extracted, in our example `C:\SG246932\ACompanyBase\ACompanyBase`. Click **Finish**.

   The directory will be imported to the workspace. You will see a list of warnings and errors in the Task view, but do not worry about them at this moment.
5. Import these directories following steps 2 to 4:
   - ACompanyEJB
   - ACompanyWeb
   - ACompanyClient

– ACompanyService

The ACompanyServices services project holds the source for the business processes. The next step is to generate the deployable code for the processes.

1. Open or switch to the Business Integration perspective and select **Service Projects -> ACompanyServices -> com.acompany**.
2. Right-click the **PO.process** file and select **Enterprise Services -> Generate Deploy Code**. The Generate Deploy Code window appears.
   a. Select **EJB** for the Inbound binding type.
   b. Specify the EAR project name. ACompanyProcess. The EJB project name should change automatically to ACompanyProcessEJB.
   c. Click **Finish**.

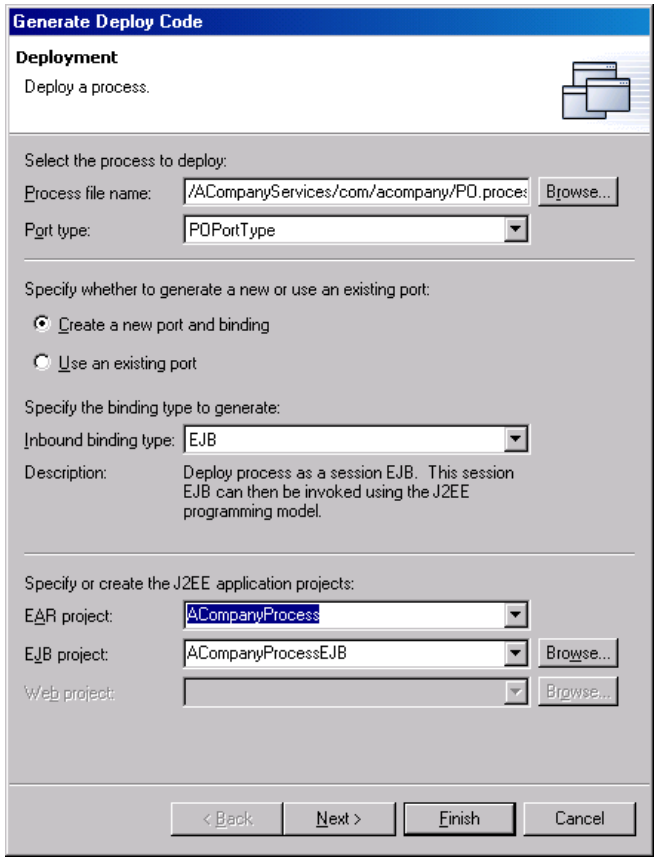


Figure B-1 Generate Deploy Code window

It may take a minute to generate the deployed code for the processes.

3. Repeat step 2, selecting the **UpdateCatalog.process** business process to generate the deployed code.
4. Generate the EJB deployed code for all the EJB modules. Open or switch to the J2EE perspective.
   a. Select the **EJB Modules**, select the modules one-by-one, right-click the module, then select **Generate -> Deploy and RMIC Code**.
   b. Click **Select all** to generate the code for all EJBs.
   c. Click **Finish** and wait until the code is generated.
5. Repeat the previous step for all EJB modules.
6. Select **Project -> Rebuild All** from the menu. It may take a few minutes to rebuild all the projects on the workspace.

   At the end, you should have 30 items on the Task view: four errors, eight warnings and 18 information items. This is normal.

Importing the extended sample application

The import of the extended sample application is identical to the base application. Follow the steps found in “Importing the sample application” on page 667.

Use the directory C:\SG246932\ACompanyBase\ACompanyExt in step 4 on page 667.

At step 5 import the following directories:

- ACompany
- ACompanyApprovalWeb
- ACompanyBrbEAR
- ACompanyBrbEJB
- ACompanyClient
- ACompanyClientI18N
- ACompanyEJB
- ACompanyEMS
- ACompanyEMSEJB
- ACompanyEMSWeb
- ACompanyWeb
- ObjectPtest
- ObjectPtestEAR
- WorkAreaTestClient

There are two different process projects: one is for an extended process Web client, and one is for a customized process Web client. Select the one you need and move the project directories to the C:\SG246932\ACompanyBase\ACompanyExt directory.

The extended process Web client projects are in the C:\SG246932\ACompanyBase\ACompanyExt.bpewebclient directory. Copy them to the workspace directory, then import them.

- ACompanyProcessWeb: You will have to add this project as a Web module to the ACompanyProcess enterprise application, once it is created while generating the deployed code for the processes.
- ACompanyServices

The following customized process Web client projects are in the C:\SG246932\ACompanyBase\ACompanyExt.MyBpewebclient directory. Copy them to the workspace directory, then import them.

- ACompanyServices
- MyBpewebclient
- MyBpewebclientEAR

Configuring the test environment

This section shows how to set up the test server for the sample application.

Create the test server

The first step is to create the WebSphere Enterprise V5 test server.

1. Open the Server perspective.
2. Select **File -> New -> Server and Server Configuration**. This will start the server wizard.
3. On the first window enter the Server name `ACompanyServer`, the folder name `Servers`, and the server type `5.0 EE Test Environment`. Click **Next**.
4. When asked Do you want to create a new server project with the name Servers?, click **Yes**.
5. On the next window, where the HTTP port number is set to 9080, click **Finish**. Wait until the system creates your test server and the initial configuration.
6. Open the server configuration for the ACompanyServer by double-clicking the **ACompanyServer** entry in the Server Configuration view under the Servers folder.
7. Select the **Configuration** tab and check **Enable administration console**.
8. Set the Application class loader policy to SINGLE under this tab.

9. Select the **Security** tab, click **enable security**, then set the Server ID and password using the operating system user name and password that started WebSphere Studio IE; in our case this is `Administrator`.
10. Select the **Variables** tab, find the DB2_JDBC_DRIVER_PATH entry under the Node Settings, then click **Edit**. Provide the path for the DB2 Java directory, for example `C:\SQLLIB\java`.
11. Select the **Data source** tab. Under the server settings, add a new JDBC provider with the following details:
    - Database type: `IBM DB2`
    - JDBC provider type: `DB2 JDBC Provider (XA)`
    - Name: `DB2 XA JDBC provider`
    - Implementation class name: `COM.ibm.db2.jdbc.DB2XADataSource`
    - Classpath: `${DB2_JDBC_DRIVER_PATH}/db2java.zip`
12. Save and close the configuration.

Configure the test server for the base application

Once the test server is available, the next step is to configure the server.

1. Open the server configuration for the ACompanyServer by double-clicking the **ACompanyServer** entry in the Server Configuration view in the Servers folder.
2. Create a new JAAS authentication entry with the following details:
    - Alias: `dbuser_alias`
    - User ID: `dbuser`
    - Password: `passw0rd`
3. Add a new data source for the new JDBC provider with the following details:
    - JDBC provider: `DB2 JDBC Provider (XA)`
    - Data source type: `Version 5.0`
    - Name: `redbookDS`
    - JNDI name: `jdbc/redbookDS`
    - Data source helper class name: `com.ibm.websphere.rsadapter.DB2DataStoreHelper`
    - Component-managed authentication alias: `dbuser_alias`
    - Container-managed authentication alias: `dbuser_alias`
    - databaseName: `REDBOOK`

    Select **Use this data source in container managed persistence (CMP)**.
4. Save and close the configuration.

