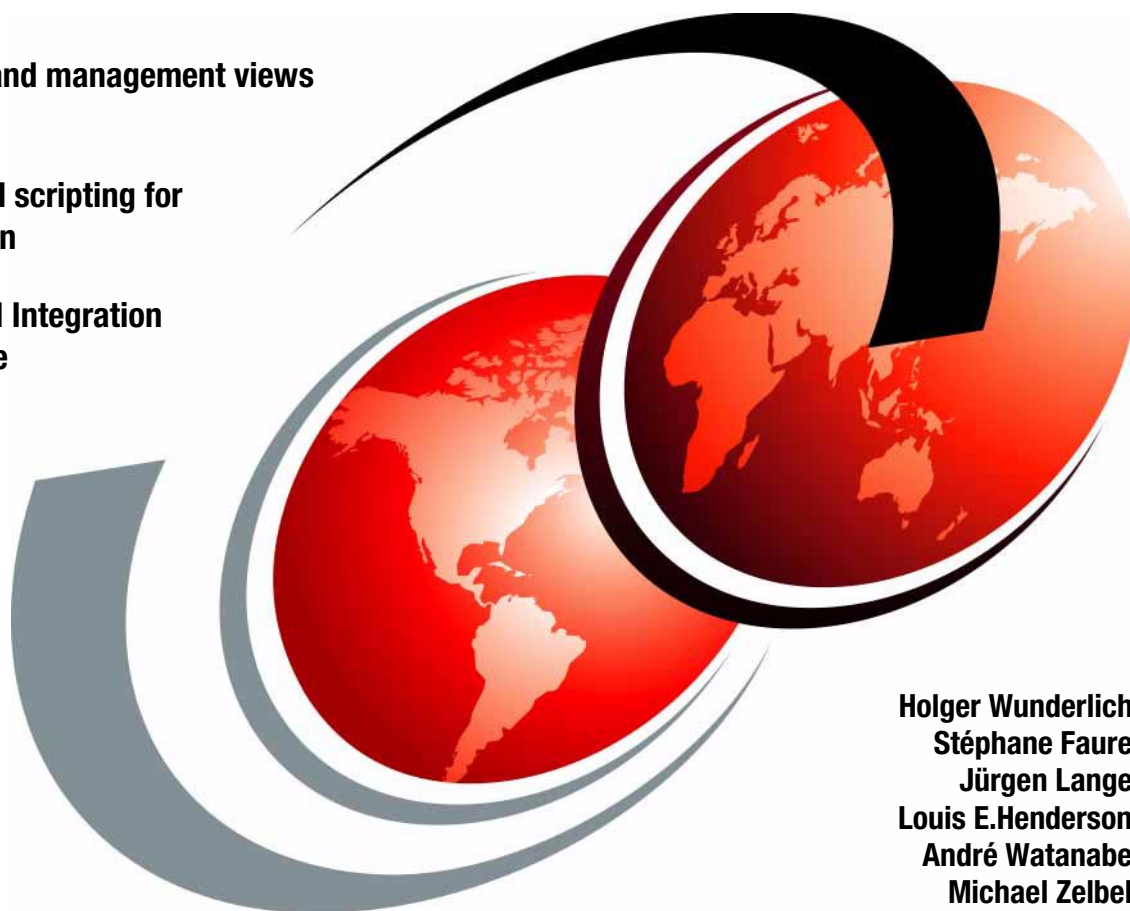


Migrating WebSphere Applications to z/OS

Technical and management views

Tooling and scripting for optimization

Operational Integration and Change



Holger Wunderlich
Stéphane Faure
Jürgen Lange
Louis E. Henderson
André Watanabe
Michael Zelbel



International Technical Support Organization

Migrating WebSphere Applications to z/OS

April 2002

Take Note! Before using this information and the product it supports, be sure to read the general information in “Notices” on page ix.

First Edition (April 2002)

This edition applies to IBM WebSphere Application Server for z/OS Version 4, for use with the z/OS Operating System.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2002. All rights reserved.

Note to U.S Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
Preface	xi
The team that wrote this redbook	xii
Who should read this book	xiv
Notice	xv
Comments welcome	xv
Part 1. Migrating WebSphere applications	1
Chapter 1. Introduction	3
1.1 Overview	4
1.2 Quick introduction to z/OS vocabulary	7
1.3 EJB roles versus job roles in the WebSphere world	8
1.3.1 Infrastructure roles	9
1.3.2 Development job roles	9
1.3.3 Implementation job roles	10
1.3.4 Skills needed during an application's life	12
1.4 WebSphere architectures and topologies	14
1.4.1 Blueprint of a multi-tiered architecture	14
1.4.2 The individual tiers	15
1.4.3 Two-tier applications	16
1.4.4 Multi-tier applications	16
1.4.5 "Common" practice: 3-tier applications	17
1.4.6 Multi-tier minus one	18
1.4.7 Moving the business tier to z/OS	18
1.4.8 Security model with IBM WebSphere Application Server for z/OS and OS/390	19
1.5 System landscape	20
1.5.1 Development system	20
1.5.2 Test system	21
1.5.3 Verification or quality assurance system	22
1.5.4 Production system	22
Chapter 2. WebSphere sample performance benchmark application ...	23
2.1 Environment and software levels	24
2.1.1 Development environment	24
2.1.2 WebSphere Application Server distributed test environment	24

2.1.3	z/OS environment	24
2.2	The sample application	25
2.2.1	WebSphere sample performance benchmark application	25
2.2.2	Trade2 topology and architecture	25
2.2.3	Outline of our migration task	30
Chapter 3.	Preparing for WebSphere application migration	31
3.1	Application analysis	32
3.2	Setting up the migration infrastructure	34
3.2.1	The integrated development environment.	36
3.2.2	The distributed Deploy- and Runtime-Environment	38
3.3	Installing the application.	39
3.3.1	Installing the original WebSphere sample performance benchmark application.	39
3.3.2	Deployment process from WebSphere Studio Application Developer	40
Chapter 4.	Migrating an application to z/OS	45
4.1	Minimum migration steps for Trade2	46
4.1.1	Overview	46
4.1.2	Install the Application Assembly Tool (AAT) for z/OS	47
4.1.3	Install the System Management Enhanced User Interface	47
4.1.4	Import the .ear file and the helper classes	47
4.1.5	Update the EJB links.	48
4.1.6	Add initial values where needed	48
4.1.7	Generate the new .ear file.	49
4.1.8	Set up the application database tables on z/OS	49
4.1.9	Deploy to WebSphere Application Server on z/OS.	50
4.1.10	Activate the application on z/OS	51
4.2	Common migration steps	51
4.2.1	Migration check list	51
4.2.2	Packaging the application	52
4.2.3	JNDI and namespace fundamentals	53
4.2.4	Initial Context factory.	62
4.2.5	Externalizing access to non-J2EE connectors	63
4.2.6	Remapping CMP beans to z/OS DB2.	63
4.2.7	Changing SQL commands	64
4.2.8	JCA connector issues	64
4.2.9	Framework and connector issues	65
4.2.10	Restrictions of the z/OS EJB runtime	66
4.3	Common problems in the migration phase	66
4.3.1	JNDI problems	66
4.3.2	Classloader problems	68
4.4	Code optimization	70

4.4.1 Absolute vs. relative path	70
4.4.2 Avoid specialties.	71
4.4.3 Use WebSphere connection pooling.	71
4.4.4 Read properties files from a .jar file	71
4.4.5 Use javax.sql.DataSource for JDBC connection.	71
4.4.6 Put supporting .jar files into the .ear files root.	71
4.4.7 Transaction attributes for Session EJBs	72
4.5 Code migration hints and tips	72
4.5.1 ASCII/EBCDIC issues.	72
4.5.2 DB2 UDB issues	74
4.5.3 Redeployment using AAT for z/OS	75
4.5.4 Handling minor changes during the testing phase	75
4.6 Final steps	76
Part 2. Managing WebSphere applications	79
Chapter 5. The lifecycle of an application.	81
5.1 Introduction	82
5.2 Setting up J2EE servers on z/OS	85
5.2.1 Common steps for J2EE creation	86
5.2.2 Setting up a J2EE test server	97
5.2.3 Setting up a production J2EE server.	98
5.2.4 Setting up WLM classification	98
5.3 From test to production	104
5.3.1 Our test and production environment	105
5.3.2 Scripted configuration	105
5.3.3 Scripted application fallback	107
5.3.4 Production infrastructure	112
5.3.5 Automatic application verification	113
5.4 Nondisruptive upgrade of an application.	117
5.4.1 HTTP access-based applications	118
5.4.2 RMI/IIOP access-based applications	119
5.5 Disruptive upgrade of WSAS runtime	119
5.6 Nondisruptive upgrade of the WSAS runtime	120
5.7 Change and problem management.	120
5.7.1 Deployment principles.	121
5.7.2 Recommendations for a controlled deployment	123
5.7.3 Deployment integration into change management	125
5.7.4 Problems not addressed (and directions for future tooling)	127
Chapter 6. Management summary.	129
6.1 Operational view	130
6.2 Technical view	130
6.3 Total cost of ownership vs. cost of migration	131

Part 3. Tools for WebSphere Application Server on z/OS	133
Chapter 7. Jinsight	135
7.1 Overview	136
7.2 Jinsight 2.1	137
7.2.1 Installation of the IBM JRE for Windows 1.3	137
7.2.2 Download of Jinsight	137
7.2.3 Installation of Jinsight	137
7.2.4 Preparing a J2EE server to host Jinsight	138
7.2.5 Packaging Jinsight code to be deployed into a J2EE server	138
7.2.6 Installing Jinsight in the J2EE Server	140
7.2.7 Additional information on Jinsight	141
Chapter 8. Object Level Trace	143
8.1 Overview	144
8.2 OLT Implementation	145
8.3 OLT workstation installation	146
8.4 WSAD Remote Debugger quick install	148
8.5 WebSphere for z/OS installation	149
Chapter 9. Introscope	151
9.1 Introduction	152
9.2 Introscope architecture	152
9.2.1 Enabling an application for Introscope management	154
9.3 Using Introscope 2	155
9.3.1 Download	155
9.3.2 Workstation installation	155
9.3.3 z/OS Agent installation	156
9.3.4 J2EE server configuration	157
9.3.5 DB2 classes instrumentation	157
9.3.6 Trade2 instrumentation	159
9.3.7 Checking the installation	161
9.3.8 Disable Introscope monitoring	162
9.4 Using Introscope 3.0	163
9.4.1 Downloading and Installing Introscope	163
9.4.2 Overview	164
9.4.3 Installation	167
9.4.4 Running the Introscope Enterprise Manager	170
9.4.5 DB2 Considerations	171
Part 4. Appendixes	175
Appendix A. Sample code and JCL	177
The test client source code	178

DB2 DDL job for Trade2 database	182
RMF examples	189
SMF extract job	189
RMF report job	190
Sample partial RMF report	191
Appendix B. J2EE server naming convention and customization	193
J2EE Server relative names	194
RACFJ2EE REXX	195
Appendix C. System Management scripting API	201
JCL used to run SMAPI samples in batch	202
Using BPXBATCH.	202
Using TSO in batch.	203
Fallback SMAPI script	203
Create a server and install a J2EE application	209
Appendix D. Multiple WSAS nodes in a sysplex	215
Multiple WSAS nodes in a sysplex	216
The main WSAS node spans PRD1 and PRD2	216
The “test” WSAS node is only on the TST1	216
Common names and definitions shared by both nodes	217
Additional notes.	217
IBM support for multiple nodes in a sysplex	218
Appendix E. ISPF wizard-driven runtime upgrade	219
Appendix F. Additional material	233
Locating the Web material	233
Using the Web material	233
System requirements for downloading the Web material	234
How to use the Web material	234
Related publications	235
IBM Redbooks	235
Other resources	236
Referenced Web sites	236
How to get IBM Redbooks	237
IBM Redbooks collections.	237
Glossary	239
Index	241

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) [™] 	Parallel Sysplex [®]	S/390 [®]	WebSphere [®]
MQSeries [®]	Perform [™]	Sequent [®]	XT [™]
MVS [™]	RACF [®]	SP [™]	z/OS [™]
OpenEdition [®]	Redbooks [™]	System/390 [®]	zSeries [™]
OS/390 [®]	RMF [™]	VisualAge [®]	400 [®]

The following terms are trademarks of other companies:

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

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

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

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

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

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

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

Preface

This IBM Redbook covers the technical and organizational issues involved in migrating a WebSphere EJB application from a distributed environment to the z/OS platform. It provides guidance to application developers and WebSphere on z/OS administrators.

Part 1 covers the migration aspects and details the following:

- ▶ Hints and tips on how to optimize application code to fit seamlessly into its final deployment environment
- ▶ J2EE naming concepts that you need to perform the migration job
- ▶ A complete walkthrough of a migration project
- ▶ The necessary setup of deployment, test and production systems within the Parallel Sysplex

Part 2 describes how to manage WebSphere applications and details the following:

- ▶ How to integrate your migrated application into the managed z/OS infrastructure, for example Change Management and System Automation
- ▶ Centralized system administration, change management, and highly automated processes

Part 3 describes the following tools for WebSphere Application Server on z/OS:

- ▶ Jinsight
- ▶ Object Level Trace
- ▶ Introscope

The team that wrote this redbook

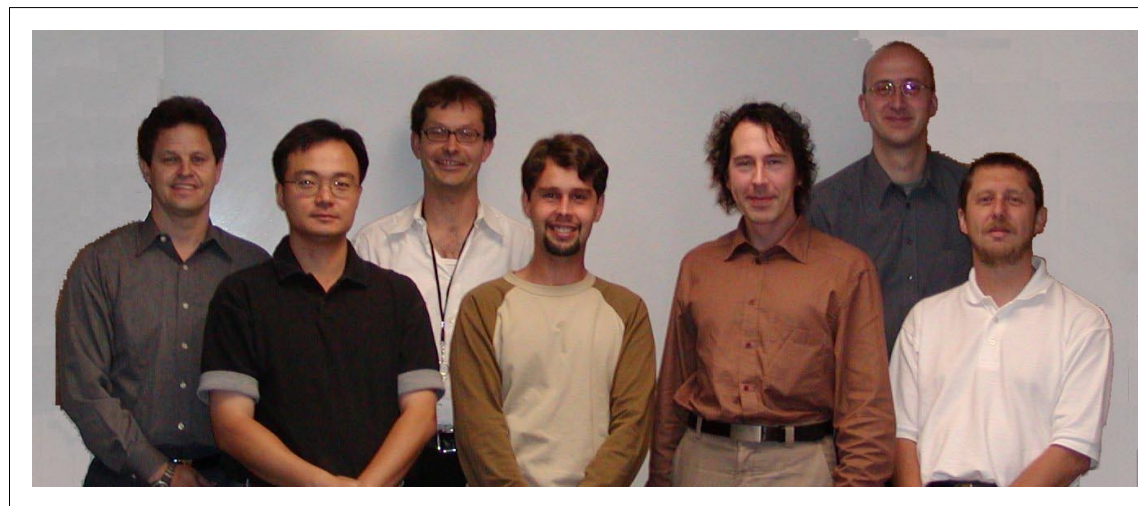


Figure 0-1 Redbook team: from the left, Louis, André, Holger, Stéphane, Michael, Jürgen and Tamas

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

Holger Wunderlich is a project leader at the IBM International Technical Support Organization, Poughkeepsie. He writes extensively and teaches IBM classes worldwide on all areas of e-business with IBM zSeries. Before joining the ITSO in 2001, he was a cross-server consulting IT Specialist in Germany. He worked for the IBM eServer Technical Support Team. He also worked in the OS/390 area as a systems programmer, systems engineer, and as technical support specialist in the USS & WebSphere world for three years. His areas of expertise include cross-server consulting, OS/390 UNIX System Services, e-business infrastructure, e-business security concepts, large scale ISP architectures and Java/390. He is co-developer of the IBM OS/390 Web server course. He also wrote several redbooks featuring large systems topics.

Stéphane Faure is a WebSphere Advisory IT Specialist with the e-Transaction Processing Design Center of the EMEA Product and Solutions Support Center in Montpellier, France. He has 12 years of experience in the computing field. He holds a degree in Computing from Gustave Eiffel College, Bordeaux. His areas of expertise include S/390, z/OS, VM, z/OS UNIX System Services, TCP/IP, IBM HTTP Server, WebSphere Distributed Platforms and z/OS. He is IBM-certified as a WebSphere Specialist, Solutions Expert in CICS Web enablement, e-business Solution Designer and e-business Solution Technologist.

Jürgen Lange is an IT Specialist for WebSphere at IBM Global Services in Germany. He has six years of experience in the computing field. His areas of expertise include e-business infrastructure and application design/development in the S/390 and e-business environment using products such as DB2, CICS, and MQSeries.

Louis E. Henderson is a Technology Infrastructure Manager at the Cap Gemini Ernst & Young Advanced Development Center in Costa Mesa, California. He has 23 years of experience in the computing field. He holds a Masters of International Management/Chinese degree from the American Graduate School of International Management. His area of expertise is in the many and various components of OS/390 and z/OS. He is currently involved in emerging technologies such as e-business, Linux and open standards/technologies.

André Watanabe is an Advisory IT Specialist in Sweden. He has 12 years of experience in the computing field. He holds a degree in Electrical Engineering. His areas of expertise include e-business infrastructure, z/OS UNIX System Services and Business Process Management products such as MQSeries Workflow. He has written extensively on e-business infrastructure on OS/390 and z/OS. He is currently involved in proof-of-concept projects and other presale activities.

Michael Zelbel is an IT architect with more than 10 years of experience working for Media For Business GmbH in Germany. His focus is on planning and developing e-business solutions on multiple platforms mainly for insurance and financial customers with a need for global integration.

Special thanks to the following:

Kenneth Muckenhaupt from the IBM Design Center for e-transaction processing in Poughkeepsie. Ken works closely with customers, helping them to migrate applications to the z/OS platform. He has extensive knowledge of application and migration issues. His work is available in an always up-to-date whitepaper version on the Internet. See our publication index for a link to it.

Tom Hackett, IBM Poughkeepsie and **Peter Kauffmann**, IBM Switzerland, independently did a great job on change management.

Michael Smith, IBM Poughkeepsie

Ivan Joslin, IBM Poughkeepsie

Many thanks to the following people for their contributions to this project:

Terry Barthel and Alfred Schwab, who edited this book.

Don Bagwell, Mike Cox, Ronald Lotter, IBM WSC Washington

Lance Buchholz, Wily Technology

Richard Conway, IBM ITSO, Poughkeepsie

David Cohen, IBM Poughkeepsie

John T. Gates Jr., IBM Raleigh

Tamas Vilaghy, IBM ITSO, Poughkeepsie

Michael Casile, WebSphere Performance, IBM Raleigh

Ed Lekanides, IBM Poughkeepsie

Michael Everett, IBM Poughkeepsie

Erik P. Jensen, IBM Watson

Lei Jiang, IBM Toronto

Who should read this book

Application Programmers. Large parts of this book are dedicated to application programmers coming from distributed platforms, scared of the “mainframe”. We establish the understanding of “390 lingua”, the need for job roles and administrative overhead, and we take away the fear that one has to log on to a green screen to develop applications for 390. We explain the obstacles that will show up when migrating to this specific platform, why they are there, and how to deal with them. You shouldn’t need much more information for your porting project than what you will find in this book (in addition to your usual documentation, of course).

z/OS WebSphere Administrators. If you are responsible for the runtime environment, for test and production systems, for change management and interaction with the z/OS infrastructure—this is your book! It will also help you to understand the application programmers, their questions and their expectations.

Decision Makers. What is the best platform for our application, where can it run, can we bring it to an enterprise level of quality? What is the cost of doing so? Is there a difference between the run times and if yes, does that affect my business? We have a little chapter just for you, briefing you on these issues.

Notice

The information in this publication is not intended as the specification of any programming interfaces that are provided by WebSphere Application Server or Java. See the PUBLICATIONS section of the IBM Programming Announcement for Websphere Application Server V4 for z/OS for more information about what publications are considered to be product documentation.

The advantage of WebSphere Application Server Version 4.01 for z/OS and OS/390 over former versions is that it is in full compliance with the J2EE specification. In particular, full support of Enterprise Java Beans (EJBs 1.1) is one of the main differences between WebSphere Application Server Version 4 and former versions. Because of this, we focused on the migration of the EJBs in this redbook.

Comments welcome

Your comments are important to us!

We want our IBM 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



Part 1

Migrating WebSphere applications

In this part we discuss the work that has to be done to migrate a distributed WebSphere Application to z/OS.



Introduction

In this chapter we establish a common understanding of the job roles in the WebSphere world and of WebSphere architectures. We introduce you to the driving reasons for migrating WebSphere applications to z/OS. We also describe the application we have selected to migrate and why. We use the word “migrate” throughout this book because of its common use, even though the term porting most aptly describes how an open architecture application should behave. It is similar to a hardware device that is unplugged from one port or connection and reconnected to another port.