5. Right-click the **ACompanyServer** entry in the Server Configuration view in the Servers folder and add the following applications to the server:
   - ACompany
   - ACompanyProcess
6. At the very end, the processes have to be deployed. Right-click the **ACompanyServer** entry in the Server Configuration view in the Servers folder and select **Deploy Process**. The deployment may take a couple minutes.
7. The test environment is ready to run in WebSphere Studio IE. You can start the test server by right-clicking the **ACompanyServer** entry in the Server Configuration view and selecting **Start**. The startup process may take a couple minutes. In the meantime you can check the trace in the Console view.

Additional configurations for the extended sample application

During the configuration you will have to start and stop the test application server a couple of times. Some components may fail to start during the startup, because of missing configuration steps during the process. Do not worry about these problems at this point. The server has to start up without any problems once all the configurations are done.

Follow these steps to configure the test server for the sample application:

1. Right-click the **ACompanyServer** entry in the Server Configuration view under the Servers folder and add the following applications to the server:
   - ACompanyEMS
   - ACompanyBrbEAR
   - MyBpewebclientEAR
2. Before you proceed to the following steps, turn off security for the ACompany application server in WebSphere Studio IE. There is a bug that prevents you from logging in to the Administrative Console when the application server is secured. Once you are done with the configuration steps with the Administrative Console, enable security again before you restart the server.

   Open the server configuration and uncheck the **Enable security** box in the Security tab.
3. Extended Messaging configurations

   Follow the steps in 6.4.1, “Configure Extended Messaging” on page 250 to configure the test server for the Extended Messaging sample.
4. Business Rule Bean configurations
   a. Create the Business Rule Beans database by following the steps in “Creating the Business Rule Beans database” on page 427.

b. Set up the Rule Management Application by following the steps in "Configure Rule Management Application" on page 428.

c. Configure the test server by following the steps in "Configure application server" on page 432.

d. Start the test server.

Once the server is running, the business rule has to be configured. Follow the steps in 10.3.3, "Creating and configuring the rule" on page 436.

5. Scheduler service configurations. The Startup Bean sample uses the Scheduler service. There are two options:

– Configure the Scheduler service first and use it together with the Startup Bean sample. This section follows this course.

– Modify the Startup Bean code and remove the scheduler part. Check the StartUpBean.java source code for comments about disabling the work scheduling part in the bean.

a. Start the test server if it is not running.

b. Create a new work manager for the application by following the steps in "Work manager service in relation to the Scheduler service" on page 527.

c. Create the Scheduler database for the service. Follow the instructions in "The Scheduler database" on page 528.

d. Create a new Scheduler service by following the steps in "Scheduler configuration using the Administrative Console" on page 530.

6. Startup Bean configurations:

a. Open the server configuration for the ACompanyServer by double-clicking the **ACompanyServer** entry in the Server Configuration view in the Servers folder.

b. Switch to the Applications tab and select the **ACompanyProcess** application.

c. Change the Start weight to 9. This means that this application starts earlier than the ACompany application, so the resources from ACompanyProcess will be available for the Startup Bean.

7. Dynamic Query configurations

a. Import the Dynamic Query supporting enterprise application into Studio. Follow the steps in "Development environment setup" on page 467.

b. Add the Query application to the test server. You have to stop the server for this operation.

8. Object pool manager configuration

a. Start the test server if it is not running.

b. Follow the configuration steps in “Object pool manager configuration” on page 553.

9. Configure the Shared Work Area service by following the steps in 15.6.1, “Shared Work Area service configuration” on page 581.
10. Configure the Internationalization (I18N) service for the test server by following the instructions in 16.4, “Unit test environment” on page 598.
11. At the end, enable security for the application server. Before you can do that, make sure that the test server is stopped. Open the server configuration, then check the **Enable security** box in the Security tab.
12. Start the test server and check the Console view to see if there is any error during the server startup.

Running J2EE clients in WebSphere Studio

This section shows you how to run J2EE clients with the test server in the development environment:

1. Open or switch to the Java perspective.
2. Select **Run -> Run...** from the menu.
3. Select **WebSphere V5 Application client** on the left side of the window.
4. Click **New**.
5. Specify the name, for example `MyClient`
6. In the Application tab, select the server type: **WebSphere V5 EE**. Select the Enterprise Application **MyApplication**.
7. In the Arguments tab, you can specify the client application, in case you have multiple J2EE clients under one enterprise archive. Use the -CCjar directive. For example if you have a client called `MyClient`, add the following to the Program arguments line:

   ```
   -CCjar=MyClient.jar
   ```

   If you have other command-line arguments you need to pass to the application, specify them at the end of the Program arguments line.
8. You can also specify additional libraries for the application in the Classpath tab.
9. Click **Apply**, to save the configuration in WebSphere Studio.
10. Click **Run** to run the application. Make sure you have the application server running for the client.

Runtime environment

This section provides step-by-step instructions for setting up the runtime environment for the sample application.

Configuring the runtime environment

The step is to create the database and populate it. In order to do that, follow the steps in “Database” on page 666.

1. Create a new application server instance and install the Administrative Console for the new instance. For detailed steps, refer to “Install Administrative Console on a new server” on page 644.
2. Start the application server if it is not running and launch the Administrative Console.
3. Select **Environment -> Manage WebSphere Variables** and select the **DB2_JDBC_DRIVER_PATH** variable.

   Change the value to `<DB2_root>\java`, where `<DB2_root>` is the directory where DB2 is installed, for example `C:\SQLLIB`.

   Save the configuration for WebSphere.
4. The next step is to install the business process container.
5. Click **Servers -> Application Servers -> ACompanyServer**.
6. In the Additional Properties section, click **Business process container**.
7. Ignore the business process container settings. Click the **Business Process Container Install Wizard** link at the bottom, and follow the step-by-step instructions provided.
8. On the Select the Database Configuration for the Business Process Container window, select **DB2** for the new XA data source.
9. In the classpath text box, enter `C:\SQLLIB\java\db2java.zip`.
10. Enter the data source user name: `bpedbuser` and password `passw0rd`.
11. Change the database name to `ACBPEDB` under the custom properties.
12. Click **Next**.
13. On the Select JMS Provider and Security window, leave the JMS provider: WebSphere JMS Provider.
14. Set the JMS User ID as `wsadmin`, and the password as `passw0rd` (it is the WebSphere administrator user).

15. In the Business Process Container Security Configuration section, set Security Role Mapping as `wsadmin`, set the JMS API User ID as `wsadmin`, adn the JMS API password as `passw0rd`.

    Important: Use short and all lowercase names for JMS user and JMS API user. If you use `Administrator` on Windows, it will not work!

16. Click **Next**.
17. On the Select JMS Resources window, the **Create new JMS resources using default values** option should be selected.
18. Click **Next**.
19. On the summary window, click **Finish** to create the business process container.
20. At the end of the process, a message appears confirming the successful creation of the container.
21. Save the configuration for WebSphere.
22. Open a DB2 command window. Change the directory to <*WebSphere_root*>\ProcessChoreographer.
23. Issue the command:

    ```
    db2set DB2_RR_TO_RS=YES
    ```

    Restart the database instance in order to make this setting effective.

24. Create the database:

    ```
    db2 create database ACBPEDB
    ```

25. Grant access to the database for the bpedbuser:

    ```
    db2 connect to acbpedb
    db2 grant dbadm on database to user bpedbuser
    ```

26. Open the file in a text editor:

    ```
    createTablespaceDb2.ddl
    ```

    Replace the @location@ occurrences with the location where you want your data stored, for example C:\ACBPEDB. Make sure that the directory exists.

27. To create the DB2 tablespace, issue the command:

    ```
    db2 -tvf createTablespaceDb2.ddl
    ```

    Make sure that the script's output contains no errors.

28. Disconnect then reconnect to the database with the bpedbuser again:

    ```
    db2 disconnect current
    db2 connect to ACBPEDB user bpedbuser using passw0rd
    ```

Note: If you do not connect to the database with the right user, the schema for the tables will be wrong and the Business Process Container EJBs will not be able to find them.