We want to make software migration or porting approach the simplicity of moving hardware devices or household appliances. For example, unplug the vacuum from the living room, move the vacuum to the bedroom, plug it in and continue with cleaning the house. The software analogy is to take an application from one environment, use totally automated processes to make porting changes and continue running the application on the new platform.

The normal life of a WebSphere application is to be created and unit-tested on a personal platform such as Windows NT and then migrated to a runtime environment. Regardless of the ultimate runtime environment, the porting steps are to be done. Migration is a natural and accepted step in the process of WebSphere applications. We discuss an application that was written without consideration of where it would run. The public download site doesn’t (or didn’t) mention z/OS as a runtime platform, but we asked, “Why not”?

1.1 Overview

A few years ago, the Java language quickly appeared to be a fast, open and reliable way to write new applications. The rising enthusiasm of the computing community pushed new standards which helped software companies to begin offering truly open software. New applications quickly started to be deployed and put into production. The first applications addressed small audiences, tended to be simplistic and had a short life span.

Today, the deployment of Java applications is more likely to be enterprise-wide and World Wide Web-wide. More users, more geographies, more open hours, and more businesses require stronger application containers. z/OS and its predecessors, OS/390 and MVS, have robust facilities and architectures that have delivered high levels of availability and reliability to other application environments for many years. With the advent of WebSphere Application Server for z/OS, the same class of service can also be utilized by Java applications on the World Wide Web.

Nowadays, application servers need:

- ▶ High reliability
- ▶ High integration with backend systems and systems operations
- ▶ High system management
- ▶ Transaction management at a two-phase commit level
- ▶ High security
- ▶ High application availability and nondisruptive upgrades
- ▶ Continuous support for adaptive loads
- ▶ More network bandwidth

These characteristics are the reason for the existence of z/OS. They are not design points added on after the need to process core-business complex transactions was discovered. This is why the optimization of one or more of the above criteria is the usual impetus to port an application to z/OS.

The second reason for porting to z/OS is to consolidate different application types to run side by side on a single platform.

Java 2 Enterprise Edition (J2EE)¹ standards have brought to the market a way to develop Enterprise Java applications in a way that they are likely to be deployed in any Java 2 Enterprise Edition compliant server. IBM WebSphere Application Server for z/OS and OS/390 is the meeting of J2EE standards and z/OS

strengths. Its integration with z/OS uses system mechanisms which have given z/OS its reputation as the most robust production server, while the implementation of J2EE specifications makes all the superior system integration transparent to programmers.

In this book, we present our findings on how to port a J2EE enterprise application already running under WebSphere on a Distributed Platform to IBM WebSphere Application Server for z/OS and OS/390. Our primary goal is a practical porting guide. To this end, we made a detailed list of migration issues that must be resolved during the migration process.

To the extent that your application follows the standards for J2EE applications, migration issues are relatively easy to identify, describe, and resolve. In some cases, it is simply a matter of performing a similar step with slightly different tools. Other cases require minor source code changes. To the extent that your application exploits specific extensions, the migration effort can become more complex.

The good news is that the majority of your applications will require only minor changes, and possibly none at all.

We assumed that the common migration path for a distributed non-J2EE application is:

1. Upgrade first to J2EE programming APIs on the distributed platform it used to run on.
2. Migrate to WebSphere on z/OS.

((FIX)) The first step of the migration is described in a Redbook listed in “Related publications” on page 235. so we chose to focus on the second migration step: installing a J2EE application on IBM WebSphere Application Server for z/OS and OS/390.

As a practical guide, we have attempted to be comprehensive, but not exhaustive. There are certainly applications that have subtle dependencies that we will not anticipate — these cases are obviously not covered. Our goal is to include the major migration issues that we expect most clients will encounter.

¹ The Java 2 Platform, Enterprise Edition defines a simple standard that applies to all aspects of architecting and developing multi-tier server-based applications. J2EE defines a standard architecture composed of an application model, a platform for hosting applications, a Compatibility Test Suite and a reference implementation. The primary concern of J2EE is the platform specification: it describes the runtime environment for a J2EE application. This environment includes application components, containers, resource manager drivers, and databases. The elements of this environment communicate with a set of standard services that are also specified.

For more information, go to <http://java.sun.com/j2ee/>

In order to illustrate our recommendations, we moved a J2EE application, the WebSphere sample performance benchmark application, to z/OS and described each step of the installation into the new WebSphere. WebSphere sample performance benchmark application is the application commonly used inside IBM to run WebSphere Application Server performance benchmarks, so it contains all J2EE objects you may encounter.

Even if we assume that our audience is familiar with the specifications and tools used to build and deploy J2EE applications, this book is not a programming guide nor a cookbook. It is most likely to be a first stone laid in a foundation to make the following audiences meet together and start working beside each others:

- ▶ Application developers
- ▶ System architects
- ▶ z/OS system programmers
- ▶ WebSphere specialists

This chapter is dedicated to define the specific terms used in z/OS which needs to be precise. Either they are specific to this operating system or they mean other mechanisms than in a distributed environment. Since new teams will have members who are not familiar with z/OS terminology, a definition of these terms will help with communications between these groups. This chapter also expands the Java 2 Enterprise Edition roles to the deeper specializations extant on z/OS system teams.

Chapter 2, “WebSphere sample performance benchmark application” on page 23 describes our environment, and the application we attempted (and succeeded) to install on IBM WebSphere Application Server for z/OS and OS/390.

Chapter 3, “Preparing for WebSphere application migration” on page 31 shows how to prepare the port: loading an application into the integrated development environment, and checking its efficiency to run in its native environment.

Chapter 5, “The lifecycle of an application” on page 81 is the port itself. We list the steps we used to install the applications. We also list the known additional porting steps you may encounter. This chapter ends by demonstrating how to automate J2EE application installations on IBM WebSphere Application Server for z/OS and OS/390.

At this point, the application is successfully running on z/OS. During the application’s life cycle, it will certainly be tested, verified, updated and finally installed in a production machine. These after-port management issues are described in 5.7, “Change and problem management” on page 120.

Chapter 6, “Management summary” on page 129 expands the considerations that favor choosing IBM WebSphere Application Server for z/OS and OS/390 as the J2EE container for enterprise applications.

Ending this book, we show how we installed tools that can help the z/OS porting and optimization tasks:

- ▶ Chapter 7, “Jinsight” on page 135
- ▶ Chapter 8, “Object Level Trace” on page 143
- ▶ Chapter 9, “Introscope” on page 151

1.2 Quick introduction to z/OS vocabulary

z/OS systems people have created their own vocabulary, which may be confusing for those working in a distributed platform environment. In “Glossary” on page 239 we list words that can cause some confusion. They are either specific to z/OS, or they are used in all environments but have a different meaning in z/OS.

Table 1-1 is a quick introduction to z/OS terms you need to know.

Table 1-1 z/OS basic vocabulary

z/OS or OS/390 vocabulary...	...and its meaning in the distributed world
DASD (Direct Access Storage Device)	Disk drive
Sysplex	Cluster with shared data
System	Node or Clone
Server Address Space	Daemon
Region or Address Space	Process or Address Space
TCB (Task Control Block)	Thread
(WebSphere) Control Region	Application Server Listener
(WebSphere) Server Region	Application Server JVM
WebSphere Daemon	Location Service Agent
IML (Initial Microcode Load)	Power off / Power on the system
IPL (Initial Program Load)	Reboot the system
Member (in a partitioned data set)	File

z/OS or OS/390 vocabulary...	...and its meaning in the distributed world
Partitioned data set	Set of files, directory
Sequential data set	File
(WebSphere) System Management Server	Administrative Server
(WebSphere) System Management End User Interface (synonyms: SM EUI, SM GUI, SM UI, Administration Application)	Administrative Client or Administrative Console
SMP/E	Software maintenance program
(WebSphere) Conversation	Configuration (or level of configuration)

1.3 EJB roles versus job roles in the WebSphere world

There are new job roles with new kinds of applications. Old roles change, getting new responsibilities. In the distributed world, different roles and responsibilities are often mingled together. In a tightly controlled environment, the roles are clearly defined and permission has to be given accordingly.

If you come from a distributed platform, you might be annoyed that you have to follow formal processes for getting access to system resources, but this separation of system control is the only way to manage large systems with thousands of users and applications.

On the other hand, z/OS people have to learn that the new tasks of the J2EE runtime environment are complex and have to be merged into existing job roles, and that new responsibilities have to be created.

The Sun “Java2 Platform Enterprise Edition Specification, v1.2” document describes (in Chapter 2, “Platform Roles”) six typical Java 2 Platform, Enterprise Edition roles (not to be confused with EJB roles, which are related to J2EE application security). Although these roles are considered to be typical, they may fit differently into a z/OS organization.

Using the names defined in the J2EE Specification, we briefly describe each EJB role and match it to one or more job roles.

We found it useful to group the roles into these three categories:

- ▶ Infrastructure roles
- ▶ Development roles

- Implementation roles

1.3.1 Infrastructure roles

Even if an infrastructure role does not match any job role, we describe it in this section in order to be consistent with the J2EE Specification.

J2EE product provider

A J2EE Product Provider implements a J2EE product providing the component containers, J2EE platform APIs, and other features defined in this specification. A J2EE Product Provider must provide:

- The J2EE APIs to the application components through the containers
- The mapping of the application components to the network protocols as specified by this specification
- Application deployment and management tools

In our context, the J2EE Product Provider is IBM and the J2EE product is IBM WebSphere Application Server for z/OS and OS/390.

Tool provider

A tool provider provides tools for the development and packaging of application components.

Platform-independent tools can be used for all phases of development of an application up to its deployment.

Platform-dependent tools are used for deployment, management, and monitoring of applications.

1.3.2 Development job roles

Application component provider

There are multiple roles for application component providers, including HTML document designers, document programmers, enterprise bean developers, etc.

These roles use tools to produce J2EE applications and components.

The people typically involved in application design are:

OO designer	Highly skilled in IT and OO and knows most of the technical environment. Works intimately with the application architect.
Java developer	Entrusted with delivering the code of the application.

- Application architect** The technical lead on the project. Has overall responsibility for determining the high-level structure of the application, and the interaction between components. Is also a senior developer.
- Web designer** Skilled in the use of HTML and the tools used to develop and maintain Web content. Responsible for delivering HTML and JSP pages and the various interface elements, such as images contained within them.

Application assembler

The application assembler takes a set of components developed by application component providers and assembles them into a complete J2EE application delivered in the form of an Enterprise Archive (.ear) file. The application assembler generally uses GUI tools provided by either a platform provider or tool provider. The application assembler is responsible for providing assembly instructions describing external dependencies of the application that the WebSphere administrator must resolve in the deployment process.

1.3.3 Implementation job roles

WebSphere administrator

The WebSphere administrator, an expert in a specific operational environment, is responsible for deploying Web applications and Enterprise JavaBeans components into that environment. The WebSphere administrator uses tools supplied by the J2EE product provider to perform the deployment tasks. The responsibilities of this role include:

- ▶ Installation
 - Move the media to the server.
 - Generate the additional container-specific classes and interfaces that enable the container to manage the application components at runtime.
 - Install the application components and additional classes and interfaces into the J2EE containers.
- ▶ Configuration
 - Resolve all the external dependencies declared by the application component provider.
 - Follow the application assembly instructions defined by the application assembler.
- ▶ Activation

Start up the newly installed and configured application. In some cases, a qualified WebSphere administrator may be able to develop sample code to check application or connector installation.

The WebSphere administrator's output consists of Web applications, enterprise beans, applets, and application clients that have been customized for the target operational environment and are deployed in a specific J2EE container.

A WebSphere administrator in an IBM @server environment sits in the middle of the participating roles and, therefore, will most likely need basic knowledge in the following areas:

- ▶ UNIX operating system and TCP/IP networking
- ▶ z/OS system overview
- ▶ z/OS security
- ▶ Backend connectors

Moreover, in most cases the administrator will be responsible for problem source identification, which requires solid expertise in Java, J2EE, and connector programming.

System administrator

The system administrator is responsible for the configuration and administration of the enterprise's computing and networking infrastructure.

z/OS installations typically use specialists to manage their systems. The vast number of different applications, users, and data demand a high degree of specialization and optimization.

The most common system roles you find on z/OS sites are:

▶ **System programmer**

Responsible for installing, customizing, and maintaining z/OS systems.
Common tasks are:

- Installing and defining the base operating system services
- Creating and naming files and data sets
- Customizing system tasks (System Logger, RRS, etc.)
- Installing and maintaining products and services using specialized facilities
- Monitoring and tuning systems

▶ **Security administrator**

- Is responsible for the customization of the security product usually mentioned as the "External Security Manager".
- Manages the users, groups and resources on the system.
- Because IBM WebSphere Application Server for z/OS and OS/390 is tightly embedded in z/OS security, the security administrator plays a key role in all phases of a J2EE application setup.

- Defines and implements security concepts together with the system programmer and the appropriate application specialist.
- ▶ **DB2 administrator**
 - Manages the database, indexes, table spaces, the tables, and all objects.
 - Is in charge of setting up connections to other databases and enabling the JDBC driver.
 - Does the bindings of static queries.
 - Depending on the kind of system (test or production), may have full security authority of DB2 resources or may delegate it to the security administrator.
- ▶ **Network administrator**
 - Maintains the TCP/IP configuration.
 - Can be involved in setting Domain Name Server (DNS) entries for IBM WebSphere Application Server for z/OS and OS/390.
 - When using workload balancing mechanisms such as DNS/WLM, WebSphere Edge Server or Sysplex Distributor, is involved in the TCP/IP setup and the dynamic routing that may be involved.
 - Deals with the TCP/IP security-related issues and might be responsible for enabling the firewall and intrusion detection services.
 - If quality of service (QoS) routing is required, is responsible for setting up the proper infrastructure and the policy agent.
- ▶ **UNIX System Services administrator**

z/OS sites that have already run UNIX System Services applications may have people specialized in this area. They know the HFS conventions, the way USS has been implemented, and the available tools—i.e., they are UNIX administrators *without* superuser capabilities.
- ▶ **System operator**

System operators are responsible for running all the system services. They:

 - Manage the starts and stops of servers, automation, and execution of changes and fallbacks.
 - Monitor application availability and response time.
 - Use defined interfaces for problem reporting.

1.3.4 Skills needed during an application's life

Here we group the people who will be involved in the tasks concerning IBM WebSphere Application Server for z/OS and OS/390.

Installing the WebSphere base runtime

- ▶ System Programmer (PROC- and PARMLIB definitions, RRS, ARM, System Logger, SYSPLEX, naming conventions, changes, etc.)
- ▶ Network Administrator (port reservation, naming conventions)
- ▶ IBM HTTP Server/Unix System Services Administrator
- ▶ Security Administrator (user IDs, authorizations, LDAP)
- ▶ Performance Specialist (Workload Manager definitions)
- ▶ DB2 Administrator

Installing a new J2EE server

- ▶ System Programmer (J2EE server System Tasks creation in PROCLIB)
- ▶ Security Administrator (J2EE server System Tasks definitions, associated user IDs and authorizations)
- ▶ Performance Specialist (Workload Manager definitions)
- ▶ DB2 Administrator (access to generic tables. Ex.: BBO.STATEFULL)

Installing a new J2EE application

- ▶ System Programmer
- ▶ Security Administrator (user IDs, authorizations, J2EE roles)
- ▶ Performance Specialist (adjusting Workload Manager definitions)
- ▶ DB2 Administrator if there is DB2 persistence or jdbc
- ▶ Backend Specialist

Managing an existing J2EE application

- ▶ System Operator
- ▶ System Programmer
- ▶ Network Administrator (workload balancing, availability)
- ▶ Performance Specialist (analyzing SMF records, adjusting Workload Manager definitions)
- ▶ DB2 Administrator if there is DB2 persistence or jdbc
- ▶ Backend Specialist

1.4 WebSphere architectures and topologies

In this section we provide an overview of a variety of appropriate topologies for applications on WebSphere Application Server. Where necessary, we point out differences between WebSphere Application Server on distributed platforms and WebSphere Application Server on z/OS. This information can be used to plan the topology of a new application or to examine the topology of an existing application that should be ported to z/OS.

Table 1-2 shows an overview of our definition of 2-, 3- and 4-tier topologies:

Table 1-2 Overview of n-tier topologies

	2-tier	3-tier	4-tier
Client tier	Separate	Separate	Separate
Presentation tier (WebTier)	Integrated in a server tier	Integrated in a functionality tier	Separate
Business tier (EJB Tier)	Integrated in a server tier	Integrated in a functionality tier	Separate
Data tier (EIS Tier)	Integrated in a server tier	Separate	Separate

1.4.1 Blueprint of a multi-tiered architecture

Sun defines a J2EE application as multi-tiered. Their J2EE architecture blueprint distinguishes between the following tiers (Figure 1-1 on page 15):

- ▶ Client Tier - browser or Java application
- ▶ Web Tier - presentation, JSP, servlets
- ▶ EJB Tier - business logic, rules, components
- ▶ EIS Tier - (Enterprise Information Services) databases and legacy systems

Even if these names are not used, this would still be the standard architecture to follow for a J2EE application.

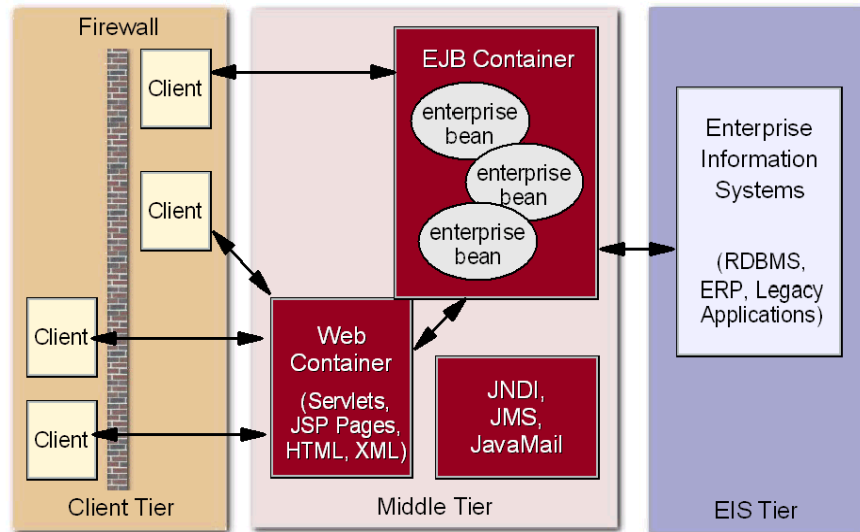


Figure 1-1 Sun's blueprint of a multi-tiered J2EE application

1.4.2 The individual tiers

Understanding the individual tiers is crucial to architecting the overall application and infrastructure design.

Client tier

In recent years, Web-based presentation of data has become a standard in a lot of scenarios, and it is widely accepted by the vast majority of users. Since we are dealing with Web applications in this publication, the client tier is assumed to be either an HTTP client such as Netscape Navigator or a Java client interacting with EJBs through Remote Method Invocation (RMI) over Internet Inter-ORB Protocol (IIOP).

XML parsing for presentation can be done within the client tier, but is most often found in the presentation tier. Nevertheless, XML over HTTP is often used by the clients to communicate with the WebSphere Application Server.

Presentation or Web tier

The presentation of structured and unstructured data is done in very different ways for different purposes. As a common point for our scenarios we assume that the presentation is Web-based. Therefore, the presentation tier is used to form all possible output data to a Web-based format suitable for the client tier.

This functionality can be provided, for instance, by a WebSphere application server. A possible scenario could be that WebSphere hosts EJBs that provide the output data, and JSPs or servlets that transform this data into HTML.

The presentation tier is usually located inside the demilitarized zone (DMZ), a secure network segment between the Internet and the companies' intranet.

Business or EJB tier

This tier hosts the business logic of the application. It should encapsulate the complete logic needed to perform all sorts of functions, and it should not leave any logic task for the other tiers.

The business tier itself might be structured in a modular way. However, here we focus on the tier as a whole, assuming it is a black box suitable for providing every logical function needed to perform a task by the application. An implementation of the business tier might be as simple as a set of EJBs for calculation hosted on an NT-based WebSphere system, or as complex as a legacy system on a mainframe.

Data or EIS tier

The complete data produced and used by the application is hosted in the data tier. Thus, being a separate tier, a goal of implementation often is to keep the data as close as possible to the components that make use of it.

For this purpose, a well-designed data tier, which avoids unnecessary copies of data but provides aliases and copies wherever they make sense, is essential for functionality, reliability and performance. One basis for such a data tier could be a DB2 system on z/OS.

1.4.3 Two-tier applications

A widespread topology of Web applications on z/OS is a 2-tier model, where a single server tier provides the complete functionality of data warehousing, computing and presenting.

1.4.4 Multi-tier applications

When talking about multi-tier topologies (often referred to as “zoo topologies”), we mean 3- and 4-tier models that divide the server tier into more detailed components such as the presentation tier, the business tier, and the data tier.

Typically the tiers can be run on separate machines and therefore add to the scalability of the application.

1.4.5 “Common” practice: 3-tier applications

The most common implementation of WebSphere is the 3-tier model:

- ▶ Client tier - browser
- ▶ Middle tier - HTTP Server with the WebSphere plug-in, or an HTTP and WebSphere Application Server combination
- ▶ WebSphere and backend tier

Some of the reasons that made this configuration popular are:

- ▶ Security
 - Only the HTTP server has to be exposed in the DMZ once the plug-in is able to access WebSphere Application Server through the network.
 - Having WebSphere Application Server integrated into the backend system makes it possible to easily share the local OS security mechanism when applications connect to the backend. If the accessing process is running with thread level security, its identity can be used directly to authorize access to the backend data.
- ▶ Management
 - Workload management can be initiated by the WebSphere HTTP Server plug-in.
 - The whole “application” logic resides in the third tier, so the maintenance and operation staff need to focus only on one machine without worrying about an “application network”.
- ▶ Cost

Presentation processing and XML parsing can be done using the cheap cycles in the middle tier, while only crucial transactions use the high quality service infrastructure in the third tier.
- ▶ Performance

When the WebSphere Application Server is in the same machine as the backend (physical 2-tier, logical 3-tier), it can make use of local connectors in place of network connectors to access backend-managed data.

In IBM WebSphere Application Server for z/OS and OS/390 Version 3.5, the 3-tier implementation is slightly different because the WebSphere runtime is hosted inside the HTTP address space. But the backend system is usually isolated on another machine that is preferably accessed by a protocol other than TCP/IP, such as SNA or XCF. This does not prevent the use of local connectors because of sysplex integration of backend systems (for example, with CICS you

can use a local connector to access a local TOR that will start a transaction in a remote AOR). Exposing the WebSphere machine inside the DMZ is less problematic than on other platforms because an External Security Manager has to be configured on this z/OS machine.

Now, with IBM WebSphere Application Server for z/OS and OS/390 Version 4, all options listed above are available.

1.4.6 Multi-tier minus one

In a typical scenario, the presentation tier of an application is hosted on a distributed platform, while the business tier and the data tier are located in a centralized environment (for example, a WebSphere cluster on Windows NT presents the data for a legacy system on z/OS).

Porting the presentation tier to z/OS provides the opportunity to replace network-based communication between the tiers with local intra-mainframe communication, which performs much better.

In this case, the presentation tier is not removed but is instead integrated into the other tiers. We express this concept by adding “- 1” to the name of the topology.

1.4.7 Moving the business tier to z/OS

When talking about moving applications to z/OS, we mainly think about moving the business tier to this platform. There are several reasons for this:

- ▶ The EJB container is closer to the data, which might already be on z/OS.
- ▶ A well-performing and reliable local 2-phase commit can be used.
- ▶ The EJBs are using CICS or IMS connectors, which are available as local connectors.
- ▶ The security available on the z/OS system can be applied until the HTTP Server request handling.
- ▶ The z/OS sysplex can provide scalability modifications (new engine, new machine in the sysplex, etc.) without service interruption.
- ▶ Due to smart workload balancing and white space computing capabilities, peaks in the workload are well managed in the z/OS system.

Therefore, it is especially rewarding for the business tier to reside on a z/OS platform, while it might be acceptable to leave the presentation tier on a distributed system.

However, if the presentation logic also resides on z/OS, it will be easier to share the same security context between the presentation logic objects (servlet/JSPs) and business logic objects (EJBs). Moreover, if both are in the same server, they will communicate through cross memory access instead of using network access. This clearly leads to better performance and avoids security issues between EJB clients (servlet/JSPs) and EJBs.

The fact that the connectors are available as local connectors in IBM WebSphere Application Server for z/OS and OS/390 Version 4 might be a good reason in many cases to port the business logic to z/OS.

1.4.8 Security model with IBM WebSphere Application Server for z/OS and OS/390

With WebSphere Application Server Version 4.0 you use deployment descriptors to define roles and associate these with servlets and EJB methods. At install time you supply a list of users and groups that are to be associated with each of the roles.

To establish identity within the business process you use the EJB *run-as* capability². For this reason we recommend that you place an EJB session bean on top of the business logic tree. The security context can be propagated to backend systems. This avoids developing any code to propagate the user's identity. z/OS and the WebSphere Application Server have all the mechanisms in place to deal with this issue on the thread-level base.

In the deployment descriptor you also define the policy for the authentication of Web clients. Choose between the following:

- ▶ HTTP Basic Authentication
- ▶ X.509 Client Certificate
- ▶ Form-based login

Digest authentication (see <http://www.ietf.org/rfc/rfc2617.txt>) is today not supported by IBM WebSphere Application Server for z/OS and OS/390.

² EJB run-as is the ability for the deployer to decide which identity should be used for authorization of any (authenticated or unauthenticated) Java request. You should be aware that run-as does not change the MVS security context in the execution; it does not change the ACEE. However, in certain situations one can use the sync-to-thread deployment function to achieve thread-level security.

Especially in larger environments, which are typical for z/OS, one alternative is to accept the requests from a trusted authentication server. In this case, you specify an HTTP(S) port that is trusted and the administrator ensures that traffic coming through this port can be trusted (for example, by controlling the network path from the authentication server and the application server and ensuring that the authentication server is the only server that can connect to the port).

For such requests, WebSphere looks up the authentication data in the HTTP header.

1.5 System landscape

When developing an application for z/OS, you usually make use of various systems such as a development system, a test system, a verification system, and a production system. Each of these systems might come with its own configuration repository. We refer to the complete collection of systems that are used in a particular project as the *system landscape*. The purpose of a landscape is to provide a way to develop, debug, validate, and run applications in parallel without interference. When porting an application to z/OS, you use the same landscape as when you develop a new application, except that the test system might be left out. z/OS now offers multiple WebSphere nodes within one sysplex as described in Appendix D, “Multiple WSAS nodes in a sysplex” on page 215.

However, when migrating the Trade2 application, we used a development, a test, and a production system, leaving out the validation system.

1.5.1 Development system

You use the development system to produce applications or parts of applications in a way that is efficient and convenient to the developer. Therefore, you choose a platform like Windows NT, which is supported by IDEs like Visual Age or WebSphere Studio Application Developer, as the tool to support the J2EE programming model.

You provide the developer with access and administrative rights for the development system.

Usually the components of an application are unit-tested on the development system.

Because code changes might be necessary when porting an J2EE application from a distributed to a z/OS environment, we also recommend that you set up a separate distributed test environment as part of the development environment. There are some other reasons for setting up such an environment, for example:

- ▶ Using VisualAge for Java (VAJ) or WebSphere Studio Application Developer for development

In comparison with WebSphere Application Server Version 4.0, VAJ supports different specification levels (JDK, Servlet, JSP and EJB Level) for developing, automatic code generation, and testing. This can cause some errors after deploying to the application server, which should be removed before installing on the z/Series. We recommend using WebSphere Studio Application Developer.

- ▶ Testing of the whole application

A Web or an J2EE application consists of several parts (EJBs, servlets, JSPs, HTML pages, graphics, etc.), which are mostly developed separately. Practical experience shows that some changes are needed in some of these parts until the application runs without problems. We recommend testing the whole application on distributed platforms to avoid these problems while deploying to z/OS.

- ▶ z/OS-specific changes

If it is known that the application is executable on distributed platforms, only z/OS- or OS/390-specific problems can occur while deploying on these platforms.

1.5.2 Test system

Especially when staging applications that use J2EE CICS or IMS connectors, there is a need for a separate z/OS test system.

In the past, Java applications for z/OS were often developed and tested on distributed systems. The tests could have been very meaningful because of the platform independency of Java and because of the variety of connectors that have been available as TCP/IP-based network versions.

Since WebSphere Application Server Version 4.0 for z/OS and OS/390, only local connectors are available for z/OS. They perform better than the network connectors, but they necessitate a z/OS test system. Any test that includes a CICS or IMS connector needs to be run on z/OS instead of on a distributed platform.

Though the test and production systems should be as similar as possible, you usually set up the test system with two major differences:

- ▶ You enable profiling and debugging tools such as Jinsight and OLT on the test system.
- ▶ You grant more access rights on the test system than to the production system.

These differences enable you to efficiently debug on the test system while keeping the well-managed setup on the production system.

1.5.3 Verification or quality assurance system

The idea of a verification or quality assurance (QA) system is to provide a system with exactly the same access rights and monitoring tools as the production system. Here you incorporate a new version of an application into the existing environment without affecting the production system. If a new application runs successfully on such a verification system, it is very unlikely that it will fail on the production system. Therefore, it is usually deployed on the production system directly after a successful test on the verification system.

If a test on the verification system fails, then the test system is used to figure out what the problem is, since the test system is usually equipped with debugging tools that are not available on the verification system.

1.5.4 Production system

The production system hosts the application and the necessary business data. It is unrelated to all other systems and therefore not affected by tests on the other systems. It is updated only with applications that are already tested on the test and verification systems, and therefore there is only a very low risk for problems. Also, typical production systems have well elaborated and tightly controlled change management procedures.



WebSphere sample performance benchmark application

In this chapter we list the software we used for this project and describe the application we chose to migrate: the WebSphere sample performance benchmark application. We explain why we chose it and how we customized it.

2.1 Environment and software levels

The software levels and infrastructures used at client and server sides are described here.

2.1.1 Development environment

At the development site, we used the following software and service levels for the integrated development environment (IDE):

- ▶ Windows 2000 Service Pack 2
- ▶ VisualAge for Java Version 4.0, Enterprise Edition
- ▶ WebSphere Studio Version 4.0 Advanced Edition
- ▶ WebSphere Studio Application Developer, beta version

For a complete description of the development environment, see 3.2.1, “The integrated development environment” on page 36.

2.1.2 WebSphere Application Server distributed test environment

At the development site, the software and service levels for the testing runtime environment were as follows:

WebSphere Application Server Version 4.0.1 Advanced Edition for Windows NT.

For a complete description of the distributed test environment, see 3.2.2, “The distributed Deploy- and Runtime-Environment” on page 38.

2.1.3 z/OS environment

The software and service levels for the target side were as follows:

- ▶ The z/OS 1.2 base operating system
- ▶ IBM WebSphere Application Server for z/OS and OS/390 Version 4.01 pre-GA code

We upgraded from Version 4.0. The upgrade process is described in 5.6, “Nondisruptive upgrade of the WSAS runtime” on page 120.

- ▶ IBM WebSphere Application Server for z/OS and OS/390
- ▶ z/OS Application Assembly Tool Level 22
- ▶ DB2 for z/OS 7.1

2.2 The sample application

We chose the WebSphere sample performance benchmark application as the sample application. It uses J2EE components such as servlets, Java Server Pages (JSP), Enterprise Java Beans (EJB), and Java Database Connectivity (JDBC).

2.2.1 WebSphere sample performance benchmark application

The WebSphere performance benchmark sample provides a suite of IBM-developed workloads for characterizing performance of the WebSphere Application Server. The workloads consist of an end-to-end Web application and a full set of Web primitives. The applications are a collection of Java classes, Java Servlets, Java Server Pages and Enterprise Java Beans built to open J2EE APIs. Together they provide versatile and portable test cases designed to measure aspects of scalability and performance.

The WebSphere sample performance benchmark application consists of two separate applications:

- Trade2

Trade2 is an end-to-end Web application modeled after an online brokerage. Trade2 leverages J2EE components such as servlets, JSPs, EJB and JDBC to provide a set of user services such as login/logout, stock quotes, buy, sell, account details, etc., through a standards-based HTTP protocol.

- Web Primitives

The Web Primitives provide a set of workloads to individually test various components of the WebSphere Application Server. The primitives leverage the Trade2 application infrastructure to test specific WebSphere J2EE components such as the servlet engine, JSP support, EJB Entity and Session beans, HTTP Session support, and more.

2.2.2 Trade2 topology and architecture

The Model View Controller

The Trade2 application implements the model view controller (MVC) architecture, leveraging the Enterprise Java programming model. Figure 2-1 on page 26 shows a simplified topology view of the Trade2 application.

The servlets act as controllers, with responsibility for page navigation, session management, business logic calls, and other such tasks. They defer the responsibility for presentation handling to JSPs.

Enterprise JavaBeans are called by servlets to run the business logic, access the backend system (DB2 in this case), and provide the data to be returned to the JSPs. Freed by servlets and EJBs from the code to handle such issues as enterprise access, JSPs can concentrate on presentation, with JSP tags used only to insert dynamic data where needed in the HTML flow.

Tip: For more information about the MVC model, refer to: *Design and Implement Servlets, JSPs and EJBs for IBM WebSphere Application Server*, SG24-5754 and the IBM Patterns for e-business Web sites at:

<http://www.ibm.com/framework/patterns>

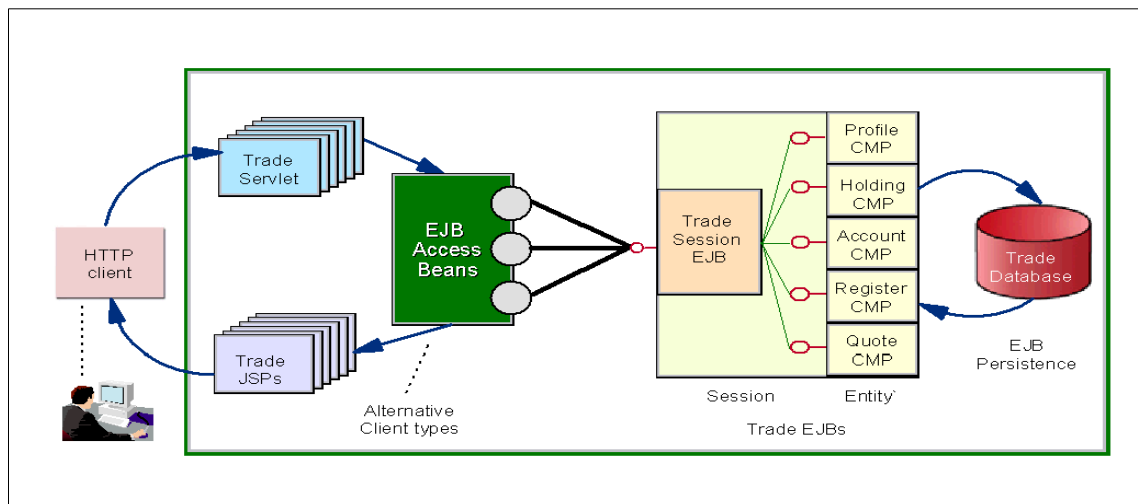


Figure 2-1 Top-level Trade2 architecture topology

Trade2 models an online securities trading firm. It provides high-level functions including login/logout, buy/sell, quote, etc.

EJB and database tier

EJBs manage data and transactions for business level operations, such as buy and sell. Figure 2-2 on page 27 shows this tier in more detail.

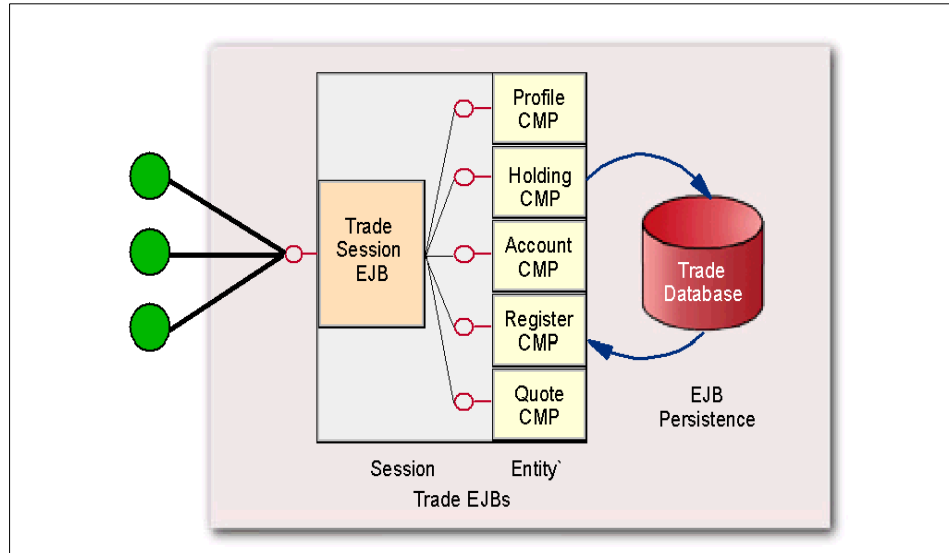


Figure 2-2 EJB and database tier

Servlet and JSP tier

Servlets coordinate user interactions with access to business operations, manage user sessions, and dispatch results to JSPs. JSPs provide user interface and input coordination.

The servlet tier plays a critical role in the scalability of a Web application through servlet instance replication. Figure 2-3 on page 27 shows this tier in more detail.

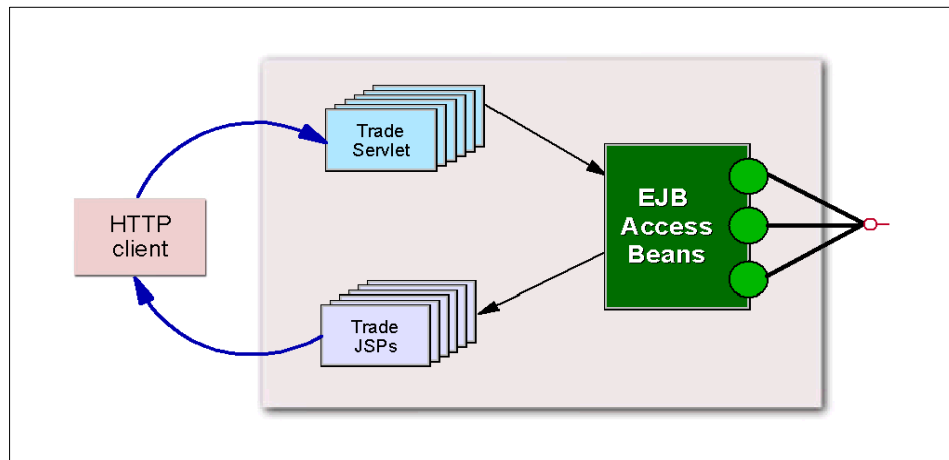


Figure 2-3 Servlet and JSP tier

VisualAge for Java EJB access beans

EJBs are wrapped by VisualAge for Java (VAJ)-generated EJB access beans. EJB access beans provide:

- ▶ Reduced complexity of EJB access through simplified client APIs
- ▶ Transparent caching of EJB Homes and initial Context objects
- ▶ Reduced remote calls by making fine-grain setter/getter methods local and providing a single refresh/commit remote call.

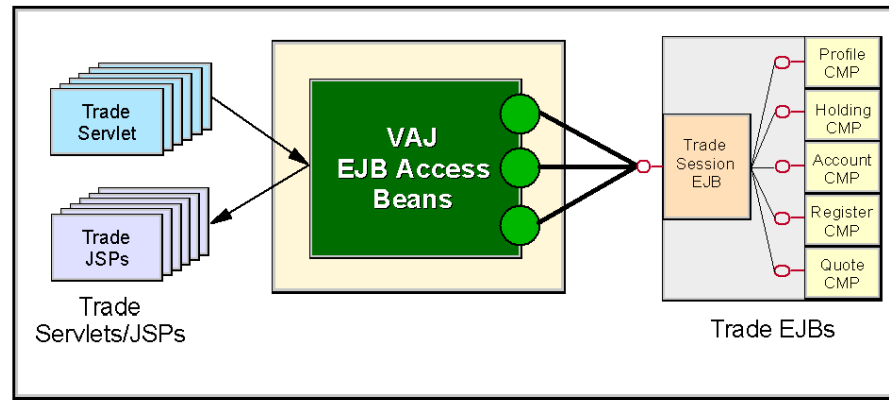


Figure 2-4 VAJ EJB access beans

VAJ access beans in Trade2 leverage VAJ EJB "Copy Helper" access bean optimizations.

See the example for Account update user service in Figure 2-5 on page 29.

- ▶ ProfileAccessBean attributes are updated individually from the user request.
- ▶ The Profile EJB entity update is then committed in a single remote call.