29. To create the DB2 scheme, issue the following command:

    ```
    db2 -tvf createSchemaDb2.ddl
    ```

    Make sure that the script's output contains no errors.

30. Bound the CLI packages to the new database. Issue the following command:

    ```
    db2 bind %DB2PATH%\bnd\@db2cli.lst blocking all grant public
    ```

31. Enable security using the Administrative Console.

32. Select **Security -> User Registries -> Local OS**, then enter the Server User ID `wsadmin`, and password `passw0rd`. Click **OK** and save the configuration for WebSphere.

    Note: You have to have an OS user with the name of `wsadmin` and password of `passw0rd`.

33. Select **Security -> Global Security**.

34. Select **Enable security**, and disable **Enforce Java2 Security** (we do not need Java 2 security for this example).

    Note: Using Windows, make sure that the current operating system user (Administrator) has the “Act as part of the operating system” privilege. WebSphere is not going to be able to validate the user without this setting.

35. Click **OK**, then save the configuration for WebSphere.

36. Log out from the Administrative Console.

37. Stop the server:

    ```
    stopserver ACompanyServer
    ```

38. Start the server (the name is case sensitive):

    ```
    startserver ACompanyServer
    ```

Deploying the base sample application

The instructions in this section apply to the base application without extensions.

1. Make sure that the ACompanyServer is running, then start and log in to the Administrative Console.

2. Select **Security -> JAAS Configuration -> J2C Authentication Data**. Click **New** to create a new entry, then provide the following info:
   - Alias: `dbuser`
   - User ID: `dbuser`
   - Password: `passw0rd`

   Click **OK**.
3. Save the configuration.
4. Select **Resources -> JDBC Providers**.
5. Change the scope to the **ACompanyServer** in order to create the data source under the server. By default the scope is set to node.
6. Select: **DB2 JDBC Provider (XA)**. Select **Data Sources** at the bottom of the page.
7. Click **New** to create a new data source, then provide the following info:
   - Name: `redbookDS`
   - JNDI name: `jdbc/redbookDS`
   - Check **Use this Data Source in container managed persistence (CMP)**
   - Component-managed Authentication Alias: `dbuser`
   - Container-managed Authentication Alias: `dbuser`

   Click **OK**.
8. Select the newly created redbookDS entry, then select **Custom Properties** at the bottom of the page.
9. Select the databaseName entry and change the value to `REDBOOK`.
10. Save the configuration for WebSphere.
11. The next step is to install the ACompany application. Select **Applications -> Install New Application**.
12. Browse for the ACompanyBase.ear file, then click **Next**.
13. Navigate through the installation steps. Make sure that you check the **Deploy EJBs** box, and provide the database type `DB2UDB_V81` and the database schema name `ITSO`. Also make sure that the application is installed on the ACompanyServer application server.
14. Once the installation is done, save the configuration.
15. Install the process application. Select **Applications -> Install New Application**.
16. Browse for the ACompanyProcessBase.ear file, then click **Next**. Navigate through the installation steps. Make sure that the application is installed on the ACompanyServer application server.
17. Once the installation is done, save the configuration.

18. Select **Applications -> Enterprise Applications**. Check the box next to the ACompany and ACompanyProcess applications and click **Start**.
19. Once both applications are running, the sample application is ready to test. You can open the Default Business process Web client at:

    `http://localhost:9081/bpe/webclient`

Uninstall the base sample

If you want to uninstall the base sample application to deploy the extended sample application, or you just want to remove it from the application server, follow these steps:

1. Select **Applications -> Enterprise Applications**. Select the **ACompanyProcess** application.
2. Select the **Business Process Modules** link at the bottom, then select **ACompanyServices**.
3. Click **Templates**, check both templates (**CatalogUpdate**, **PO)** and stop them.
4. Go back to the applications list and stop the ACompany and ACompanyProcess applications.
5. Once they are stopped, you can select both and uninstall them.
6. Save the configuration for WebSphere.

Deploying the extended sample application

If you have the base sample installed, you will have to remove the applications from the application server. Follow the steps from the previous section to do that.

7. Install the ACompany extended application. Select **Applications -> Install New Application**.
8. Browse for the ACompanyExt.ear file, then click **Next**.
9. Navigate through the installation steps. Make sure that you check the **Deploy EJBs** box, and provide the database type `DB2UDB_V81` and the database schema name `ITSO`. Also make sure that the application is installed on the ACompanyServer application server.
10. Once the installation is done, save the configuration.
11. Install the process application. Select **Applications -> Install New Application**.
12. You can select to install the customized or the extended Process Web client.
    a. To install the customized Process Web client, browse for the ACompanyProcess.MyBpewebclient.ear file, then click **Next**. Navigate

through the installation steps. Make sure that the application is installed on the ACompanyServer application server.

For this scenario you have to install the MyBpewebclientEAR.ear application also.

b. To install the extended Process Web client, browse for the ACompanyProcess.bpewebclient.ear file, then click **Next**. Navigate through the installation steps. Make sure that the application is installed on the ACompanyServer application server.

13. Once the installation is done, save the configuration.

14. Extended Messaging configurations

    Follow the steps in 6.5.2, “Configuration with JMS Embedded Messaging” on page 259 to configure the application server for the Extended Messaging sample. Ifyou want to use the external WebSphere MQ messaging provider, refer to 6.5.3, “Configuration with WebSphere MQ as the JMS provider” on page 268.

15. Business Rule Bean configurations

    a. Create the Business Rule Beans database by following the steps in “Creating the Business Rule Beans database” on page 427.

    b. Deploy the Business Rule Beans application by following the steps in 10.6, “Deployment” on page 448.

       After starting the application server to setup the rule the ACompany application will fail to start. It is normal at this stage since the rest of the application is not configured yet.

    c. Run the Rule Management application by following the steps in 10.6.1, “Running the Rule Management Application” on page 449.

       Once the server is running the business rule has to be configured. Follow the steps in 10.3.3, “Creating and configuring the rule” on page 436.

16. Scheduler service configurations. The Startup Bean sample uses the Scheduler service, you have to configure the scheduler to make it work.

    a. Start the ACompanyServer application server if it is not running.

    b. Create a new work manager for the application by following the steps in “Work manager service in relation to the Scheduler service” on page 527.

    c. Create the Scheduler database for the service. Follow the instructions in “The Scheduler database” on page 528.

    d. Create a new Scheduler service by following the steps in “Scheduler configuration using the Administrative Console” on page 530.

17. Startup Bean configurations:
    a. Select **Applications -> Enterprise Applications**, and select the **ACompany** application.
    b. Change the Start weight to 2, which means that this application starts later than the ACompanyProcess application, so the resources from ACompanyProcess will be available for the Startup Bean.
    c. Save the configuration.
18. Dynamic Query configurations

    Follow the steps in 11.6, “Configuration” on page 482 to install the Dynamic Query application for the ACompanyServer.

    Also change the class loader policy for the server to SINGLE.
19. Configure the Object pool manager by following the steps in “Object pool manager configuration” on page 553.
20. Configure the Shared Work Area service by following the steps in 15.6.1, “Shared Work Area service configuration” on page 581.
21. Configure the Internationalization (I18N) service by following the instructions in 16.8, “Configuration” on page 622.
22. Once everything is configured for the extended sample, stop and start the server.

    Check the log file for the server to see if there was any error during the startup.

Deploy the Universal Test Client (optional)

The Universal Test Client (UTC) is part of WebSphere Studio. It is a very useful and handy application for testing purposes. If you want the same functionality in the runtime environment for the application server, follow these steps:

1. Copy the Universal Test Client enterprise application, IBMUTC.EAR, from the <*WebSphere_Studio_IE_root*>\wstools\eclipse\plugins\com.ibm.etools.utc_5.0.1 directory to the <*WebSphere_root*>\installableApps directory.
2. Start the application server, launch the Administrative Console, and log in.
3. Start installing a new Enterprise application (.ear) and select the **IBMUTC.ear** to install from the <*WebSphere_root*>\installableApps directory.
4. Go through the usual application installation steps.
5. Start the UTC application.