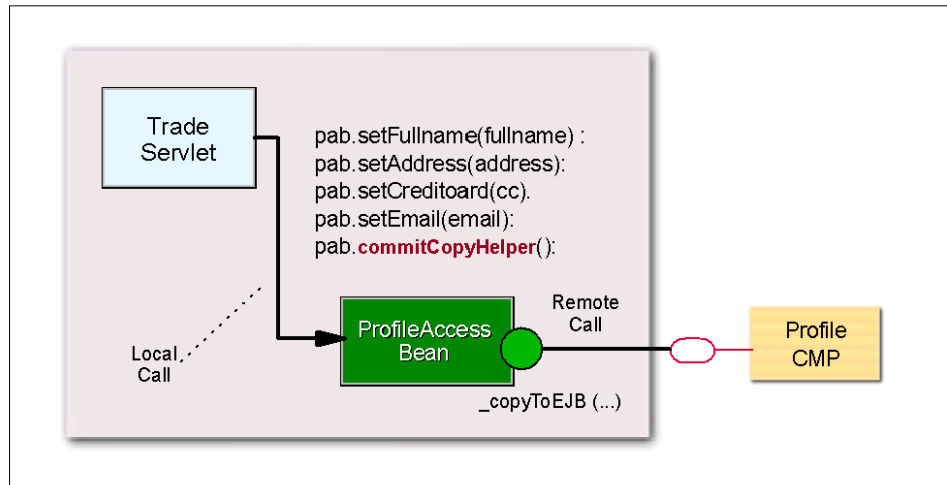


Figure 2-5 Account update example

See the example of the Quote Cache Java bean in Figure 2-6.

- ▶ QuoteCache uses VAJ "Copy Helper" access beans to cache Quote entities.
- ▶ Quote data is initialized from the Quote EJB entity and cached in a servlet side hashtable.
- ▶ Quote data is refreshed from the EJB entity on a user-configured timeout providing "delayed quotes".

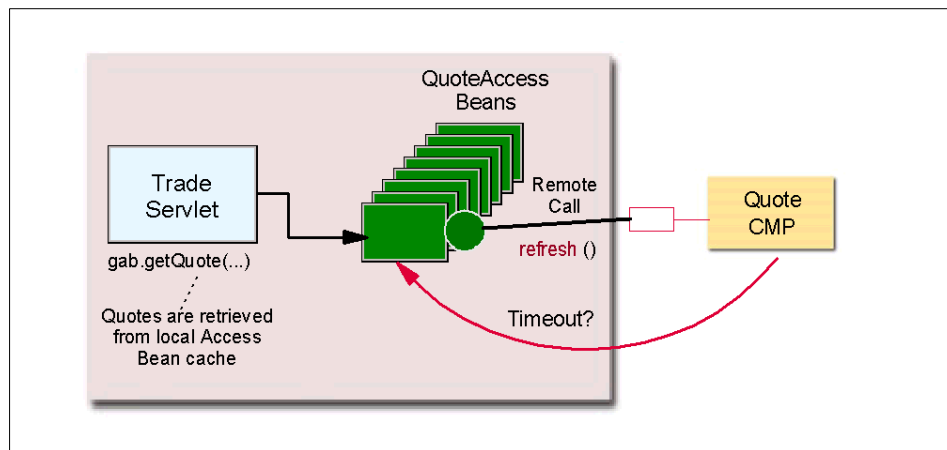


Figure 2-6 Quote Cache example

2.2.3 Outline of our migration task

Here we give a brief outline of how to migrate the Trade2 application from NT to z/OS. The supporting system infrastructure consists of Visual Age 4, one system with WebSphere Application Server Version 4.0 and DB2 on Windows NT, and two systems with WebSphere Application Server and DB2 on z/OS.

The database is split into a configuration part and a business data part. In our example one configuration database is shared by both z/OS systems, while all three systems have separate business data parts.

The business data is transferred directly from the NT-based database to the production system. On the test system test data is needed in the business database; a copy of the contents of the Windows NT-based database is used for that purpose.

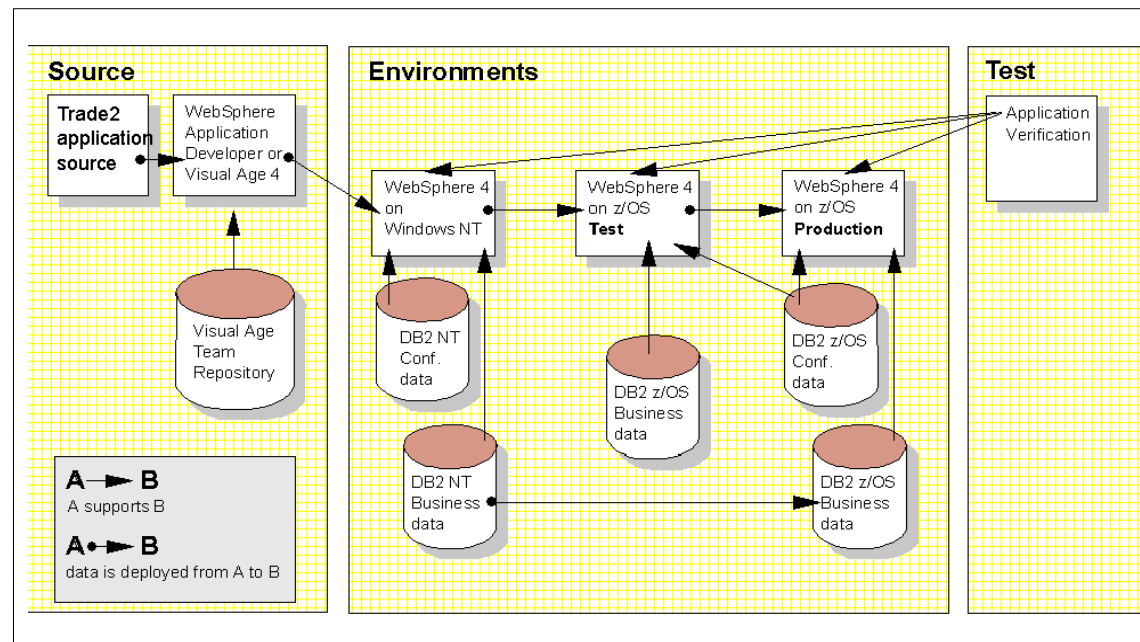


Figure 2-7 Outline of our porting task



Preparing for WebSphere application migration

Before migrating any application, you need to do an initial analysis to identify issues that might lead to problems on the target platform. This chapter presents the topics to consider.

We discuss how to migrate the WebSphere Sample Benchmark Application.

3.1 Application analysis

Before you start porting an application to z/OS, you should carefully analyze certain aspects of the application. Refer to 4.2.1, “Migration check list” on page 51 for an overview of items to consider.

When you port the Trade2 application you might not run into any problems with the mentioned aspects, but you will notice that the absolute URL links within JSPs and HTML pages of Trade2 are a problem if you want to run two instances of this application (i.e., test and production) on one system.

Note: It is always helpful to have access to the original authors of your application. They can help you with code changes that make the application more portable.

As soon as you have a complete picture of the issues that need to be addressed during the migration, you can make a rough estimate of the time needed for the project. Take into account the following steps:

- ▶ Setup of the z/OS infrastructure
- ▶ Setup of clients
- ▶ Changing the code
- ▶ Testing on z/OS
- ▶ Deployment on z/OS
- ▶ Administration (needed to run the porting project)

Availability of source code

It is very likely that the application you want to port needs changes in the source code. The source code also provides an easier way to identify code fragments that cause trouble on z/OS.

In case you cannot get the original source code you might want to generate the source code yourself, using a recompiler.

Platform dependencies

Assumptions about the underlying platform might need to be changed when porting an application. Common issues are ASCII/EBCDIC codepage problems and platform-dependent JNI calls.

Native methods

While pure 100% Java code might not require any changes to run on z/OS, native methods need to be ported to z/OS. For specific information on how to port C and C++ applications to z/OS, see *Porting Applications to the OpenEdition MVS Platform*, GG24-4473.

The way JNI is used in your application may also vary from the way it is used for z/OS, which is described in *Java Programming Guide for OS/390*, SG24-5619.

The J2EE specification strongly discourages the use of native methods out of J2EE applications.

Code pages

There is no central point in z/OS where a system-wide code page is defined how textual data is interpreted. The predominantly used code page is one of the EBCDIC code pages. But z/OS and the hardware are transparent to using other code pages. So it is possible to have data stored on z/OS that is encoded in one of the ASCII code pages. The application has to know about the actual coding. File tagging support introduced with z/OS 1.2 allows the management of ASCII/EBCDIC translation to be handled within the logical file system (LFS) of the UNIX file system support.

Compliance with J2EE specifications

J2EE applications run fine on WebSphere 4 on z/OS as long as they are programmed 100% according to the J2EE 1.2 specifications. IBM WebSphere Application Server for z/OS and OS/390 seems to be more rigid than WebSphere Application Servers on other platforms. It expects that your code does not deviate from the standard.

Remote connectors

In WebSphere 4 on z/OS there are no remote connectors for CICS and IMS yet. If the application makes use of such a connector, the code needs to be changed to use a local connector instead. Refer to 4.2.5, “Externalizing access to non-J2EE connectors” on page 63 and 4.2.8, “JCA connector issues” on page 64.

Security model

If the security model of the application is not supported by IBM WebSphere Application Server for z/OS and OS/390, code changes will be necessary. See 1.4.8, “Security model with IBM WebSphere Application Server for z/OS and OS/390” on page 19 for a discussion of security models in IBM WebSphere Application Server for z/OS and OS/390.

Column names in DB2

Unlike DB2 on distributed platforms, z/OS limits the maximum length of a column name to 18 characters. You need to change the column names in your application if they exceed this limit. Also refer to the following:

- 4.2.6, “Remapping CMP beans to z/OS DB2” on page 63
- 4.2.7, “Changing SQL commands” on page 64
- 4.5.2, “DB2 UDB issues” on page 74

3.2 Setting up the migration infrastructure

In this section we describe the environment we chose to develop, deploy, and run an enterprise application on distributed as well as on centralized platforms. We describe the non-production deployment environment, as it logically lives always close to the development environment.

We first set up a deployment process for distributed platforms to develop an enterprise application and get it up and running in the WebSphere Application Server for distributed platforms. Then we modified this process to deploy the same application to a z/OS environment.

Figure 3-1 on page 35 shows the three parts of the deployment process and the tools we chose to migrate an enterprise application from a distributed environment to z/OS or OS/390.

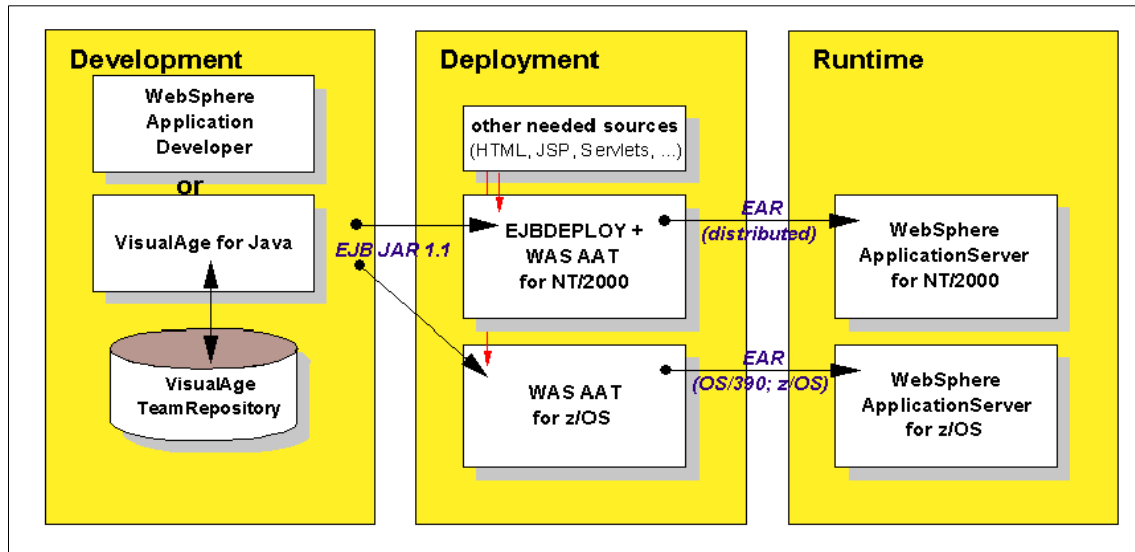


Figure 3-1 Deployment process

1. To develop the EJBs we used VisualAge for Java Version 4.0 and WebSphere Studio Application Developer. In 3.2.1, "The integrated development environment" on page 36 we describe these tools in more detail. The development ends with the generation of the EJB 1.1 JAR file.

Note: At the time of writing, WebSphere Studio Application Developer was not generally available, so we used an early version of this IDE. Therefore, it might behave a little bit differently when it is installed on your workstation.

2. Up to this point the developed EJBs are independent from any platform, which means that the code can be transferred to any platform for further development or deployment. But in this state they are *not* executable. Therefore, the generation of system- and container-specific code, also called deployed code, is needed. This code generation is part of the deployment as well as the creation of an enterprise application archive (EAR), which contains all parts of the application (EJB, JSP, servlets, HTML pages, pictures, etc.). For this part of the process we used the following tools, which are part of the WebSphere Application Server for the specific platforms:
 - Distributed
 - Application Assembling Tool (AAT)
 - EJB Deploy Tool (ejbdeploy)
 - z/OS and OS/390

- Application Assembling Tool (AAT)
 - System Management End User Interface (SMUI)
3. The next step in the deployment process targets the runtime environment itself. To run the whole enterprise application we chose WebSphere Application Server Version 4.0 for the distributed platform (test environment) as well as for the z/OS platform (test and production environment). We describe this runtime environment in more detail in 3.2.2, “The distributed Deploy- and Runtime-Environment” on page 38.

3.2.1 The integrated development environment

To develop EJBs we recommend using an integrated development environment (IDE) because of the assistance the developer gets from these tools (for example: syntax checking and automatic code generation). To develop the EJBs or to make possibly needed changes to the code, among other things, you can choose one of the following IDEs:

- VisualAge for Java Version 4.0 Enterprise Edition
- WebSphere Studio Application Developer

In our project we used both environments successfully for the application migration task. Nevertheless, we only published the WebSphere Studio Application Developer path. It's less cumbersome than the VisualAge for Java Version 4.0 path, and also is state-of-the-art application development.

WebSphere Studio Application Developer (WSAD)

WebSphere Studio Application Developer is the follow-on technology for VisualAge for Java Enterprise Edition. It is designed from the ground up to meet the requirements for J2EE server-side (EJB and servlet) development, deployment, and profiling. These requirements include open standards, Java, XML, Web services, varying levels of integration with other components and ISV products, pluggability, expandability, role-based development, increased usability for all users, and enhanced team support. Using it should decrease time-to-market significantly.

To achieve open standards and interoperability with Independent Software Vendor (ISV) tools, IBM donated huge amounts of the WebSphere Studio and Visual Age code base to the open source community. For more information see:

<http://www.eclipse.org>

WebSphere Studio Application Developer is a combination of existing features, taken from VisualAge for Java and WebSphere Studio, and new features. Figure 3-2 on page 37 offers an overview of the features.

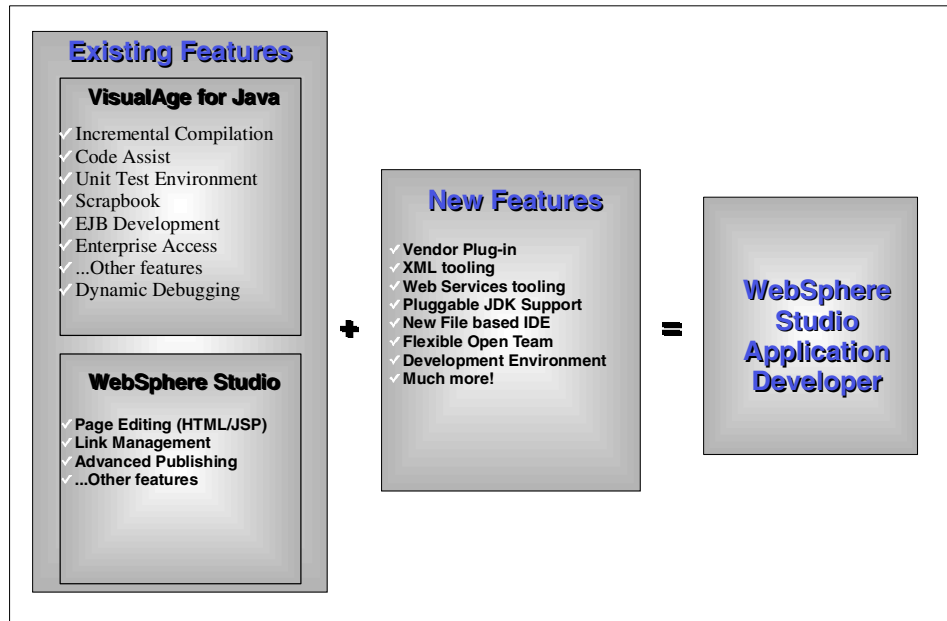


Figure 3-2 Features of WebSphere Studio Application Developer

The following is a list of the standards supported by WebSphere Studio Application Developer:

- ▶ EJB 1.1
- ▶ Servlet 2.2
- ▶ JSP 1.1
- ▶ JRE 1.3
- ▶ Web Services Definition Language (WSDL) 1.1
- ▶ Apache SOAP 2.1
- ▶ XML DTD 1.0 10/2000 Revision
- ▶ XML Namespaces 1/99 Version
- ▶ XML Schema 5/2001 Version
- ▶ HTML 4.01 (other levels should work)
- ▶ CSS2 (Page Designer displays a subset)

Due to the fact that at the time of writing only beta code of WSAD was available, and that the focus of this book is the migration of WebSphere Application to z/OS, we do not want to go into further details about the development environment. In 3.3, “Installing the application” on page 39 we give a short description of what has to be done to deploy an application from WebSphere Studio Application Developer.

Detailed information for WebSphere Studio Application Developer can be found on the Web site:

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

or in the Redbook *Web Services Wizardry with WebSphere Studio Application Developer*, SG24-6292.

3.2.2 The distributed Deploy- and Runtime-Environment

Practical experience shows that in projects where Web or J2EE applications are developed, a separate test environment is installed, which is a mirror of the production environment with regard to the products and versions used. For a z/OS or OS/390 installation this means setting up a separate partition or server as a test environment, which is a normal procedure on such systems.

For our separate test environment on a distributed platform (see also 1.5, “System landscape” on page 20), we chose to set up WebSphere Application Server Advanced Edition Version 4.0.1 on Windows 2000. Detailed information about WebSphere Application Server for distributed platforms can be found at:

<http://www-4.ibm.com/software/webservers/appserv/>

or in the following Redbooks:

- ▶ *WebSphere V4.0 Advanced Edition Handbook*, SG24-6176
- ▶ *WebSphere Version 4 Application Development Handbook*, SG24-6134

We used the following tools, which are part of this WebSphere version, to deploy the application:

- ▶ Application Assembly Tool (AAT)
AAT is a new feature introduced in WebSphere 4.0. It allows packaging and assembling an enterprise application and its modules according to the J2EE specification.
We used this tool to create an enterprise application archive (EAR), which can be deployed into the application server.
- ▶ EJB Deploy Tool

This tool is used to generate deployed code for Enterprise Java Beans and can be started from the command line or as part of the AAT.

The deployed code is *WebSphere-specific* code that connects the J2EE-compliant code with the WebSphere container. If you want to deploy your EJB 1.1 JAR to a J2EE container of another vendor, there would be a similar process to create the deployed code. The deployed code also contains the persistence mapping code for the CMP EJBs. The generation of system-specific code during the deployment process to an application server is according to the Sun J2EE specification.

For information about the installation and deployment of the sample application, see 3.3, “Installing the application” on page 39.

3.3 Installing the application

In this section we describe the deployment process for the chosen application (for further information about the WebSphere sample performance benchmark application, see 2.2.1, “WebSphere sample performance benchmark application” on page 25). This includes the following steps:

- ▶ Import into and export from an Integrated Development Environment (IDE):
 - VisualAge for Java Version 4.0
 - WebSphere Studio Application Developer (WSAD).
- ▶ Code generation for the EJBs
- ▶ Generation of the Enterprise Application Archive (EAR) with the Application Assembly Tool (AAT)
- ▶ Import of the EAR file into the WebSphere Application Server (Windows 2000, z/OS)

3.3.1 Installing the original WebSphere sample performance benchmark application

Before starting to set up the deployment process, we recommend that you install the original application, for the following reasons:

- ▶ The necessary database for WPBS will be created using the installation script.
- ▶ Within the application, the created tables could be populated with fictitious users.
- ▶ You can have a closer look at the application.

You can download WPBS (Trade2 Application) from the following URL:

http://www-4.ibm.com/software/webservers/appserv/wpbs_download.html

To go through the whole deployment process, which is described next, download the following versions of WebSphere sample performance benchmark application:

WebSphere sample performance benchmark application version for WebSphere Application Server Advanced Edition 4.0 (in this book now referred to as Trade2V4) - file: DB2_AE.zip

To install Trade2V4, unzip the downloaded file (DB2_AE.zip) and follow the steps in the installation documentation (Trade_DB2_AE.html).

After you get a little bit familiar with this application and want to step into the deployment process, you have to uninstall the enterprise application from WebSphere Application Server. In the new deployment process, a new enterprise application with the same definitions will be deployed.

To uninstall the existing enterprise application, perform the following steps:

- ▶ Open the WebSphere Administrator's Console.
- ▶ Expand **Nodes > NodeName > Application Servers** and right-click **Default Server**.
- ▶ Choose **Stop**, wait until the information dialog pops up, and click **OK**.
- ▶ Expand **Enterprise Applications**.
- ▶ Right-click **NodeName/TradeSample**.
- ▶ Choose **Remove**, and a dialog box pops up.
- ▶ Click **No**, and another dialog box pops up.
- ▶ Click **Yes**.

3.3.2 Deployment process from WebSphere Studio Application Developer

One of the major improvements in WebSphere Studio Application Developer is the possibility to develop complete enterprise applications (WebApps and EJB applications). For Trade2V4, only an .ear file is used to install the application. Therefore, the following steps show how this application is imported into and exported from WebSphere Studio Application Developer and deployed to WebSphere Application Server Version 4.0.

To perform these steps you need the file TradeSample.ear, which is contained in the file DB2_AE.zip (see 3.3.1, “Installing the original WebSphere sample performance benchmark application” on page 39).

1. Import into WebSphere Studio Application Developer:

- After WSAD has started, choose **File > Import**.
- In the Import window choose **EAR file** and click **Next**.
- In the new window choose the TradeSample.ear file and assign a name for the enterprise application project (for example: WSADTrade).
- Click **Finish**.
- Now the J2EE Perspective of WSAD should look similar to Figure 3-3 on page 41.

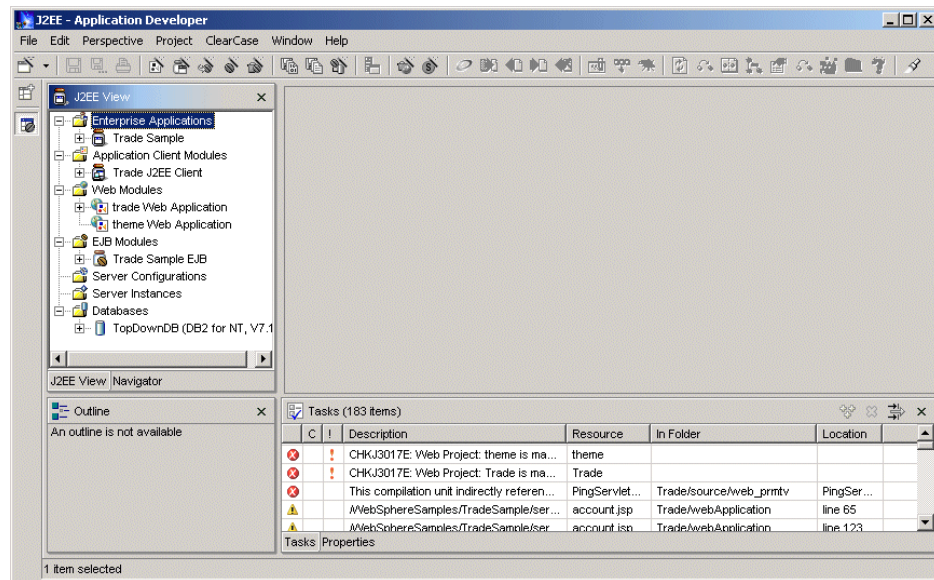


Figure 3-3

If the code for the EJBs has to be changed for any reason, it can be found under **EJB Modules > Trade Sample EJB**.

2. Export from WebSphere Studio Application Developer

After all changes to the code are made, the whole enterprise application can be exported as an .ear file. Because the original .ear file was imported into WebSphere Studio Application Developer, the deployment and mapping information is also available in the enterprise application project and will be automatically added to the exported .ear file.

- Choose **File > Export** from the menu bar.
 - Choose **EAR file** and click **Next**.
 - In the following window:
 - Select the enterprise application project you want to export (for example: WSADTrade).
 - Specify a name for the EAR file (for example: WSADExport.ear).
 - If you also want to export the source code, mark the **Export source files** check box.
 - Click **Finish**.
3. Deployment to WebSphere Application Server Version 4.0
- The exported .ear file now contains all information needed to install an enterprise application in WebSphere Application Server Version 4.0 without using the EJB Deploy Tool or the Application Assembly Tool.
- Make sure the WebSphere administrative server is fully started.
 - Launch the WebSphere Administrator's Console.
 - Expand **WebSphere Administrative Domain**.
 - Right-click **Enterprise Applications** and choose **Install Enterprise Application**.
 - In the next window, select the .ear file you created while exporting the application from WebSphere Studio Application Developer and specify an application name.

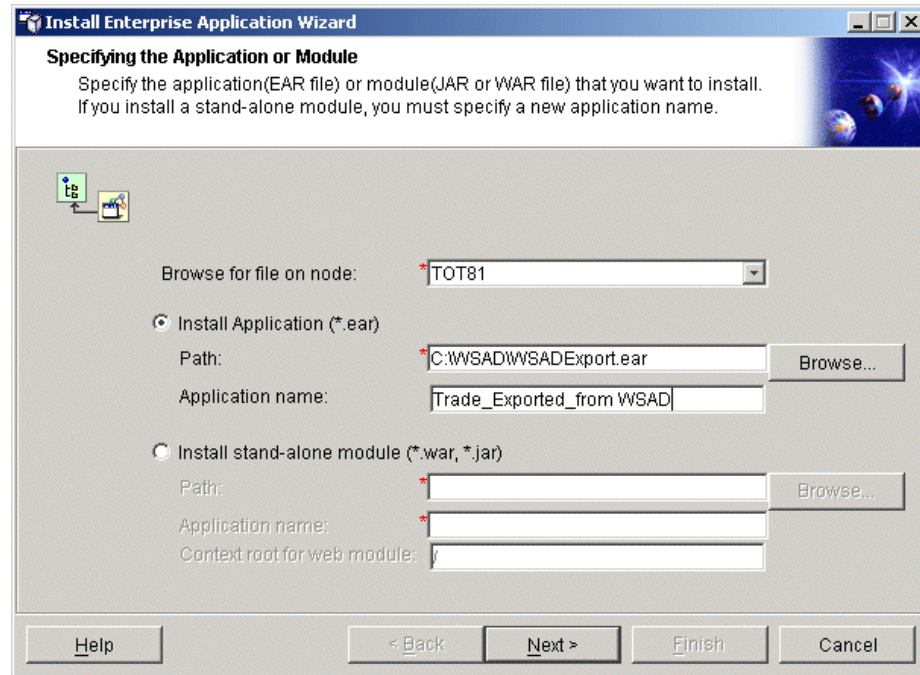


Figure 3-4 Install Enterprise Application Wizard

- Click **Next**.
- Click **Next** until the Selecting Application Server window appears (see Figure 3-5 on page 44).

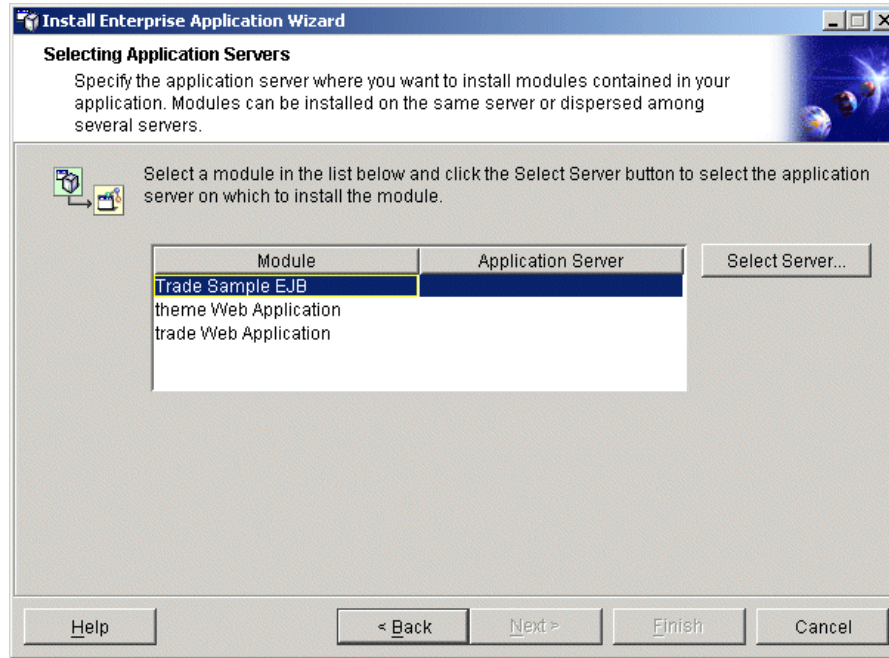


Figure 3-5 Selecting Application Servers

- Select all modules and click **Select Server** to assign an application server (for example: Default Server).
- Click **Next**.
- In the next window click **Finish**.
- Click **No** in the following dialog box.
- Click **OK** in the following information dialog.
- Now the enterprise application is installed in the WebSphere Application Server.
- Right-click **YourNodeName** and select **Regen Webserver Plugin**.
- (Re)start the Default Server.

You can test the newly deployed application by pointing your browser to:

`http://localhost/WebSphereSamples/TradeSample/TradeDocs/index.html`



Migrating an application to z/OS

Once the development and system infrastructure is ready and we know that the application runs on its native environment, it is time to dig into the application. In this chapter we describe the core of the porting work. We discuss how to install the application on IBM WebSphere Application Server for z/OS and OS/390 Version 4. We list some common porting steps you may encounter and how to improve the installation process regarding performance and application installation verification.

For a generic description of the deployment process, see 5.7.1, “Deployment principles” on page 121.

4.1 Minimum migration steps for Trade2

In this section we explain actions necessary to port an application that is currently running on an NT-based WebSphere server to WebSphere on z/OS.

4.1.1 Overview

The software needed on your client system:

- Application Assembly Tool (AAT)
- System Management Enhanced User Interface (SMUI)

Preconditions:

- Application is deployed to WebSphere 4 on NT (.ear file is accessible).
- The database for the application on z/OS is set up.

Steps:

1. Import the .ear file for NT into AAT.
2. Update the EJB links.
3. Add initial values where needed.
4. Generate the new .ear file.
5. Deploy to WebSphere on z/OS.
6. Activate the application on z/OS.

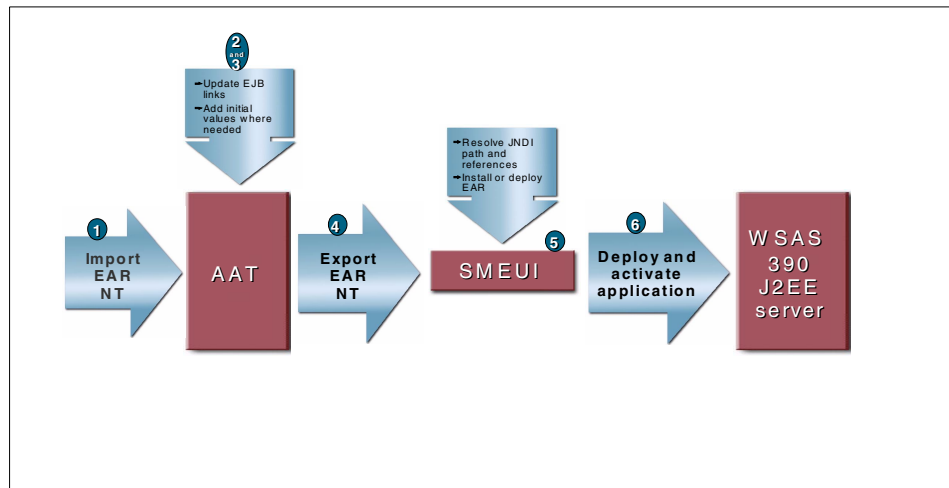


Figure 4-1 Minimum migration steps

Note: the original Trade2 makes use of absolute paths in its design. In order to be able to exercise test and production environments using the same application, we adapted Trade2 to use relative paths.

4.1.2 Install the Application Assembly Tool (AAT) for z/OS

According to the J2EE specifications, the application must be deployed to WebSphere in the “ear” format. However, the .ear file also contains some platform-specific information. To convert the .ear file that is deployed on an NT-based WebSphere server into an .ear file deployable to a z/OS-based WebSphere Server, you can use the Application Assembly tool (AAT) for z/OS, which runs on Windows. The AAT can be downloaded from the IBM Web site: http://www-4.ibm.com/software/webservers/appserv/zos_os390/index.html

4.1.3 Install the System Management Enhanced User Interface

The SMUI is a Windows-based user interface which can be used to deploy a new application to WebSphere on z/OS. Therefore, the SMUI needs to be installed on your Windows client.

The install file of the SMUI (bboninst.exe) comes with WebSphere for z/OS. You can download the code from the /usr/lpp/WebSphere/bin/ directory.

4.1.4 Import the .ear file and the helper classes

We assume that the Web application has been successfully deployed to the NT-based WebSphere server using an .ear file. This .ear file needs to be imported into the AAT for z/OS.

Right-click **applications** in the left pane of the AAT. Choose **import**. Browse to your .ear file and import it.

When you import the file it will be automatically validated. In case of the Trade2 application you import TradeSample.ear.

While importing this file, a pop-up panel named “Confirm file import” opens. Click **Yes** to import the helper runtime class (ivjejb35.jar). You can also click Help to get an explanation of what will happen when you confirm the file import. You will then get the message BB094001I Application Trade Sample was imported from file c:\trade2\TradeSample.ear.

4.1.5 Update the EJB links

EJBs can be referenced by other modules of the application. An EJB link is the name by which it can be called from other modules. This is similar to the link between JNDI Names and the J2EE resources created in the SMUI.

You need to redefine the EJB links even if they were already defined on NT:

1. Right-click the EJB and select **Modify**.
2. For each EBJ select the **EJBs** tab and choose an appropriate link in the link column for every reference. In the Trade2 application this has to be done for the two session beans KeySequence and TradeSession.
3. Right-click **KeySequence** and select **Modify**.
4. Click the **EJBs** tab. It lists exactly one reference named ejb/KeyEntity.
5. Activate a drop-down choice by clicking in the empty field in the column Link.
6. Select the link **KeysEntityBean**.
7. Click the disk icon to save this modification. You must either save this modification or cancel the modification by pressing Esc to be able to select another bean.

To modify the remaining bean, right-click **TradeSession** and select **Modify**. Switch to the **EJBs** tab, which lists a number of references. Clicking in any blank field in the Link column shows a drop-down choice with exactly one entry. Set this entry for all references and save the modification by selecting the disk icon.

Tip: It is not obvious that you have to scroll down this list and complete all links.

4.1.6 Add initial values where needed

Make sure that every init parameter of every servlet has an initial value. Otherwise the System Management Enhanced User Interface (SMUI) will complain.

For the Trade2 application you need to supply the credentials for the database access to the TradeAppServlet. Expand the tree in the left pane to **Applications/Trade Sample/Web Apps/trade Web Application/Web Components/TradeappServlet**. (You can also expand the entire tree by clicking the round blue + icon). Right-click this servlet and select **Modify**. Change to the **Parameters** tab on the right side of the screen and select the row **JDBC_PWD**. Click **Modify** to open a dialog, supply the password as a value, and confirm by clicking **OK**. Supply the user ID by applying the same steps in the JDBC_UID row. Save the changes by selecting the disk icon.

4.1.7 Generate the new .ear file

Validate the application by opening the drop-down menu **Build** and selecting **Validate**. The success of the validation is indicated by the message `Application can be deployed` in the message window at the bottom of the screen.

Now select **Build/deploy** to generate the deployable code. During deployment, the message `BBO94101I` appears. Accept the restrictions and allow deployment to continue. The message `Application was deployed` informs about successful generation of the code.

Now open the drop-down menu **Selected** and click **Export** to start the process of exporting. You are prompted for an appropriate filename and path. Confirm your selection by clicking **OK**. After the generation the message `Application was exported to .ear file` indicates the success of the operation. The generated .ear file can be deployed to WebSphere Application Server by using the SMUI.

Tip: The `EJBdeploy` checkbox must be checked. Even if this is the default, it remembers if someone has unchecked it before.

4.1.8 Set up the application database tables on z/OS

You need to create all database tables necessary to run your application. If you want to use DB2 on z/OS and have database scripts for another DB2 on Windows NT, you will have to adopt those scripts.

We found that the IBM WebSphere Application Server for z/OS and OS/390 calls DB2 using the server region identity. In order to bypass this porting difficulty, we chose to prefix the DB2 objects with the server instance identity. See “DB2 DDL job for Trade2 database” on page 182. Likewise, given the Trade2 application design, we created DB2 aliases for the same objects using the identity required by servlet `TradeAppServlet`. DB2 on z/OS also requires the addition of a DB2 table specifying the primary key, which is reflected in our sample JCL.

4.1.9 Deploy to WebSphere Application Server on z/OS

Note: How to automate J2EE server definitions is described in 5.3.2, “Scripted configuration” on page 105.

Make sure to define all data sources to include all resource references that need to be resolved, including JDBC DataSources, JMS Administered Objects, JCA Connectors, Java Mail etc. The data sources need to be defined before the .ear file is imported. This can be done first in the same conversation as adding the .ear file, to save time.

Open the SMUI and right-click **Conversations** in the left pane. Select **Add** and supply a name for the conversation in the right pane of the window. Click the disk icon (Save) to add the conversation.

Tip: When you create new conversations, the old ones become useless, so once in a while you should delete them. During this housekeeping process you might also clean up all the current.env files in the backup directory.

Select your conversation and expand the tree on the left pane to show your desired J2EE server. Right-click this server and select **Install J2EE application**. You are prompted to supply the .ear file. After browsing to the .ear file that you exported from the AAT, confirm by clicking **OK**.

After the .ear file is read a window pops up which shows the Reference and Resource Resolution. All EJBs are listed in the left pane of the window. Every EJB which is ready for deployment is marked with a green check mark. For the Trade2 application the JNDI names have to be added and in some cases J2EE resources have to be added to make the EJBs ready for deployment.

To add the JNDI name to an EJB, select the EJB in the left pane. Select the **EJB** tab in the right pane. Click **Set Default JNDI Path & Name**. Now a green check mark in the headline of the tab shows the path and name which are supplied. In the Trade2 application, this step needs to be done for all beans.

To add a J2EE resource to an EJB select the EJB in the left pane. Select the **J2EE Resource** tab in the right pane. In the J2EE Resource column, click an empty box to activate a drop-down choice. Select the appropriate resource.

In the Trade2 application this step needs to be done for most of the beans. In the drop-down menu there is always just one choice, so just use this.

You have to define the JNDI names and the resources for the items included in the JAR files as well.

After you confirmed all the changes by clicking **OK**, the .ear file is automatically transported via ftp to the server.

Tip: Depending on the size of your .ear file and the number of beans in it the deployment process can be very time- and CPU-consuming. Don't deploy when the system resources are needed for other work. And be patient.

4.1.10 Activate the application on z/OS

If all EJBs are checked with a green mark, the final steps can be taken.

Highlight your conversation and from the Build menu item select **Validate** -> **Commit** -> **Complete** -> **Activate**. The activation is recycling the affected server instance and registering all items to the naming services where necessary. The application is now ready for use.

4.2 Common migration steps

This section investigates the porting issues that we did not encounter with Trade2, but which can appear in other porting projects.

4.2.1 Migration check list

The Trade2 application was already well described, so we easily found all we needed to prepare and run this application in our project. This is not always the case. Before starting any porting project for any software, a wise practice is to evaluate the structure and details of the application. Here is a list of questions that need to be resolved before starting anything:

Java-related questions

- ▶ What classloader mode was selected?
- ▶ On what level of Java SDK runs the application?
- ▶ Is the source of the application available?
- ▶ Does it rely on a framework?
- ▶ What connectors does it use? How do they connect to the backend system? Is the connector access security based on programming or system authentication?
- ▶ What is the structure of the application (JSPs, servlets, Java Beans, EJBs)?
- ▶ What kind of EJBs does it contain (stateless, stateful, BMP, CMP)?

- ▶ Are there associations or inheritance between EJBs?
- ▶ Are there property files to customize?
- ▶ Is there an XMLCONFIG file for the Java 2 Enterprise Edition server where the application runs?

IDE questions

- ▶ What is the IDE used for development?
- ▶ What is the IDE level?
- ▶ What is the Java level used by the IDE?
- ▶ Are CMP/DB2 mappings defined inside the IDE?