C

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246932

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, SG246932.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name *Description*

SG246932.zip	The sample application in a ZIP package (both base and extended versions are included).

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	20MB minimum
Operating System:	Windows 2000 with SP3 and the latest security updates
Processor:	Pentium® 4, 1GHz or faster
Memory:	512 MB minimum, 1GB preferred

How to use the Web material

Download the SG246932.zip file to a temporary directory. Create a directory for the additional material, for example: `C:\SG246932`, and unzip the contents of the Web material ZIP file into this folder.

For further instructions on how to use the sample application included in the additional material, refer to Appendix B, “Sample scenario” on page 665.

Abbreviations and acronyms

1PC	One-phase commit
2PC	Two-phase commit
AAT	Application Assembly Tool
ACID	Atomicity, Consistency, Isolation, Durability
AMC	Application / Module / Component
AMI	Application Managed Internationalization
API	Application Programming Interface
BMP	Bean Managed Persistency
BPE	Business Process Engine
BPEL4WS	Business Process Execution Language for Web Services
CMI	Container Managed Internationalization
CMP	Container Managed Persistency
CMR	Container Managed Relationship
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CSS	Cascading Style Sheet
DD	Deployment Descriptor
DMZ	Demilitarized Zone
EAR	Enterprise Archive
EIS	Enterprise Information Systems
EJB	Enterprise JavaBeans
EJB QL	EJB Query Language
EM	Extended Messaging
FAR	Flow Archive
FDML	Flow Definition Markup Language
FP	FixPack
GMT	Greenwich Mean Time
GUI	Graphical User Interface
HA	High Availability
HACMP	High Availability Cluster Multi-Processing
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IE	Integration Edition (WebSphere Studio)
ITSO	International Technical Support Organization
J2EE	Java 2 Enterprise Edition
JAR	Java Archive
JCA	Java Connector Architecture
JDBC	Java Database Connection
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JSP	JavaServer Pages
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MDB	Message-Driven Bean
MVC	Model-View-Controller
ND	Network Deployment
NLS	National Language Support
OASIS	Organization for the Advancement of Structured Information Standards

ORB	Object Request Broker
OS	Operating System
PME	Programming Model Extension
PMI	Performance Monitoring Interface
QL	Query Language
QoS	Quality of Service
RDB	Relational Database
RMI/IIOP	Remote Method Invocation / Internet Inter-ORB Protocol
RRA	Relational Resource Adapter
SDK	Software Development Kit
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSS	Staff Support Service
UDDI	Universal Description, Discovery and Integration
WAR	Web Archive
WIM	Work Item Manager
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
WSIF	Web Services Invocation Framework
XML	Extended Markup Language
XSLT	Extensible Stylesheet Language Transformations

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 689.

- *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series,* SG24-6195
- *WebSphere V5.0 Applications: Ensuring High Performance and Scalability,* SG24-6198
- *IBM WebSphere V5.0 Security WebSphere Handbook Series,* SG24-6573.
- *Exploring WebSphere Studio Application Developer Integration Edition 5.0,* SG24-6200
- *WebSphere Version 5 Web Services Handbook,* SG24-6891
- *WebSphere Application Server Enterprise Edition 4.0: A Programmer's Guide*, SG24-6504
- *Java Connectors for CICS: Featuring the J2EE Connector Architecture,* SG24-6401
- *WebSphere Enterprise: WebSphere Application Server Enterprise Edition 4.0: A Programmer's Guide*, SG24-6504
- *IBM WebSphere V5.0 Performance, Scalability and High Availability,* SG24-6198
- *WebSphere Application Server V5 for iSeries: Installation, Configuration, and Administration,* SG24-6588

Other resources

These publications are also relevant as further information sources:

- *DB2 Universal database for UNIX, Quick Beginnings V 7,* GC09-2970
- *DB2 Universal database for Windows, Quick Beginnings V 7,* GC09-2971
- *DB2 Universal database, Quick Beginnings for DB2 Servers V 8,* GC09-4836

- *DB2 Universal database, Quick Beginnings for DB2 Clients V 8,* GC09-4832
- *High Availability Cluster Multi-processing for AIX, Concepts and Facilities Guide V4.5*, SC23-4276-04
- *WebSphere MQ for Windows, V5.3 Quick Beginnings,* GC34-6073
- *WebSphere MQ for AIX, V5.3 Quick Beginnings,* GC34-6076
- *WebSphere MQ for Solaris, V5.3 Quick Beginnings,* GC34-6075
- *WebSphere MQ for HP-UX, V5.3 Quick Beginnings,* GC34-6077
- *WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3 Quick Beginnings,* GC34-6078

Referenced Web sites

These Web sites are also relevant as further information sources:

- WebSphere InfoCenter

  http://publib7b.boulder.ibm.com/webapp/wasinfo1/index.jsp?deployment=Enterprise&lang=en

- WSIF implementation

  http://cvs.apache.org/viewcvs.cgi/xylem-axis-wsif

- WSDD Process container architecture document

  http://www7b.boulder.ibm.com/wsdd/library/techarticles/wasid/WPC_Concepts/WPC_Concepts.html

- Process Choreographer Web client

  http://www7b.boulder.ibm.com/wsdd/zones/was/wpc.html

- Staff resolution architecture

  http://www7b.boulder.ibm.com/wsdd/zones/was/wpc.html

- (WSDD): WebSphere Application Server Enterprise Process Choreographer using Process Choreographer in a distributed environment

  http://www7b.boulder.ibm.com/wsdd/library/techarticles/wasid/WPC_UsingDist/WPC_UsingDist.html

- WebSphere MQ platform specific books:

  http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html

- Sun's JMS page

  http://java.sun.com/products/jms/index.html

- Extended Messaging tutorial on defered response

  http://www7b.software.ibm.com/wsdd/library/tutorials/0303_cox/cox_reg.html

- Amazon.com Web services

  http://associates.amazon.com/exec/panama/associates/ntg/browse/-/1067662/ref=gw_hp_ls_1_3/

- WebSphere Enterprise Scheduler API

  http://publib7b.boulder.ibm.com/wasinfo1/en/info/ee/javadoc/ee/com/ibm/websphere/scheduler/package-summary.html

- Web Services Technology preview

  http://www7b.software.ibm.com/wsdd/downloads/web_services.html

- Sun's J2C page

  http://java.sun.com/j2ee/connector/

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

Symbols

.project file 120
“flat” work area 564
“push down” technology 461
“type safe” notification 319

A

Abstract Schema Name 464
Access intent 21, 345, 641
  Assembly 367–368
  Bean level 346
  Create 366
  Dynamic Query 378
  Method level 347
  Read Ahead 360
  Studio 368
Access Intent Policy 345, 349
  Create 363
  Predefined 351
Access type 350
  Read 350
  Update 350
    Exclusive 350
    No Collision 350
    Weakest Lock at Load 350
Accessing PMEs 639
ACID properties 56
ACID transactions 396
Action commands 66
Activate At policy 407
Activities
  Empty 50
  Java snippets 49
  Process 49
  Receive event 49
  Service 48
  Staff 49
Activity ID 102
Activity state
  Claimed 102
  Finished 102
  Ready 102
ActivitySession 21, 396
  beginSession() method 405
  Boundary 398
  Client scoped life cycle 410
  Client side demarcation 410
  Configuration 398
  Coordination of one-phase commit resources 410
  endSession() method 405
  Exceptions 411
  Option A 408
  Option B 408
  Option C 408
  Option C+ 408
  Properties 398
  resetSession() method 405
  Resolution control 398
  Runtime 410
  Troubleshooting 411
  Unresolved action 399
  Usage Scenarios 410
ActivitySession attribute values 401
ActivitySession service API 404
Actors 27
acwa.jar 562
Add Access intent
  Studio 370
Add Breakpoint 149
Add client page 88
Adding libraries 75
Admin application 632
Admin service 630, 632
Admin UI 632
Administration 643
Administrative Console
  Install 644
Aggregation 460
AIID 102
Alarm 295, 321
Alarm development 324
AlarmListener 298, 305, 321
  fired() 298
  fired() method 323
AlarmManager 291, 294–295
AMC 464