Tip: It is definitely a good practice to store the development tools with the application source code. When you have to work with the source code after some years you are lucky if you can import it into new state-of-the-art tools. If this doesn't work, you will be able to utilize the tools that have been used to build the application.

WebSphere questions

- ▶ What is the level of the IBM WebSphere Application Server Advanced Edition?
- ▶ Does it host the application on a single server or in a WebSphere cluster?
- ▶ If the original backend is on a distributed platform, what is its level?
- ▶ If a database is used, can the Data Definition Language (DDL) statements for creating the database be provided?
- ▶ Is there a document with the deployment procedure?
- ▶ Does the application access any backend system using a non-J2EE compliant connector?

4.2.2 Packaging the application

It is common for the application development team to be split between two different entities (for example, service company vs. customer company, distributed programmers vs. porting team) so the developer from one entity may send the application to a programmer from the other entity who will run the porting project. This second programmer will receive the code in his IDE, and eventually modify and redeploy the application.

In such a case, there are many ways to transfer the application between the two IDEs. Most of them are confusing and miss some part of the application.

The most efficient way to publish a Java 2 Enterprise Edition application is to send the following:

- ▶ The WebSphere Studio archive file
- ▶ The VisualAge for Java repository file of the project after having saved database schemas and having versioned the whole project
- ▶ The .ear file that was used on the distributed platform with the corresponding DDLs

4.2.3 JNDI and namespace fundamentals

To understand how the application will use the namespace to look up EJBs we first describe the Java Native Directory Interface (JNDI) namespace:

What is a JNDI namespace?

A JNDI namespace can be thought of as a specialized environment that identifies and describes the attributes of an application's enterprise java beans (EJBs). In this specialized environment reside the names by which EJBs are referenced by other code within an application. Typically, this other “code” is servlets and other EJBs that make up a J2EE application.

According to the Enterprise JavaBeans Specification, v1.1, every EJB has an associated environment that is declared in the EJB's deployment descriptor. Recall that an EJB's deployment descriptor is a special “profile” that describes the attributes of an EJB. This descriptive information, also known as “meta data”, is used by application assembly and deployment tools such as IBM's Application Assembly Tool (AAT) and Systems Management Extended User Interface (SMUI) to install EJBs into a container.

Furthermore, the container uses the deployment descriptor meta data to manage an EJB during runtime. Since the environment description is outside of the EJB's source code, an EJB can be customized without ever changing the EJB's code. Therefore, the platform transparency of EJBs can be maintained while only the deployment information needs to be altered for each target platform.

JNDI naming under WebSphere z/OS

So what does all this have to do with the JNDI namespace? One of the attributes in an EJB's deployment descriptor identifies the name of the EJB itself. The name of an EJB is used by application code to find or look up an EJB in a namespace. When a deployment tool installs an EJB into a WebSphere z/OS container, it gathers information about each EJB in an application and determines where in the JNDI namespace the home for that bean should be stored.

Then, as part of the application installation process, references to the EJB homes are bound into the JNDI namespace with the supplied name. Since WebSphere z/OS utilizes an LDAP database as the backing store for its JNDI implementation, an LDAP entry is created for the home references. Therefore, when an application such as a servlet or an access bean wants to create an instance of an EJB, it simply looks up the name of the desired EJB in the namespace created by the deployment process.

As shown in Figure 4-2 on page 55, JNDI naming under WebSphere z/OS can be summarized with the following fundamentals:

1. All EJBs have a deployment descriptor that, among other things, contains the name of the EJB.
2. During EJB deployment into a server, the SMUI determines the name under which the home reference should be bound in the JNDI namespace. This namespace is backed in LDAP.
3. A global environment or initial context is the required starting point for locating EJBs in a JNDI namespace.
4. During bean development, if EJB developers reference an external EJB in their runtime code, they must make that evident by defining an <ejb-ref> tag with /ejb/home-name in the EJB's deployment descriptor. This is the string that gets passed to the lookup() method to locate an external EJB under the java:comp JNDI context.

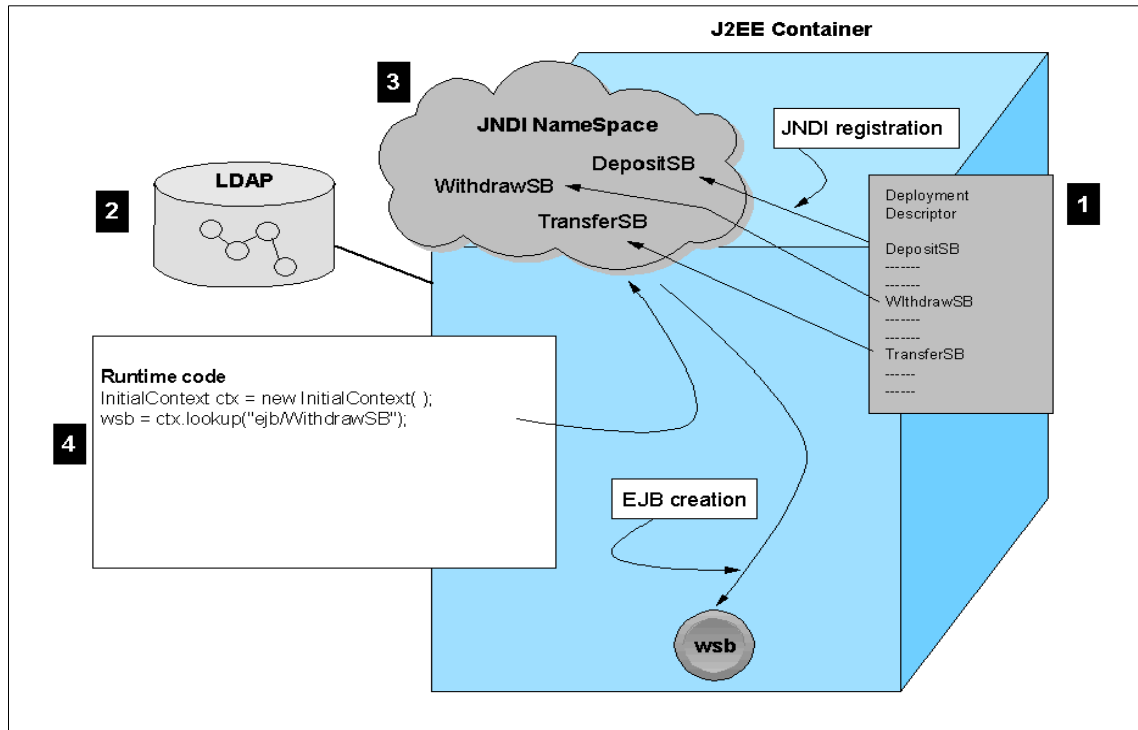


Figure 4-2 JNDI namespace fundamentals

Creating instance of EJBs during runtime

To access an EJB's registered deployment descriptor information, an application must first create an instance of the initial context. Once this instance is obtained, the application can look up the home or factory used to create an instance of the target EJB.

As you can see in Figure 4-2, the J2EE container's scope of control encompasses the deployment descriptor, the JNDI namespace, the application, and the instance of the target EJB. When an EJB is first accessed, for example when the first method is driven on an EJB, the container loads the EJB's meta data and establishes its `java:comp` namespace.

Creating EJB instances with the `java:logical` reference

WebSphere z/OS complies with the Enterprise JavaBeans Specification, v1.1 by providing a JNDI naming implementation based on the Lightweight Directory Access Protocol (LDAP). LDAP provides a directory structure for efficiently locating names in a namespace. When you deploy a J2EE application (servlets and EJBs) into a WebSphere z/OS container with the SMUI, the Systems

Management and Naming components of the WebSphere z/OS runtime work together to register the EJB home instances into the namespace. The registration process uses the deployment descriptor located in the .ear file to generate an indirect object reference (IOR) for the home interface, which is stored in the LDAP database for each EJB.

Even though all EJB home references are stored in LDAP under WebSphere z/OS, EJB providers can designate a logical means for creating EJB instances during runtime. The `java:name` for a home is a logical name which an EJB provider specifies during runtime to create instances of a certain type of bean. Therefore, the EJB deployer must map the logical home to the physical home so that at runtime the lookup succeeds. Since EJB providers at development time do not know where in the namespace the home reference is going to be found, they can use the `<ejb-ref>` XML tag in the `ejb-jar.xml` file to declare that the EJB contains runtime code that is going to look up and use a home.

Therefore, the logical notation, `java:comp/env/ejb/home-name`, is passed on the lookup method call. To ensure a successful lookup, the deployer (the person responsible for creating the server with the SMUI) must make sure that the declared `java:comp` name returns the EJB's home reference wherever it is located. The WebSphere z/OS container is ultimately responsible for determining how to return the home reference. You should understand that the `java:namespace` is a container-use only namespace that provides quick retrieval of important bean-related objects during the container-managed life cycle of an EJB.

Anatomy of a WebSphere Application server LDAP entry

Example 4-1 shows the fully-qualified JNDI namespace reference for the WithdrawSB EJB represented by the following LDAP entry:

Example 4-1 A JNDI-LDAP entry

[illegible]

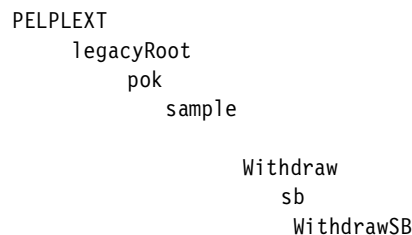
```
998a70000086000000017c0a8008cb7057b6dd236b8080000086000000017c0a8008cb7057b6
dd89b062a0000086000000017c0a8008cc2d5c9c4000000010000000100000001000000db70
57b6cd73998a70000086000000017c0a8008cb7057b6dd236b8080000086000000017c0a8008
cb7057b6dd89b062a0000086000000017c0a8008c6d6d88969485d686c8969485a2000000001
300090000000300000000000000080000000049424d0049424d0400000007000500010200000
0000000010000001c00000000100204170000000110020417100204170000000110020417000
0000000002000001010000000011503230422e4554502e49424d2e434f4d0000015b30000017
5000000b4d6d9c2d200000002c4d5e2d2e2d9e500b70216a75e04898e0000081400000140c0a
8008c020000000000000000000040d9d4c97a84924b848195a292854b83994b9385818481979
7934b97999682854ba2824bd799968285e2c2c89694857af0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f
0f0000000003d84924b848195a292854b83994b93858184819797934b97999682854ba2824bc
5d1e2d9859496a385e2a381a3859385a2a2d799968285e2c2c896948500000000000000b9000
0000000000001000000000000000000000000000000000009db7057b6cd73998a70000086
000000017c0a8008cb7057b6dd236b8080000086000000017c0a8008cb7057b6dd89b062a000
0086000000017c0a8008cc2d5c9c4000000010000000100000001000000db7057b6cd73998a
70000086000000017c0a8008cb7057b6dd236b8080000086000000017c0a8008cb7057b6dd89
b062a0000086000000017c0a8008c6d6d88969485d686c8969485a2000000000000000000000
5000000000000080000000049424d0049424d0400000007000500010200000049424d01000
00010d7f2f0c24bc5e3d74bc9c2d44bc3d6d449424d0000000000000000010000001c0000000
0100204170000000110020417100204170000000110020417
```

To understand a little more about how JNDI naming is implemented under WebSphere z/OS, let us break down the preceding LDAP entry into its constituent parts.

The following directory path, which is entered into the SMUI by the deployer, maps the WithdrawSB EJB:

```
/PELPLEXT/legacyRoot/pok/sample//Withdraw/sb/WithdrawSB
```

This path can be represented graphically by the following diagram:



PELPLEXT: This entry represents the sysplex on which the WebSphere z/OS application server is running.

LegacyRoot: This entry represents the root anchor for the remaining segments of the LDAP entry.

pok/sample//Withdraw/sb/WithdrawSB: This is the fully-qualified directory path which maps to the WithdrawSB EJB home. This segment typically maps to a package name in VisualAge for Java or a source path in the WebSphere Studio Application Development (WSAD) tool.

The final significant segment of the LDAP entry for an EJB is the IOR. An IOR is a CORBA-defined construct that uniquely identifies an object in a namespace. When your code performs a lookup on the initial context, the WebSphere z/OS naming service returns a reference to the target EJB's home. Once this reference to the EJB's home is returned, application code must convert it to a reference to an EJB home interface through a process called narrowing. With the reference to the EJB home, the application can create an instance of the EJB by driving the `create()` or `findByPrimaryKey()` methods on the EJB home.

How to look up an EJB

The following code sample shows the typical technique for establishing an initial context, performing a JNDI lookup on the context, narrowing the home reference, and creating an instance of a WithdrawSB EJB:

```
// Get the initial context
InitialContext ctx = new InitialContext();

// Look up the WithdrawSB EJB in the JNDI namespace
Object obj = ctx.lookup("java:comp/env/jdbc/BankApp/WithdrawSB");

// Narrow the obj reference to an EJB home reference
wsbRef = (pok.sample.BankApp.Withdraw.sb.WithdrawSB)
portableRemoteObject.narrow(obj,
pok.sample.BankApp.Withdraw.sb.WithdrawSB.class);

// Create an instance of the WithdrawSB EJB
wsb = wsbRef.create();

    or

Wsb = wsbRef.findByPrimaryKey(<key>);
```

Note: Application development tools such as IBM VisualAge for Java and WebSphere Studio Application Development provide wizards for generating access beans that contain code for performing the initial context lookup and instantiation of EJBs. By using these wizards, you do not have to code the JNDI lookup operations yourself. Instead, all you have to do is create an instance of the access bean and all the JNDI naming resolution is done in the access bean.

Locating datasources under WebSphere z/OS

WebSphere z/OS does not store datasource entries in LDAP; rather, to ensure faster access to database resources, these entries are stored exclusively in a JNDI cache. Therefore, datasource lookup strings for J2EE applications on WebSphere z/OS must begin with the `java:comp/env` string. For example, the lookup string for the datasource that might support the banking application referred to earlier could be `java:comp/env/jdbc/BankAppDataSource`.

Mapping EJB resource reference names to J2EE datasources

During application assembly, you specify datasource reference names or lookup names for EJBs on the Resources tab of the AAT. In the example we are using, the Reference name for the banking application would be `jdbc/BankAppDataSource`.

When you deploy this application into a WebSphere z/OS server, you must define a J2EE resource and J2EE resource instance that represent a datasource that your EJBs access. Then, when you import the `.ear` file for your application into the SMUI, you map the JNDI name of the datasource that each EJB accesses to the J2EE resource specified on the SMUI. This is done through the Reference and Resource Resolution panel displayed by the SMUI during the deployment process. When the SMUI creates the container for the server, the datasource name, in this case `jdbc/BankAppDataSource`, is registered in the JNDI namespace under the `java:comp/env` context.

To perform a lookup on this datasource during runtime, provide the following code in your servlets or access beans:

```
// Get the initial context
InitialContext ctx = new InitialContext();

// Look up the datasource in the JNDI namespace
DataSource ds = ctx.lookup("java:comp/env/jdbc/DataSource");
```

Instead of hardcoding the datasource lookup string in the code itself, you can specify the string as a variable in a properties file that you can read during servlet initialization or access bean construction. This allows you to change the datasource name if necessary without regenerating the jar file that contains the access code. Several java classes, such as the Properties and ResourceBundle classes, provide methods for managing and accessing properties files.

Figure 4-3 shows the interaction between a lookup in application code on a JDBC resource and how that lookup resolves to a reference to the actual database mapped by that reference.

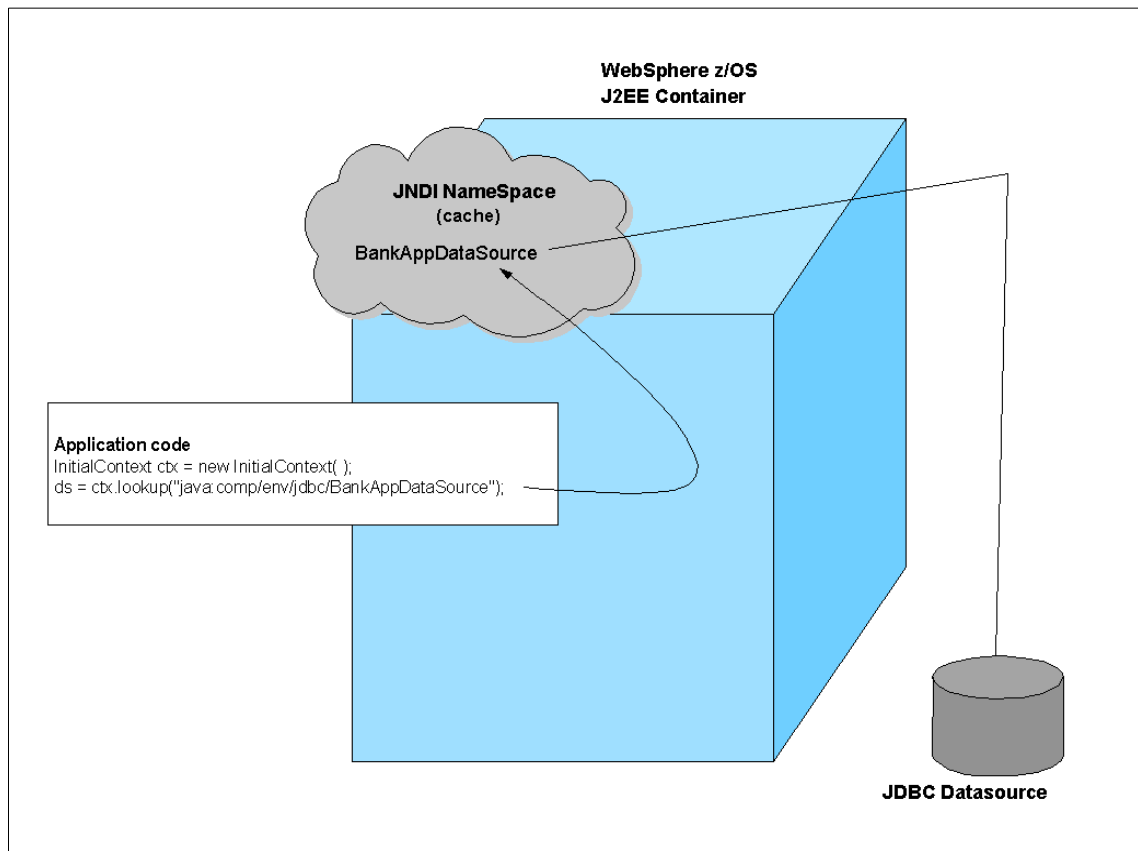


Figure 4-3 Datasource lookup in the JNDI namespace

EJBs located from local namespace within Trade2

Trade2 application components use the local JNDI name space when looking for EJB references.

The Initial Context lookup is done using a URL like `java:env/comp/ejb/Trade`.

Under the AAT, in the EJB folder, the `ejb/Trade` reference was created and resolved to one of the EJBs included in the `.ear` file.

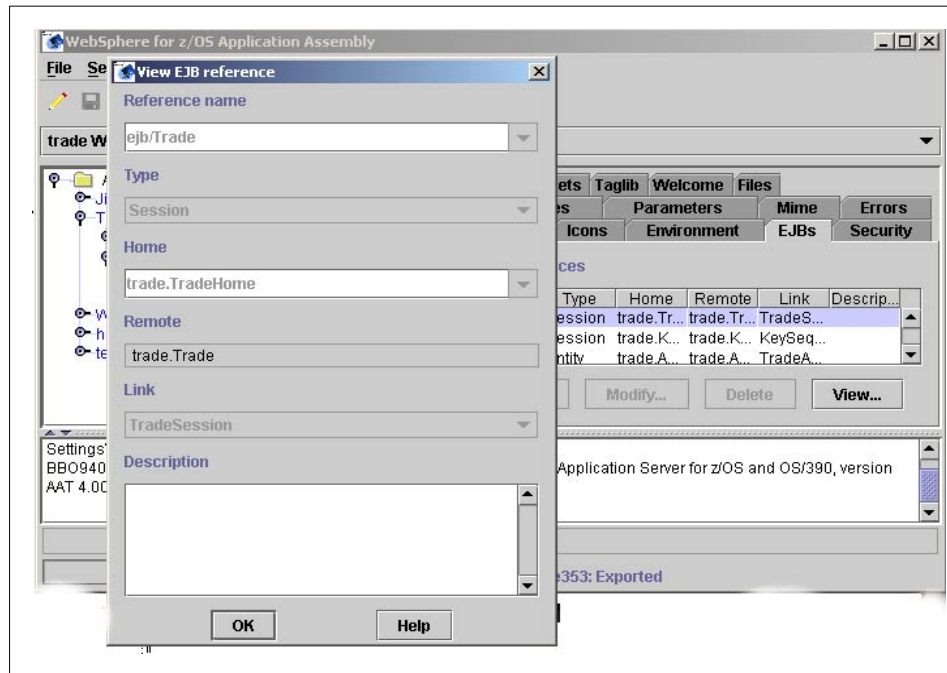


Figure 4-4 AAT EJB references

`java:comp/env` tells the container to use the local namespace to resolve a reference instead of looking in the LDAP global name space. A local lookup searches the reference to `ejb/Trade` in the J2EE servers address space.