AMI 587
AppBootstrapBean 307
Application callable receive 229
Application client test 550
Application Profile 355, 371
Application profiling 21, 346, 641
  API 349, 379
  Assembly 363
  Decision algorithm 347
  Sample flow 362
  Supporting application 379
  Test 381
  Tracing 381
Application server 630
  architecture 629
  cluster 180, 183, 185
  crash 503
  network deployed 179
  services 638
Application shutdown 503
Application start up 502
Application/Module/Component 464
Application-Managed Internationalization 587–588
appprofile-ee.jar 380
Architecture 628
Assembly
  Business Rule Beans 447
  Dynamic Query 478
  Dynamic Query Access Intent 480
  Internationalization service 599
  Object pools 543
  Scheduler service 516
  Startup Beans 493
Assign query 161
Asynchronizing applications 312
Asynchronous application 303
Asynchronous Beans 20, 641
  API 289
  Application Profile 300
  Base application 306
  Compare 299
  Datasource setup 329
  Deployment 331
  Development 312
  EJB cache code 308
  framework 288
  J2EE Component MetaData 300
  J2EE Contexts 301
  Java object 289
  List display code 309
  Programming interface 289
  Programming Model 300
  Quality of Service 333
  Resource references 292
  Sample 335
  Security 300, 302, 335
  Singleton code 307
  Test 325
  Test server 325
  Three types 297
  Transactions 299, 301
  Work development 313
Asynchronous patterns 305
Asynchronous search 337
  Flow 337
Asynchronous service invocation 54
Asynchronous Work test 316
AsynchScope 291, 294
Attach nodes 203
Attach to Process Engine 151
Audit trail 225

B

Background processing 303, 305
Base application
  Test server 671
Base rule 426
Base sample
  Deployment 677
  import 667
  Uninstall 679
Basic configuration 647
Bean cache policy 407
BeanTaskInfo interface 512
Blocks 50
BPE
  Artifacts 216
  Availability 177
  Class loader issues 637
  Client settings 88
  Configuration file 117
  Debugging 148
  Error messages 224
  External interfaces 174
  Factory 174
  Internal interfaces 174
  Management 216

Navigator 173
People interaction 174
Process debugger 149
Scalability 177
Staff repositories 176
Staff resolution plug-ins 176
Staff support 158
Staff support service 176
Supporting database 637
Testing 148
Tracing 224
ViewContext 107
Web client 175
Work item manager 175
BPE API
Processes 62
BPE architecture 172
BPE Java client 132
BPE Java client libraries 139
BPE servlet client 140
BPE Web client 71
API 72
Customization 73
Customize 71
Customizing the content 123
Customizing the header 122
Customizing the layout 122
JSP fragments 84
Look and feel 71
Process information 72
Process Input page 83
Process output page 82
Register custom JSPs 88
Security role 119
Splash screen 121
Staff activity 72
BPEContainer 118
bpecontainer_ejb 118
BPEL 60
BPEL4WS 60
BPERemoteDeploy 118
bperemotedeploy_ejb 118
bpesoapclient 118
bpeWebclient 118, 471
copy 118
BRBeansDB2.jar 427
brbeansDefaultProperties 430
brbPropertiesFile 450
Built in query sample 469
Business Analyst 421
Business logic 435
Business Process Container 635
Install Wizard 675
Supporting database 676
Business process engine 42
Concurrency 43
Heterogeneous execution 43
Quality of Service 43
Recoverability 43
Business Process Execution Language for Web Services 60
Business process-based applications 42
Business processes 36
Business Rule Beans 21, 641
Architecture 424
Assembly 447
Base rule 426
brbeansDefaultProperties 430
Classifier rule 425
Client component 424
Client-side Caching 451
Components 424
Configure rule 436
Create folder 437
Create project 426
Create rule 436–437
Creating database indexes 454
Database 427
Deployment 448
Deployment code 427
Development 426
Development process 426
EJBs 423
Flow 424
framework 420, 423
Java libraries 427
New folder 437
New rule 437
Non-classifier rule 426
Persistence 425
Precedence 439
Process flow 442
Rule 425, 427
RuleFolder 427
RuleImplementor 423, 425
Sample 422, 441
Sample flow 442
Scenario 422

Security 455
Test environment 432, 444
Test sample 446
Trigger Point 423
TriggerPoint object 424
VM arguments 429
XML 424
Business rules 420
Business scenario 26

C

Calendar
Resource environment reference 524
Caller identity 346
Caller Internationalization context 594
Cascading Style Sheets 117
Cell 643
Cell configuration 203
Class loader 636
issues 637
policy 477, 483
Classifier rule 425
Cloning 647–648
CMI 587
CMM 228
CMR 351
Code assist 119
Collection Increment 350
Access Intent 359
Collection increment 359
Collection Scope 350
ActivitySession 351
Transaction 350
Compare Asynchronous Beans 299
compensate_ejb 118
Compensation 56
Concurrency Control 349
Configuration
ActivitySession 398
Class loader policy 483
Dynamic Query 482
Extended Messaging 250
Internationalization service 622
Last Participant Support 390
Object pools 552
Process cache settings 68
Process commands 68
Process view 68
Scheduler service 511, 527, 530
Shared Work Area service 580
Startup Beans 502
WebSphere 664
Configuration repository 629
Configure Test server 671
Connection pooling 631
Container ActivitySession 401
Container internationalization attribute 601
Container Managed Messaging 228
Container Managed Relationship 351
Container Tasks 372
Container Transactions 372
Container-Managed Internationalization 587, 589
Context root 74
Contextual information 560
Controller 64
CORBA C++ SDK 23
Country code 590
Create Application profile 372
Create Data source 448, 678
Create new JSP 84
Create Rule implementor 435
Create scheduler database 529
Create Sender bean 236
Create service proxy 497
Create servlet 143
Create startup session bean 498
Create task session EJB 518
Create Test server 670
Create work area 565
createFastPool() method 540
Cron calendar 167, 514
CSS 117
Custom installation 655
Custom object pool 541, 553
Custom Staff Query Verbs 163
Customer loyalty scenario 422
Customization
BPE Web client 73

D

Data access patterns 358
Data dictionary 78
Data integrity 363
Data mapping 230
Database connection 631
Database lock monitor 357