The issue is that the EJB to be referenced needs to be in the same `.ear` file in the same J2EE server. This restriction applies only for IBM WebSphere Application Server for z/OS and OS/390. In IBM WebSphere Application Server Advanced Edition AAT you can refer to an EJB which was/is/will be deployed in another J2EE application server.

The solution is to modify the code not to use the `java:comp/env` local name space, so the lookup will go to LDAP for getting the reference to the EJB. You will need then to check that the JNDI name (specified during J2EE installation in the System Management Enhanced User Interface) of the EJB is the same as looked for in the code. A good option would be to create an environment variable for this name in order to externalize it.

For a deeper discussion of name spaces, read Chapter 13 of *WebSphere Version 4 Application Development Handbook*, SG24-6134.

4.2.4 Initial Context factory

The J2EE specification recommends that code accessing JNDI obtain a reference to a JNDI Initial Context object using the default constructor, with no arguments. Furthermore, the specification requires that the container provides an environment in which a valid Initial Context will be obtained by using the default constructor.

IBM WebSphere Application Server Advanced Edition Version 4.0 and Version 3.5, as well as IBM WebSphere Application Server for z/OS and OS/390 Version 4 fulfill this requirement, meaning that an Initial Context can be simply obtained using the code:

```
import javax.naming.InitialContext;
...
InitialContext context =new InitialContext();
```

This code will function correctly in EJBs, servlets, JSPs and application clients running in the appropriate WebSphere container.

Earlier versions of IBM WebSphere Application Server Advanced Edition required you to specify the Initial Context factory class and the naming service provider URL in a hash table or properties object to the Initial Context constructor. This approach will still work, but you should be aware that the factory class has changed in Version 4.0:

- ▶ `com.ibm.ejs.ns.jndi.CNInitialContextFactory`
in IBM WebSphere Application Server Advanced Edition Version 2/3
- ▶ `com.ibm.ws.naming.Idap.WsnLdapInitialContextFactory`
in IBM WebSphere Application Server for z/OS and OS/390 4 before PTF 18
- ▶ `com.ibm.websphere.naming.WsnInitialContextFactory`
in IBM WebSphere Application Server Advanced Edition Version 4 and IBM WebSphere Application Server for z/OS and OS/390 Version 4 after PTF 18

The factory class supported by earlier versions of IBM WebSphere Application Server Advanced Edition is still provided in IBM WebSphere Application Server Advanced Edition Version 4 for backwards compatibility with old code.

You may encounter some code containing, hardcoded, an old factory class which cannot be found in IBM WebSphere Application Server for z/OS and OS/390 Version 4.

The naming service provider URL also needs to be changed.

4.2.5 Externalizing access to non-J2EE connectors

Although you can be lucky and have to deal only with full J2EE applications, you may encounter applications accessing non-J2EE resources. This kind of resource usually uses URLs to locate the server resources. You should externalize these URLs in an environment variable to make them more flexible.

4.2.6 Remapping CMP beans to z/OS DB2

Under IBM WebSphere Application Server Advanced Edition, CMP EJBs use databases for persistence. In order to manage this persistence, the deployment tool is in charge to generate persistence classes containing the JDBC calls used during EJB life. Although IBM WebSphere Application Server Advanced Edition can generate these persistence classes, there is the option in VisualAge for Java to generate these classes and use them without using the AAT.

Until IBM WebSphere Application Server Advanced Edition Version 3.5, the only way to change the default CMP mapping (the name of the EJB is the name of the table and the name of the EJB persistent field is the name of the database column) was to use the VAJ schema editor to specify your own unique mappings. Then the application needed to be deployed and exported as a deployed application from VisualAge for Java. With the IBM WebSphere Application Server Advanced Edition 3.5 administrative client, the deployer did not have to check the “Deploy” check box for the VAJ persisters to be used.

In IBM WebSphere Application Server Advanced Edition Version 4, there is a new possibility to create XML files describing the persistence rules, but this option does not exist in IBM WebSphere Application Server for z/OS and OS/390 Version 4.

So, here are two issues with IBM WebSphere Application Server for z/OS and OS/390 Version 4:

- ▶ EJB names and fields do not fit in DB2/390 constraints (table name length). See 4.5.2, “DB2 UDB issues” on page 74.
- ▶ Persistence classes were reworked under VisualAge for Java and you need to keep this work available.

To change the persistence classes in order to fit in DB2/390 constraints you will have to change the database mapping under VisualAge for Java Schema Browser.

You will have to export EJB code as a “deployed jar”.

To use this .jar file with the correct EJB/DB2 mapping you need to disable the “Enable ejb deploy” option of the AAT. Otherwise, the AAT will again deploy the CMP EJB creating the default persistence.

Because the “Enable EJB deploy” option is disabled, you can now use the AAT to create an .ear file without overriding the VisualAge for Java persistence. The .ear file persistence classes will use the DB2 tables and column names you have selected previously when working with the VAJ Persistence Builder.

For a more complete description of CMP database mapping, see *Enterprise JavaBeans for z/OS and OS/390*, SG24-6283.

4.2.7 Changing SQL commands

The z/OS DB2 restriction does not only apply to CMP beans! Any SQL code and DDL is to be checked in order to avoid z/OS DB2 limitations.

See 4.5.2, “DB2 UDB issues” on page 74 for limitations we identified.

4.2.8 JCA connector issues

CCF (Common Connector Framework), which was used in IBM WebSphere Application Server for z/OS and OS/390 Version 3.5 servlets, is not supported in EJBs. In IBM WebSphere Application Server for z/OS and OS/390 Version 4 or higher, CCF is allowed only inside servlets.

As a replacement, JCA connectors are supported both for servlets and EJBs, but it is strongly recommended that if a servlet needs to access a backend system, a stateless session bean using a JCA connector should be used.

At the time of writing, the only JCA connectors supported by IBM WebSphere Application Server for z/OS and OS/390 were the IBM WebSphere Application Server for z/OS and OS/390 CICS EXCI and IMS APPC connectors, but they were still beta versions.

These connectors:

- ▶ Cannot be tested from IBM WebSphere Application Server Advanced Edition or IDE environment; they are z/OS-specific.
- ▶ Have some implementation differences compared to backend delivered connectors.
- ▶ Offer full JCA support.
- ▶ Support two-phase commit (2PC) using Resource Recovery Services (RRS).

At the time of writing, the connectors supported on IBM WebSphere Application Server Advanced Edition Version 4 were the backend-delivered connectors (beta):

- ▶ CICS Transaction Gateway for CICS Transaction Server access
- ▶ IMS Connect for IMS access

However, the Java Connector Architecture (JCA) Common Client Interface (CCI) layer hides the differences on the backend connector implementation. So you should see no or little difference from a programming point of view when switching from IBM WebSphere Application Server for z/OS and OS/390 CICS EXCI connector to CICS Transaction Gateway. CCI APIs should be the same regardless of the connector, and the developer should only use CCI APIs. The final mapping from program CCI classes to connector-delivered interfaces is to be done by the connector-provided CCI implementation classes (hidden to the programmer).

You can implement backend access using command beans. It is a good practice to isolate backend access code from pure business logic. Maintenance and debugging are easier and reuse is more effective than spraying backend access across a lot of business-oriented classes.

This has major consequences:

- ▶ If the application you want to move to z/OS runs on IBM WebSphere Application Server Advanced Edition 3.5, then you will have to migrate from CCF connectors to JCA connectors.
- ▶ If you have been using IBM WebSphere Application Server for z/OS and OS/390 3.5 to access CICS or IMS, you will have to consider using IBM WebSphere Application Server for z/OS and OS/390 JCA connectors installation. Also closely check the differences in implementation between “old” connectors and those provided by WebSphere.
- ▶ You may consider to test the application with IBM WebSphere Application Server for z/OS and OS/390 JCA connectors and use the subsystem connectors when available.

4.2.9 Framework and connector issues

Before starting the porting project, you should examine which frameworks and connectors are used by the application. Then ensure they have already run on z/OS with the same level of support (2PC, security, etc.). If this is not the case, you should schedule some unit tests of this code. Don't test at the same time with

connector code or application code. For example, we found that the IBM WebSphere Application Server Advanced Edition command framework was not delivered with IBM WebSphere Application Server for z/OS and OS/390; we needed to install it manually.

4.2.10 Restrictions of the z/OS EJB runtime

Here is a list of functions that are available on IBM WebSphere Application Server Advanced Edition Version 4.0.1 but not on IBM WebSphere Application Server for z/OS and OS/390 Version 4.0.1:

- ▶ JMS Listener, message driven beans
- ▶ Business Rules support
- ▶ Workarea facility
- ▶ Pluggable custom authentication
- ▶ Security APIs JCE JSSE come in the Java2 SDK on z/OS 1.3.1
- ▶ Security Feature (LDAP authentication)
- ▶ WebSphere for z/OS has a 10 MB limit for GIOP messages
- ▶ ActiveX interoperability

Most of the restrictions for WebSphere Application Server Version 4.01 are documented in APAR II12832.

4.3 Common problems in the migration phase

The behavior of the z/OS runtime is not exactly the same as on distributed platforms. Therefore, you might encounter problems in the porting process that are related to platform-specific implementations. For example, you will most likely encounter JNDI definition and lookup or class loader problems. Here are the most common ones:

4.3.1 JNDI problems

When you deploy a J2EE to a WebSphere z/OS container for the first time, especially if you migrate an application from distributed WebSphere, you may encounter some problems related to registering or accessing J2EE objects in the JNDI namespace. This section addresses some of the most common problems and what you can do to resolve them.

TIP: Often, JNDI problems can be tracked down in the LDAP address space.

The MVS command `F BBOLDAP,APPL=DEBUG=65535` enables detailed tracing. To switch it off again, issue `F BBOLDAP,APPL=DEBUG=0`.

Naming registration failure and `NoClassDefFoundError`

A Naming registration failure manifests itself as a message displayed on the OS/390 or z/OS console. The first time a server region is started after you deploy an application through the SMUI, the names of all EJBs and servlets are registered in the JNDI namespace. If an exception occurs during the Naming registration phase, this message is posted to the console.

To debug this problem, do the following:

1. Under SDSF, locate the active server region job.
2. Expand the job into its constituent parts with the “?” line command.
3. Select the SYSPRINT file and locate the bottom of the trace.
4. Search backward using the word “exception” as a search argument.

Once you locate an exception trace entry, search for a Java stack trace that might identify the source of the naming registration problem. Typically, this problem occurs when the Naming server cannot locate a class referenced by an object such as an EJB in the runtime environment. During JNDI registration, the container requires that all classes referenced by objects be available. The topmost entry in the Java exception stack trace identifies the class that the container cannot locate. Usually, a `NoClassDefFoundError` exception is posted in the stack trace.

When you have identified the missing class, locate the jar file that contains that class in the HFS directory and add the path to that jar file to the `CLASSPATH` record in the `current.env` file.

`NameNotFoundException`

The exception *“Name <name> not found in context “java:comp/env”:* *NameNotFoundException* frequently occurs when a server application fails to locate a reference (<name>) in the JNDI namespace.

The code that populates the `Java:namespace` relies on exception-triggered recursion during the loading process. The first time EJBs are referenced, much of the `java:comp` namespace structure does not exist for the EJB being activated. The container detects this fact through `NameNotFoundExceptions`. For these cases, the container's code that populates the namespace correctly builds the missing portions before trying to bind in the `java:information`.

Name jdbc not found in context “java:comp/env”: NameNotFoundException

The NameNotFoundException exception can also occur for failed datasource lookups during runtime. In this case, the datasource name specified in the code on the lookup method call, in a properties file, or environment variable does not contain the java:comp/env prefix. To correct this problem, ensure that the datasource name includes this prefix. For example, if your datasource name is jdbc/DataSource, it must be defined as java:comp/env/jdbc/DataSource.

javax.naming.NameNotFoundException: LDAP: Error code 32 - No Such Object Remaining name 'ibm-wsnName=<name>,ibm-wsnName=jdbc'

This error is another exception caused frequently by the javax.naming class. This error occurs when you do not pass the “java:comp/env” prefix in the lookup() method when you look up a datasource name. To remedy this problem, add the “java:comp/env” prefix to the parameter string passed to the lookup() method. To ensure platform neutrality, specify the datasource lookup string in a properties or environment file. This way, you can have one version of the datasource name on WebSphere Application Server Advanced Edition and another on WebSphere Application Server for z/OS. The z/OS version must contain the java:comp/env prefix.

4.3.2 Classloader problems

Note: For detailed information about WebSphere z/OS classloader modes, see the documentation for APAR PQ53684 and *WebSphere Application Server V4.0.1 for z/OS: Assembling J2EE Applications*, SA22-7836.

Class violates loader

This condition frequently occurs after you correct a NameNotFoundException or NoClassDefFoundError. In this case, the container has successfully located the reference in the JNDI namespace but is unable to load the class associated with the reference. The exception

```
constraintsjava.lang.LinkageError: Class <classpath/classname> violates  
classloader constraints
```

is caused and displayed at the top of the Java exception stack in the server region's SYSPRINT log. To solve this problem, try changing the classloader mode in the server region's jvm.properties file. WebSphere z/OS uses classloader mode 1 (compatibility mode). To change the mode, do this:

1. Edit the server region's `jvm.properties` file, which can be found in the HFS directory
WebSphere390/CB390/controlinfo/envfile/<sysplex-name>/<server-name>/.
2. Add the following line to the file:
`com.ibm.ws390.server.classloadermode=n`
where `n` is value 0, 1, 2, or 3.

ClassNotFoundException

This occurs when an application tries to load in a class through its string name using:

- ▶ The `forName` method in class `Class`
- ▶ The `findSystemClass` method in class `ClassLoader`
- ▶ The `loadClass` method in class `ClassLoader`

but no definition for the class with the specified name could be found.

NoClassDefFoundError

This occurs if the Java Virtual Machine or a classloader tries to load in the definition of a class (as part of a normal method call or as part of creating a new instance using the new expression and no definition of the class could be found). The searched-for class definition existed when the currently executing class was compiled, but the definition can no longer be found.

NoSuchFieldError

This occurs if an application tries to access or modify a specified field of an object, and that object no longer has that field. Normally, this error is caught by the compiler; it can only occur at runtime if the definition of a class has incompatibly changed.

NoSuchMethodError

This occurs if an application tries to call a specified method of a class (either static or instance), and that class no longer has a definition of that method. Normally, this error is caught by the compiler; this error can only occur at run time if the definition of a class has incompatibly changed.

4.4 Code optimization

Code optimization, especially for programs or applications written in Java, can fill and has already filled whole books. The fact that Java can be used for very different kinds of applications is only one reason. A more important reason for code optimization is the independence of Java programs from the underlying system. This means that an application that works without problems or bottlenecks on one platform, can behave quite differently after migration to another platform.

In this redbook we focus on the main steps to migrate a running J2EE application from a distributed platform to z/OS or OS/390, with the goal of getting the application up and running.

If you are looking for detailed information on application tuning, we highly recommend the WebSphere Best Practice Library on the WebSphere Developer Domain which can be found on the following link:

<http://www7b.boulder.ibm.com/wsdd/library>

At the time of writing, another project dealing with z/OS- or OS/390-specific Java optimization was already launched. Stay tuned for the Java for z/OS-specific redbook coming out (probably) 1Q 2002.

Nevertheless, we recommend using code analyzer tools and application server monitors to step into the application and runtime environment. You will find the bottlenecks only with the appropriate tools. Further information about analyzer tools can be found in Part 3, "Tools for WebSphere Application Server on z/OS" on page 133.

We recommend that you heed the following advice while developing or designing a J2EE application in order to ease its later migration.

4.4.1 Absolute vs. relative path

Try to avoid absolute path statements in any code, especially in HTML and JSP files and servlets.

If only relative path statements are chosen, the deployment, for example, to a test or production server on the same system can easily be accomplished by changing the context root statement for each application.

4.4.2 Avoid specialities

Don't exploit specialities in the distributed runtime or in the tooling. For example, do not use some WebSphere Application Server Advanced Edition-specific classes in the hierarchy `com.ibm.websphere`; maybe they exist on z/OS, maybe not, maybe they do not act the same! The safest way is to deviate as little as possible from the J2EE specification.

4.4.3 Use WebSphere connection pooling

Don't roll your own database connection pooling. Use the connection pooling provided by the JDBC 2.0 compliant driver. In one word: be J2EE (compliant).

4.4.4 Read properties files from a .jar file

To avoid codepage problems while porting an application to z/OS, we recommend that you package the properties files needed by your application into a .jar file. These properties can then be read using `ResourceBundle.getBundle()`. Because of the use of Java, this principle works on any platform, and no additional porting steps are needed.

4.4.5 Use `javax.sql.DataSource` for JDBC connection

A `javax.sql.DataSource` is obtained from WebSphere Application Server through a JNDI naming lookup.

While porting the application to another server or system, only a new data source with the old JNDI lookup name has to be defined within the application server. This new `javax.sql.DataSource` establishes the database connection and can be reached by the application via the same JNDI name. So no code changes are needed.

An additional advantage of using `DataSource` defined by WebSphere Application Server is the connection pooling. The connection pools can be easily defined for each `DataSource`, and with regard to performance, they are the recommended way to connect to a database.

4.4.6 Put supporting .jar files into the .ear files root

The Windows environment allows you to put supporting .jar files in any subdirectory of the .ear file. WSAD and AAT for Windows have a "classpath" field for Web Modules and EJB Modules, where you can specify in which subdirectories the .jar files reside. In AAT/390 this classpath field does not exist, and the entry in the .ear file (produced with Windows) is ignored. Therefore, you

must put all your supporting .jar files in the root directory of the .ear file. In a Windows environment you must then put an entry like “support.jar” in the classpath field. IBM WebSphere Application Server Advanced Edition allows you to define in the property *ws.ext.dirs* where you can place common .jar files. This support is not implemented in WebSphere Application Server on z/OS today.

4.4.7 Transaction attributes for Session EJBs

The Windows environment allows you to specify TX attributes for only some Session EJB methods. All “unspecified” methods are defaulted to be “Supports”, which is quite a good default value in most cases. The z/OS environment does not allow that some methods are mapped and some not. Therefore, you must specify TX attributes for *all* methods to avoid any confusion.

4.5 Code migration hints and tips

Some of the issues you find should be reflected in the code checklist in 3.1, “Application analysis” on page 32 .

4.5.1 ASCII/EBCDIC issues

Chapter 24 of *Java Programming Guide for OS/390*, SG24-5619 describes the most common issues concerning ASCII/EBCDIC and how to resolve them.

Codepage conversion

An XML or property file that is contained in a .jar file needs to be in ASCII, while a property file that is stored in the hierarchical file system needs to be in EBCDIC. To easily convert from ASCII to EBCDIC, just create a little shell command file like the following:

```
echo “Enter as parameters the list of files to be converted to EBCDIC.”
for i in $*
do
mv $i $i.ascii
iconv -f IS08859-1 -t IBM-1047 $i.ascii >$i
echo $i” OK”
done
```

If you need to upload a file to z/OS in EBCDIC through ftp, you need to ensure with the local network specialist that the FTPDATA file has been correctly customized for IBM-1047 code page. Otherwise, you can use the following ftp command from your PC to force load the appropriate ftp translation table:

```
ascii
quote site SBDATACONN=(IS08859-1,IBM-1047)
```

To ensure proper translation, use *readers* and *writers* versus *streams* for `HTTPServletResponse` and file conversions.

Take care of conversion of strings to byte arrays using the `getBytes` method. Many Java products use byte arrays (`byte[]`) to manipulate data. Byte arrays are safe to use if you ensure that they are created correctly. If you are creating them from a string, you must know how the string was created. In many cases, strings are created using the default encoding of the system (in our case, this was cp1047, or EBCDIC Latin 1), but the programmer assumed the string was in ASCII (ISO8859-1), by using a deprecated `getBytes` method. Use of `getBytes()` or `getBytes(String enc)` is safe. We prefer the `getBytes()` method because it uses the default encoding for the system the program is running on.

Check connector customization issues about code page handling. Ask yourself: Is there a connector customization or is it some parameter coded within a Java function?