DB2_JDBC_DRIVER_PATH 194
Dead beats 297
Deadlock 356
Debugging 148
  Step-by-step mode 154
Default access intent 346
Default user calendar 514
Deferred response 229
Define Data source 311
Define local EJB reference 145
Define notification 513
Define task 512
Define user calendar 513
DenyAllRole role 455
Dependent Rules 439
Deploy process 218, 672
Deployment
  Business Rule Beans 448
  Dynamic Query 483
  Internationalization service 623
  Scheduler service 531
  Startup Beans 502
  Universal Test Client 681
Deployment Manager node 654, 661
Design
  Internationalization service 586
  Object pools 539
  Scheduler service 510
  Shared Work Area service 561
  Startup Beans 490
Develop AlarmListener 321
Development
  Internationalization service 591
  Object pools 539
  Scheduler service 511
  Shared Work Area service 562
  Startup Beans 491, 496
Development Environment Setup 426
Different locales 585
Different time zones 585
Display commands 65
distexcep.jar 562
Domain Expert 421
Dynamic Query 21, 458, 641
  Access Intent 378, 480
  Aggregate functions 459
  API 463
  Assembly 478
  Bean business methods 459
benefits 460
bpeWebclient 470
Business methods 459
Calendar comparisons 458
Client implementation 465
Configuration 482
Delimited identifiers 458
Dependent Value methods 459
Deployment 483
Design 462
Development 463
Development environment 467
DISTINCT predicate 458
Dynamic query execution 459
Example 460
EXISTS predicate 458
Financial applications 460
GROUP BY, HAVING 459
Import supporting application 467
Install query.ear 482
Interfaces 463
Java libraries 463
JNDI name 463, 471
JSP 472, 474
Local client programming model 465
Locking 484
Multiple element select 459
Order by 458
parameterVars 464
Performance 462, 484
Process Choreographer integration 470
Query over inheritance graph 458
queryDomain 464
queryStatement 464
Remote client programming model 466
Results JSP 474
Return multiple fields 460
Sample client 467
Sample deployment 468
Sample flow 468
Sample integration 475, 481
Sample query client 466
Scalar functions 458
Scenario 460
Security 486
Select clause 458
service 458
Servlet 474
SQL date/time expressions 458

String comparisons 458
Subqueries 458
Test environment 476
Test sample 477
Transactions 484
Dynamic Query Bean 464
Dynamic SQL 458
Dynamic tuning 334
Dynamically listen for JMS queues 303

E

EAAT0046I 377
eBank sample 363
Editing process 222
EE Test Environment 148
efixes 660
efixWizard.bat 645
EJB 344, 631
EJB Access Intent 480
EJB container 344, 630–631
EJB custom finder 307
EJB life cycle 396, 406
EJB method authorization 462
EJB module 631
re-start 503
EJB performance 344
EJB QL 458
EJB RDB mapping 307
EJB references restriction 146
EM 228
E-mail interfaces 631
Embedded HTTP server 630
Embedded JMS Server 633
Embedded Messaging 258
Empty activities 50
Enable ActivitySession service 411
Enhanced Internationalization Service 596, 624
Install 624
Enterprise architecture 634
Enterprise Extensions 628, 659
Enterprise Java Beans 631
Event 293
Event driven applications 302
Event source
Reference 293
Event style receive 229
EventListener 299, 305, 320
EventSources 291, 293, 319
Everybody staff plugin 210
executePlan() method 465
executeQuery() method 464
Existing base product 654
Export 645
Export adminconsole 645
Export EAR 479
Export project 479
Extended Local Transactions 397
Containment 397
Enlistment 397
Extended messaging 20, 228, 640
code 241
Configuration 250
Late responses 278
Port 231
Receiver bean 243
Transactions 276
Wizards 233
Extended sample application 669
Configuration 672
Deployment 679
Extensibility 628
Externalize rule processing 420

F

Facades 63
FAR 208, 636
FDML 59
Feature dependencies 656, 659
Features to install 655
findByPrimaryKey() method 351, 361
Fire and forget 229
Flow archive 208, 636
flushPool() method 541
Forward operations 58

G

Gather security roles 434
Generated code 241, 247
Generic rules library 423
getObject() method 540
getPool() method 540
Global Transaction 388
Global Transaction scenario 388
GMT 584
Greenwich Mean Time 584

H

HACMP 185
Handling responses 279
Heuristic condition 394
Heuristic reporting 394–395
High availability 647
High Availability Cluster Multi-processing 185
High-availability configuration 185
Horizontal scaling 182, 648
HTTP protocol 630
HTTP Server 630
HTTP Server plug-in 630
HTTPRequest 471

I

i18n 584
I18nUtils 620
IBM Agent Controller 148
IBM HTTP Server 654
IBM JDK 140
Implementation 320
Import 669
Install 192, 202
  Administrative Console 644
  Database server 202, 205
  Existing base 658
  Existing Network Deployment 661
  Features 655
  Network Deployment 205
  query.ear 482
  Troubleshooting 663
  Verify 662
  Web server 204
  WebSphere MQ server 205
  WebSphere Network Deployment 202
Installation 654
  Steps 655
  Topologies 177
  Verify 661
  WebSphere 654
interface 435
interim fixes 645, 660
internal JMS server 266
Internationalization 527
Internationalization attribute 589
  EJBs 601
  Servlet 603
Internationalization context 586
  API 591
Internationalization interface 592
Internationalization service 23, 642
  Application client test 617
  Assembly 599
  Caller context 587
  Configuration 622
  Deployment 623
  Design 586
  Develop sample 606
  Development 591
  Invocation context 587
  NameNotFoundException 623
  Resource environment reference 610
  Run as Caller 589
  Run as Server 589
  Run as Specified 589
  Sample application 605
  Sample application client 606
  Sample details 620
  Sample Web client 618
  Scenarios 584
  Test environment 598
  Tracing 623
  Traditional solutions 585
  Web Services 596
Internationalization type 588, 599
  EJB 599
  Servlet 600
internationalized 584
Interruptible processes 52
Intra-Application Notification 305
Intra-process load balancing 183
Invocation internationalization context 595
InvocationInternationalization interface 592
Invoke RMLT 387

J

J2C 19
J2C Container 630, 632
J2EE
  Asynchronous 303
J2EE client
  Run 674
JAAS Authentication Entries 433
Java
  Swing 134
  Visual Editor 133

Java 2 Connectors 19
Java application
Run 428
WebSphere Studio 428
Java Connector Architecture 632
Java GUI application 133
Java Message Service 228
Java process debugger 153
Java SDK 586
Java snippets 49
Java Transaction Service 395
Java Virtual Machine 538, 630
JCA 632
JMS 228
JMSAPIUser Role 216
JNDI 630
JNDI Explorer 444
JSP 631
JSP development 471
JTA Extensions 412
JTS 395
JVM 538, 630

K

key-value-mode triplet 561

L

Language code 590
Last Participant Support 21, 389
Configuration 390
Tracing 392
Troubleshooting 392
Late binding values 161
Late response extension 283
Late responses 278–279
launchClient 138, 617
LDAP schema 211
LDAP staff plugin 163
Configuration 213
LDAP staff plugin provider 210
LDAP user registry 666
License agreement 655
Listener port 282
Load At policy 407
Load point 359
Local Transaction Containment 387
Locale 586
Locales field 590
lock conversion 353
lock escalation 353
Loops 50
Lowest level of locking possible 484
LPS 389
LTC 387
LTC scenario 387

M

macroflow 52
Manage tasks 515
Manage WebSphere Variables 189
Managed process 629
Managing processes 216
MDB 228
Message formats 228
Message Listener Service 265
Message Sending Pattern 231
Message-Driven Beans 228, 633
MessageTaskInfo interface 512
Messaging interfaces 631
Messaging output port 263
Messaging Patterns 230
Messaging Receiving Pattern 232
Metadata 461
microflows 51
Model 64
Model-View-Controller 64
MQ messaging 258
MQJMS_LIB_ROOT 195
Multinode environment 654, 661
Multiple WorkManagers 334
MVC 64

N

Name server 633
Name service 630
NameNotFound 284
Nested work areas 564
Network Deployment 180, 661
New Access Intent Policy 365
New Web project 74
NLS 224
Node 629
Node Agent 629
Non-classifier rule 426
Non-interruptible process 51
non-JMS middleware 303

Notification 513
Notification action 513
Notification framework 319
Notification mask 513
NumberTimeinterval 515

O

OASIS 60
Object pool manager 540, 553
Object pools 22, 538, 540, 642
  Add a resource environment reference 546
  Advantages 538
  API 540
  Application client development 545
  Assembly 543
  Configuration 552
  Design 539
  Development 539
  Disable 552
  Find 542
  init() method 541
  JNDI 542
  Performance 555
  Resource environment reference 543
  returned() method 541
  Runtime 554
  Sample 544
  Specifed class 542
  Synchronized pool 539
  Test 550
  Test environment 543
  Test results 552
  Tracing 555
  Unsynchronized pool 539
  Workload management and failover 555
Object query statement 461
Object Request broker 630
Object usage frequency 539
objectpool.jar 540
Open Business process 79
Open message part types 80
Open process file 93
Optimistic concurrency 353
Optimistic concurrency control 349
OptimisticRead 357
ORB 630
Organization for the Advancement of Structured Information Standards 60

P

Parallel search 335
Parallel task 303
Parameterized verb 160
Partition tasks 303
Pass data between application components 560
Performance
  Dynamic Query 484
  Object pools 555
  Scheduler service 534
Performance and analysis tool 654
Performance Monitoring Interface 631
Performance Monitoring Service 534
Performance report 354
Persistence Manager 344
Persistence Manager cache 359
Pessimistic concurrency control 349
PessimisticUpdate 356–358
Phantom reads 353
PIDD 98
PME 638
PoolableObject 541
Potential owner 159
Predicate clause 459
prepare commit 394
prepareQuery() method 465
Process activities 49
Process choreographer 20, 640
  Activities 48, 63
  API JNDI 131
  API Local interface 131, 140
  API Remote interface 131
  Audit trail 225
  Blocks 50
  Command implementation 67
  Compare WebSphere MQ Workflow 39
  Compensation 56
  Composition model 45
  Configuration file 67
  Data source configuration 193
  Default JSPs 68
  Deploy Process 218
  EJB façade 64
  Empty activities 50
  External API 60
  General API 61
  Information Model 44
  Installation wizard 193
  J2EE approach 42

J2EE programming model 45
J2EE security roles 215
Java snippets 49
JMS provider configuration 195
Loops 50
Managment 216
MDB façade 63
Modelling language 59
Organization model 45
Parallel activities 47
Process activities 49
Process engine 38
Process model 47
Process template 47
Programming 42
Programming model 44
Receive event activities 49
Security 213
Sequential activities 47
Service activities 48
Services model 44
Staff activities 49
Tracing 224
Transaction boundaries 53
Using 38
Variables 50
Web client 64
Work items 63
Process Choreographer API 128
Process container
Architecture 172
Clustering 206
Database setup 199
Install 186
Resources 186
Runtime topologies 176
Security configuration 196
Software components 188
WebSphere MQ setup 201
Process Debug Perspective 150
Process debugger 149
Java Debugger 153
WebSphere Application Server 158
Process deployment 218
Process EJB module 636
Process ID 98
Process Instance Name Panel 123
Process life cycle 56
Process modeling language 59
Process navigation 172
Plug-ins 173
Process Security Context 214
Process template 221
Process versioning 219
Process Web client 636
ProcessAdministrator Role 215
Process-Level Staff Roles 162
Process-specific API 63
Profiling Levels 344
Profiling priorization 347
Programming Model Extensions 638
Overview 19

Q

QoS 628
Qualities-of-Service 628
Query bean 461
Query Engine 462
Query enhancements 458
Query projection 459
Query Result Caching 213
Query support 458
query.ear 476, 482
QueryBean 464
QueryClient.jar 463
QueryIterator() method 464
Queue destinations 261, 271
QueueConnectionFactory 259, 269

R

Read Ahead 351, 360, 377
Receive event activities 49
Receiver bean 228, 243
Code 247
Redbooks Web site 689
Contact us xix
Registration 320
Relational Resource Adapter 344
Repeatable read 353
Repository 632
Resource Manager 387
Prefetch Increment 351
Resource Manager Local Transaction 386
Resource reference
Work 317
returnObject() method 540
RMLT 386

Role-based permissions 159
RRA 344
Rule
  Configuration 436
  Unit test 444
Rule client 440
  Create 440
Rule Firing Location 453
  Anywhere 453
  Configuration 454
  Local 453
  Remote 453
Rule Implementation Developer 421
Rule implementor
  Create 435
Rule loader 424
Rule maintenance 421
Rule Management 421
Rule Management Application 421, 423, 428, 436
  Configuration 428
  Run 449
  Tasks 436
  WebSphere Studio 428
Rule Management tool 421
Rule object 425
Rule properties 437
Rule settings 437
Rule state
  Expired 440
  In effect 440
  Invalid 440
  Schedules 440
  Unavailable 440
Rule-based Application Component Developer 421
RuleImplementor 425, 435
  fire() 435
  getDescription() 435
  init() 435
RuleManager role 434, 455
rulemgmt 450
Rules states 440
RuleUser role 434, 455
Running Java application 428
Runtime architecture 628
Runtime environment
  Sample configuration 675
  Startup Beans 502

S

Sample
  Activity input page 100
  Activity output map 104
  Activity output page 102
  Activity pages 99
  Application profiling 362
  Approval pages 94
  Asynchronous Beans 335
  Asynchronous Beans output 332
  Asynchronous Beans test 311
  Asynchronous search servlet 338
  Base application 306
  BPE Java client 132
  BPE servlet client 140
  Business Rule Bean 441, 672, 680
  Business Rule Beans flow 442
  CMP Cache 304
  Configuration 310, 341
  Configuring activity pages 106
  Customized activity input page 111
  Customized activity output page 113, 115
  Customized JSPs 92
  Customized process information input page 109
  Customized process input page 107
  Customizing BPE JSPs 79
  Database 666
  Database configuration 310
  Development environment 667
  Dynamic Query 468, 475, 673, 681
  eBank application 363
  Extended messaging 672, 680
  Generate EJB deploy code 669
  Import 306
  Import to Studio 667
  Input request page 94
  Interaction diagram 304
  Internationalization service 674, 681
  Look and feel 117
  Object pool 544, 673, 681
  Parallel Search 337
  Predefined queries 469
  Process error page 87
  Process information page 98
  Process input map 96
  Process input mapping 85
  Process input page 84, 95, 123
  Process output message page 90

Process page 95
Rebuild projects 669
Runtime environment 675
Scheduler service 517, 673, 680
Shared Work Area service 674, 681
Singleton 304
Startup Bean 304, 315, 673, 681
Test BPE servlet client 146
Test environment configuration 670
Test server configuration 310
Testing BPE Java client 137
User registry 666
Sample application 666
Sample business processes 29
Sample code
EMS 229
Sample configuration 496
Sample JSP 471
Sample process
Generate deploy code 668
Sample scenario 26
Extended messaging 234
Sample servlet 473
Scalability 647
Startup Beans 506
Schedule a Task 515
Scheduler API 511
Scheduler Calendar JNDI name 166
Scheduler database 528
Scheduler service 22, 510, 642
Alarm thread 531
Assembly 516
Configuration 511, 527
Deployment 531
Design 510
Development 511
Disable 531
Example 510
Find 515
JNDI 514
Locking mechanism 532
Performance 534
Runtime 531
Sample scenario 517
Scenario 510
Security 536
Supporting database 528
Test environment 516
Tracing 532
Wakeup daemon 531
Work Manager service 527
Scheduler service enablement 520
schedulerclient.jar 517
SchedulerProxy bean 610
Scripting client 633
Security
Business Rule Beans 455
Dynamic Query 486
executeQuery() 486
Extended messaging 285
Scheduler service 536
Startup Beans 492
Security context 214
Security server 633
Security service 630
Sender bean 236
Code 241
Service activities 48
service oriented architecture 16
Servlet 631
Servlet development 473
Session management 631
Set of alarms 297
setProperties() method 541
Shared Work Area 22, 642
Sample scenario 560
Shared Work Area service 560, 584
API 563
Application client 568
Change property mode 567
Characteristics 561
Client properties 581
Configuration 580
Design 561
Development 562
Find 564
Get properties 566
JNDI 564
Modify properties 564, 566
Pervasiveness 562
PropertyModeType 563
Remove properties 567
Sample 568
Size 562
Test 577
Test environment 568
Tracing 582
user defined properties 565

Sharing object pools 555
Simple arithmetic calendar 515
SIMPLE calendar 167
Simple Object Access Protocol 634
SimpleTimeZone 586
Single machine 178
SOA 16
SOAP 634
Special Enterprise Archive 636
Specific finders 461
Splash screen 121
SQL statement 461
SQLJ 462
Staff activity 49, 54
  Assign query 161
  Global Variables 164
Staff Activity Data 166
Staff Activity Duration 166
Staff activity roles 159
Staff plugin 208
Staff plugin provider
  Configuration 208
Staff queries 159
Staff resolution plugin 208
Staff settings 208
Staff support 158
Stale beats 297
Stand-alone application server 178
Stand-alone topology 192
Start WebSphere Studio 667
Starter role 163
Starting process template 221
Startup Beans 22, 490, 642
  Assembly 493
  Configuration 502
  Default priority 505
  Deployment 502
  Design 490
  Development 491, 496
  Example 490
  Home interface 491
  Implementation 500
  Priorites 493
  priorities 504
  Remote interface 491
  Runtime environment 502
  Sample 496
  Scalability 506
  Security 492
  Security identity 493
  start() method 315, 492, 501
  stop() method 492
  Test environment 493
  Tracing 506
  Transactional properties 495
  Transactions 492
  Using 491
Startup service runtime flow 502
startupbean.jar 497
Stopping process template 221
Stopping templates 679
Stratified transactions 52
Studio
  Import sample 667
  J2EE clients 674
SubsystemMonitor 291, 296
SubSystemMonitorManager 294
Supporting database 199, 640
Swing GUI 134
Synchronized object pools 554
Synchronous response 229
Synchronous service invocation 53
System staff plugin 210
System staff plugin provider 210

T

Table prefix 529
Task 512
  Cancel 515
  Partition 303
  Permanently delete 515
  status 515
  Suspend 515
Task handler bean implementation 519
Task information 512
Task target action 512
Terminate work area 567
Test
  Business Rule Beans 446
Test environment
  Business Rule Beans 444
  Configuration 432, 476, 670
  Dynamic Query 476
  Internationalization service 598
  Object pools 543
  Scheduler service 516
  Shared Work Area service 568

Start 137
Startup Beans 493
Test server 433
Data source 433
Testing 148
Time zone ID 590
Time-Based activity 305
TimeZone 586
Topology 178–180
Trace service 630
Tracing
Application profiling 381
Last Participant Support 392
Transaction
Compensation 56
Hazard 390
Phase one 388
Phase two 389
XA 56
Transaction isolation level 352
Transaction management 631
Transaction Manager 388
Transaction termination 466
Transactional services 641
Samples 413
Transactions
Dynamic Query 484
Startup Beans 492
Transient activity 305
Trigger point 420
Troubleshooting
ActivitySession 411
Last Participant Support 392
Two-phase commit 388
Type safe notification 293
Typical installation 655

U

UDDI 634
Umbrella installation 654–655
Undo 56
Undo operations 58
Uninstall process 222
unit of work scope 387
Unit test
Extended messaging 249
Universal Description, Discovery and Integration 634
Universal Test Client 254, 444
Deployment 681
Unsynchronized object pools 555
Updating process 221
Use cases 26
Use work area 565
User calendar 513
User defined calendar 522
User registry 666
User Registry staff plugin provider 210
UserCalendar 167
UserInternationalization interface 591
UserWorkArea 563
Using Internationalization service 594
Using Object pools 542
Using Scheduler service 511
Using Shared Work Area service 564
Using Startup Beans 491
UTC 681

V

Value object methods 460
Variables 50
VerbSet.xml 163
Verify installation 658
Versioning processes 219
Vertical scaling 181, 647
View 64
Virtual host 631
Visual Editor 133

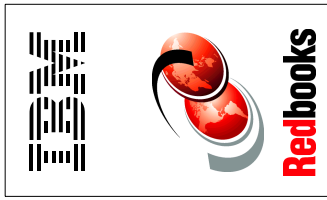
W

Wakeup daemon thread 531
wasStartupPriority 504
wasStartupPriority property 502
WeakestLockAtLoad 355
Web client
ViewContext 107
Web container 630–631
Web Deployment Descriptor 143
Editing the source 146
Web server plug-ins 654
Web services
Architecture 17
Web Services Description Language 634
Web services engine 633
Web Services Invocation Framework 634
WebClientUser Role 216

WebSphere
Configuration 664
Installation 654
WebSphere Application Server
Process debugger 158
WebSphere Application Server Enterprise 634
Architecture 634
cell 643
WebSphere Application Server V5 base 629
WebSphere cell 643
WebSphere Client 138
WebSphere Embedded Messaging 258
WebSphere Enterprise
cell 643
features 655
Installation 654
WebSphere environment variables 189
WebSphere JMS Provider 259, 633
WebSphere JVM settings 449
WebSphere MQ 183
Clustering 183
Messaging 258
messaging provider 268
Workflow 36
Compare Process Choreographer 39
WebSphere plug-in 630
WebSphere Studio
Visual Editor 133
WebSphere Variables 675
WebSphereTrader 303
Welcome Pages 122
Work 291, 298
Amazon Search 340
Background processing 305
Development 313
EJB search 339
File Search 340
JDBC Search 340
release() 298
run() method 298, 314
Work Area property
Fixed 563
Fixed read-only 563
Normal 563
Read-only 563
Work Item Manager 159
Work Manager
create() 293
doWork() 292
join() 293
startWork() 292
Work manager service 527
Workflow 36
Workload management 646
WorkManager 291
Reference 292
Tuning 334
WorkManager configuration 330
WorkManager reference 328
wsadmin 633
WSDL 17, 634
wsexception.jar 620
WSFL 59
WSIF 18, 634
wsOptimisticRead 352
wsOptimisticUpdate 352
wsPessimisticRead 351
wsPessimisticUpdate 351, 355
wsPessimisticUpdate-Exclusive 351
wsPessimisticUpdate-noCollision 352
wsPessimisticUpdate-weakestLockAtLoad 352

X

XA capable resources 395
XA resource 56
XA standard 56
XAResource 388
XML files 632



WebSphere Application Server Enterprise V5 and Programming Model Extensions

WebSphere Handbook Series

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

WebSphere Application Server Enterprise V5 and Programming Model Extensions

WebSphere Handbook Series

Programming Model Extensions in WebSphere Enterprise V5

Sample application for each PME

WebSphere Enterprise V5 Runtime

This IBM Redbook provides system administrators, developers and architects with the knowledge needed to implement WebSphere Application Server V5.0 Enterprise runtime environment, to design, develop, assemble and deploy enterprise applications, and to perform ongoing managmement of the WebSphere environment.

Part 1, “Introduction” explains how the book is organized to cover the WebSphere Enterprise product. It helps you to understand the product line for planning. This part gives a broad description of the sample scenario used for the book on the business requirements level.

Part 2, “Programming Model Extensions” is the major part of the book. It covers all the Programming Model Extensions for WebSphere Application Server V5.0 Enterprise. Each extension is discussed in its own chapter starting with planning and design, through development and deployment to the configuration and administration. The book follows the J2EE roles and actions to be taken in an end-to-end solution design.

The Appendixes give detailed steps for installing and configuring WebSphere Application Server V5.0 Enterprise.There are also step-by-step instructions for configuring and deploying the sample application that is shipped with the book.

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

SG24-6932-00 ISBN 0738428493