ASCII editing

If you want to edit ASCII files from a UNIX terminal emulation session you will find the following shell script useful:

```
#!/bin/sh
#
# The idea is to edit an ascii file. The approach is to
# copy the original file for safekeeping.
# run iconv to create a file with the contents in codepage 1047
# create an empty file with the same timestamp as the one just created.
# invoke vi against the work file (in codepage 1047)
# if the edited file was changed, then iconv the work file
# into the original file.
# In all cases, get rid of the temporary files.
#

TMP_NAME="tmp_"$LOGNAME_"$(date '+%d_%m_%y_%T')_work"
TMP_NAME2="tmp_"$LOGNAME_"$(date '+%d_%m_%y_%T')_time"
TMP_NAME2="tmp_"$LOGNAME_"$(date '+%d_%m_%y_%T')_orig"

cp -m $1 $TMP_NAME2

iconv -f ISO8859-1 -t IBM-1047 $1 > $TMP_NAME
if [ $? -ne 0 ]; then
    if [ -a $1 ]; then
        rm $TMP_NAME $TMP_NAME2
        echo "Error converting input file"
        exit 1
    fi
fi
```

```

fi

touch -r $TMP_NAME $TMP_NAME2

vi $TMP_NAME
if [ $? -ne 0 ]; then
    rm $TMP_NAME $TMP_NAME2 $TMP_NAME2
    echo "Error encountered in vi"
    exit 2
fi

if test ! $TMP_NAME -nt $TMP_NAME2 ; then
    rm $TMP_NAME $TMP_NAME2 $TMP_NAME2
    exit 0
fi

iconv -f IBM-1047 -t ISO8859-1 $TMP_NAME > $1
if [ $? -ne 0 ]; then
    cp -m $TMP_NAME2 $1
    rm $TMP_NAME $TMP_NAME2 $TMP_NAME2
    echo "Error encountered converting the updated file"
    exit 3
fi
rm $TMP_NAME $TMP_NAME2 $TMP_NAME2

```

Note: This shell script will not work in a 3270 emulated UNIX session (like using the OMVS command within a TSO session).

4.5.2 DB2 UDB issues

DB2 for OS/390 limitations are usually tighter than those for distributed platforms.

To check them, have a look in:

- ▶ Appendix A of *DB2 UDB for OS/390 and z/OS V7 SQL Reference*, SC26-9944
- ▶ Appendix A of *IBM DB2 Universal Database SQL Reference Version 7*, SC09-2974
- ▶ *WebSphere Application Server V4.0.1 for z/OS and OS/390: Migration*, GA22-7860

Table 4-1 lists the limitations you are most likely to face.

Table 4-1 DB2 for OS/390 limitations

Item	Distributed platforms	z/OS
Constraint name	18 bytes	8 bytes
Correlation name	128 bytes	18 bytes
Hhost identifier	255 bytes	64 bytes
Schema name	30 bytes	8 bytes
Unqualified column name	30 bytes	18 bytes
Unqualified table space name	18 bytes	8 bytes
Maximum length of VARCHAR	32672 bytes	depends on page size
Maximum length of LONG VARGRAPHIC	16350 bytes	depends on page size

4.5.3 Redeployment using AAT for z/OS

The .ear file contains pure J2EE descriptors, but it can also contain application server-related descriptors that are not supported by IBM WebSphere Application Server for z/OS and OS/390. AAT generates system-specific code to connect EJBs with the container. Therefore, it is a requirement that redeployment (see CMP persistence issues on page 63) has to be done using the AAT.

Generating system specific code is compliant to the SUNs EJB specification.

4.5.4 Handling minor changes during the testing phase

When a J2EE application is deployed on IBM WebSphere Application Server for z/OS and OS/390 Version 4, the system management task extracts from the deployed .ear file all of the files it contains to the WebSphere Application Server apps subdirectory. Under this subdirectory, you can retrieve all servers and resources that have been deployed.

When you are testing your application and have only minor application structure changes, you can update the files directly in the server subdirectory without running the entire deployment process again.

To do so you can use ftp or, as we did, SMB (the setup of the server installation for SMB is described in *S/390 File and Print Serving*, SG24-5330).

This quick deployment method is dirty, so please don't call the IBM support center if you mess things up.

Figure 4-5 shows the WebSphere Sample Benchmark Application files after it was installed on our BBTEST server.

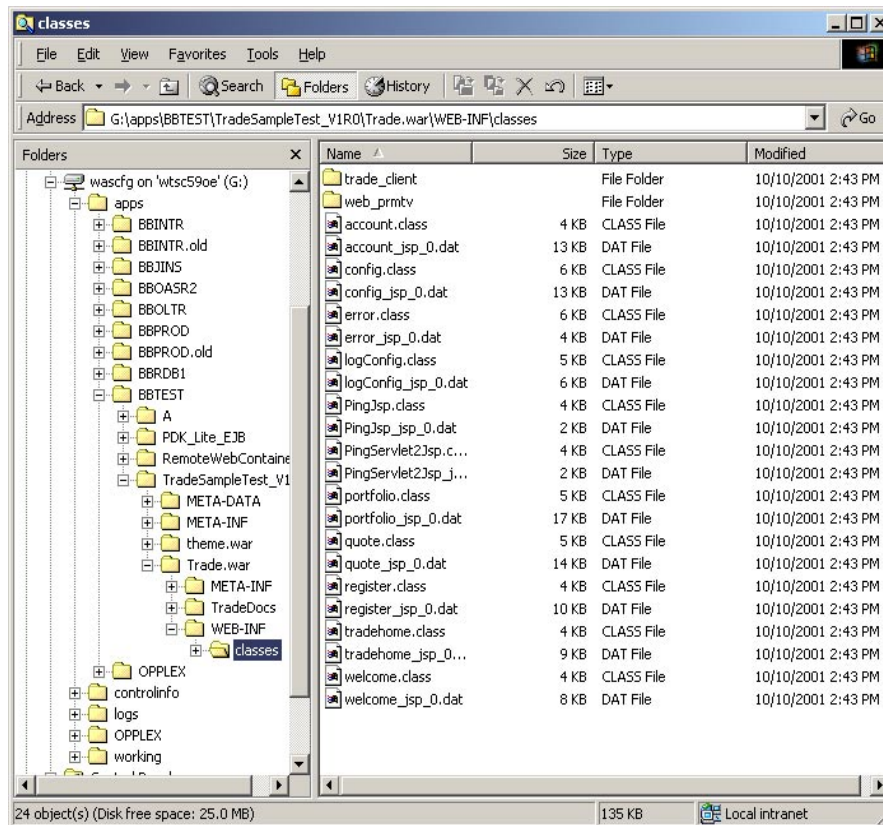


Figure 4-5 .ear file expanded in HFS

4.6 Final steps

Once the code has been tested, you need to freeze it and consider some non-functional modifications:

- ▶ If you modified some application files directly under USS, move the modification back to the IDE source files.
- ▶ Comment out all the code added to debug the installation.
- ▶ Compile the code without debug attributes.

- ▶ Apply versioning in application design tools.
- ▶ Consider disabling some J2EE server traces (especially, select the J2EE Production Server box in the J2EE server property panel) and testing the application with the final WebSphere implementation (for example, sysplex wide).
- ▶ The very last step before going to production is to benchmark the application and apply optimizations to the code and the system that will receive the application.



Part 2

Managing WebSphere applications

This part describes the issues and actions that are involved during the life cycle of a WebSphere application, from cradle to grave.



The lifecycle of an application

This chapter describes the life cycle of an application, from test to production, integrated into System Management processes.

5.1 Introduction

IBM WebSphere Application Server for z/OS and OS/390 provides a highly available, secure, reliable, and scalable runtime environment for Java 2 Enterprise Edition (J2EE) applications. It is probably one of the few major products designed to use and take advantage of Parallel Sysplex and Data Sharing. These are capabilities enabled by IBM e(logo) Server z/Series hardware and operating systems that provide enhanced capacity, intelligent workload balancing and availability.

Parallel Sysplex brings unique opportunities and challenges in the area of software testing for maintaining and upgrading both software products and user-written applications.

The core function that makes it possible to dispatch diverse workloads intelligently among various systems in a sysplex is Workload Manager (WLM). In the WLM configuration, *application environments* are defined and applications are assigned to performance goals. Performance goals are defined according to your Service Level Agreements (SLA). Then WLM is in charge to manage system resources to achieve the assigned performance goals.

Moreover, if an application is installed across several machines of the sysplex, WLM will determine which instance on which sysplex server should better process an incoming request. This tight integration between z/OS, WLM and the applications makes it possible to implement a very efficient and high available load balancing solution within the system boundaries.

WebSphere is aware of WLM, so that when a request to create an EJB comes in, the WebSphere system regions refer to WLM to assign the request to a J2EE server region instance. For a description of this mechanism, see 5.7, “Control flow between the client and WebSphere” of *Enterprise JavaBeans for z/OS and OS/390*, SG24-6283.

Default performance goals have to be assigned to the application servers. These defaults can be overridden when incoming requests are WLM classified by the scalable Web server or by the HTTP protocol handler. See Figure 5-1 on page 83 for more details on WLM classification for different work requests.

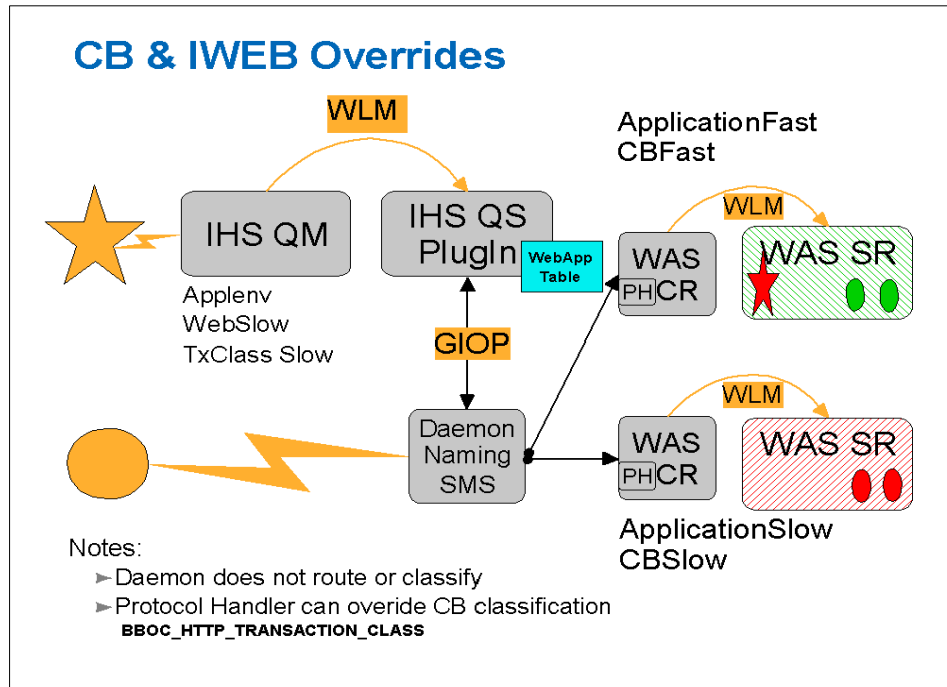


Figure 5-1 WLM classification flow and overrides

Starting by running a monoplex sysplex (one machine with sysplex mechanisms enabled), we used WLM to assign different performance parameters for test servers and production servers on the same machine; see Figure 5-2 on page 84. In such a way, the production environment has a high priority compared to test servers.

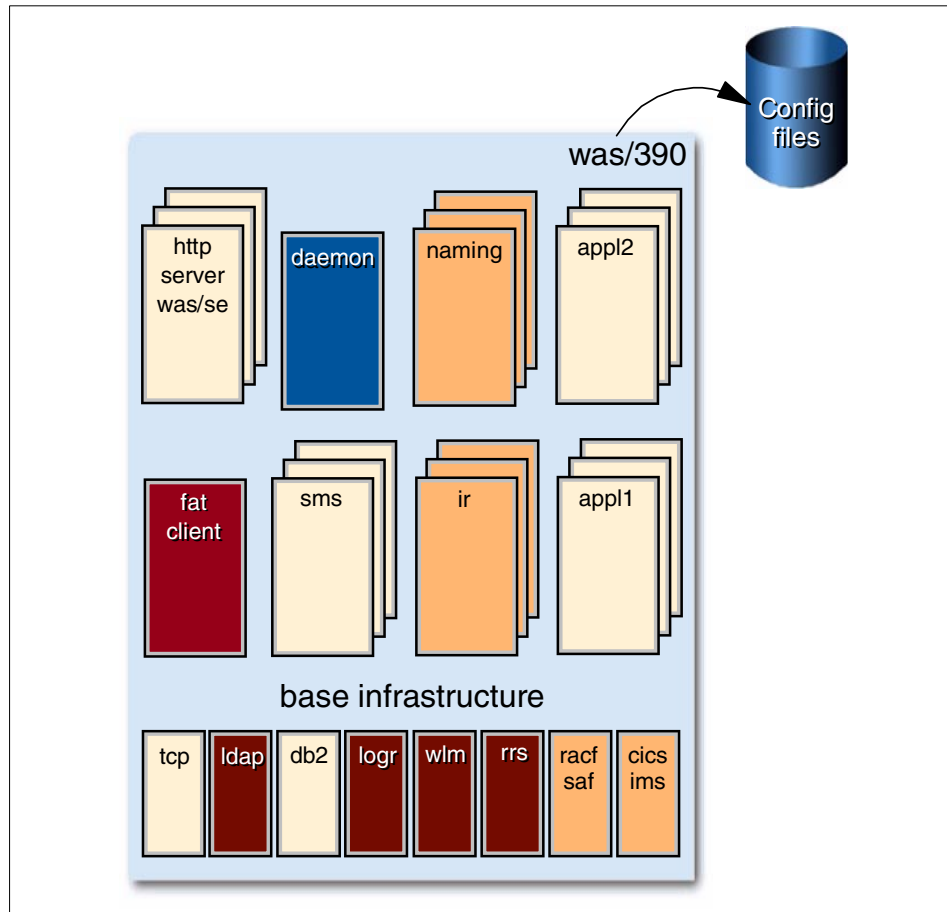


Figure 5-2 WebSphere monoplex infrastructure

We then extended our tests to a three-system sysplex in order to improve the reliability of our system. See Figure 5-3 on page 85 for a simplified view of a WebSphere sysplex infrastructure. Production server instances were defined on all images of the sysplex. An HTTP request could come to an HTTP server on the system and then the WLM mechanism could choose to process it in another system J2EE server instance.

As a result, we were able to stop a production server or even a whole z/OS image without interruption of the application service.

We did not implement a TCP/IP load balancing mechanism (WebSphere Edge Server, DNS/WLM, Cisco MNLN, Sysplex Distributor) to route the HTTP requests to the active images. Such products communicate with WLM to route IP packets to the TCP/IP stack on the machine where the application should perform the best. We strongly recommend to investigate the implementation of load distribution mechanisms in a multi-image production environment.

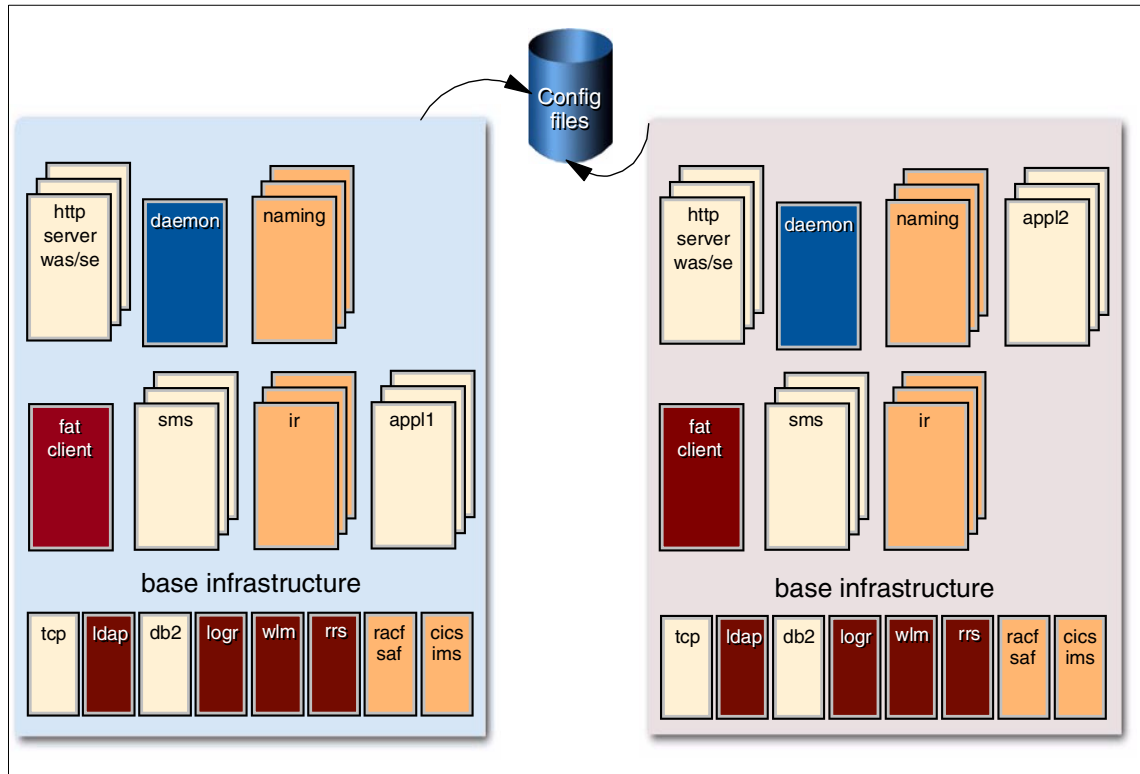


Figure 5-3 WebSphere sysplex infrastructure

5.2 Setting up J2EE servers on z/OS

Once you have developed and assembled an executable J2EE application, you need to install it in an appropriate runtime environment. The IBM WebSphere Application Server for z/OS and OS/390 provides that runtime environment.

5.2.1 Common steps for J2EE creation

There are two ways to create a new J2EE server under IBM WebSphere Application Server for z/OS and OS/390. One is by using the System Management Enhanced User Interface Administration Application (SMUI), and the other is by using the System Management Scripting API(SMAPI) scripts.

The values in table Table 5-1 are used for the J2EE server in our example.

Table 5-1 Input definitions for J2EE server creation

input information	Value
Server name	BBTEST
Server instance name	BBTESTA1
Control Region RACF userid	BBTESTC
Control Region Started Task name	BBTESTC
Control Region RACF groupid	BBTESTG
Server Region RACF userid	BBTESTS
Server Region PROC name	BBTESTS
Server Region RACF groupid	BBTESTR
Default remote RACF userid	BBTESTI
Default local RACF userid	BBTESTD
Default RACF groupid for default userids	BBTESTP
WLM Application Environment	BBTEST
Host name	wtsc59oe.itso.ibm.com
sysname	sc59
Host port	8110
JAVA_HOME	/usr/lpp/java/IBM/J1.3
WSAS HOME	/usr/lpp/WebSphere

SMUI-driven server installation

We performed the following steps to create a new J2EE server and install one J2EE application:

1. Make sure the WebSphere Application Server infrastructure is up and running.

2. Logon to the SMUI and add a new conversation.

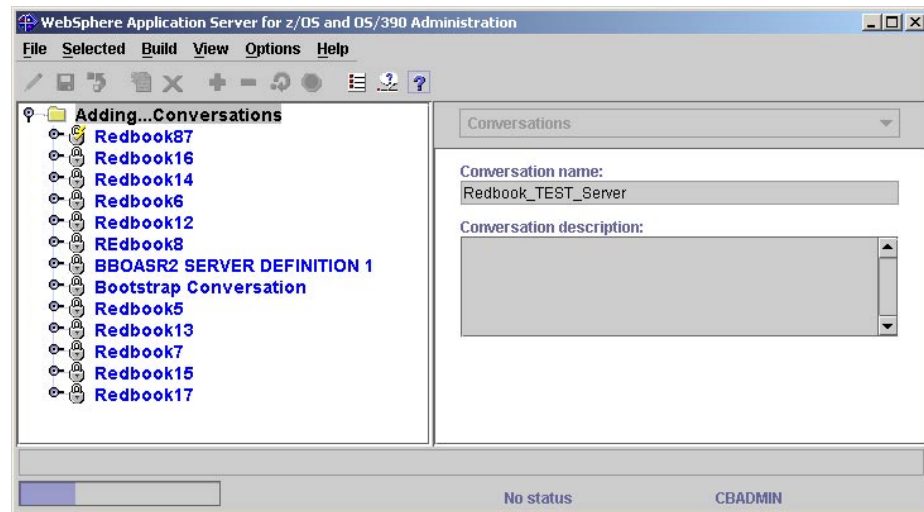


Figure 5-4 Adding a new conversation

3. Add a new J2EE server

At this point we did not add any specific environment variables. We will do it after the J2EE application installation. See also considerations in 5.2.2, “Setting up a J2EE test server” on page 97 for test servers and 5.2.3, “Setting up a production J2EE server” on page 98 for production servers.

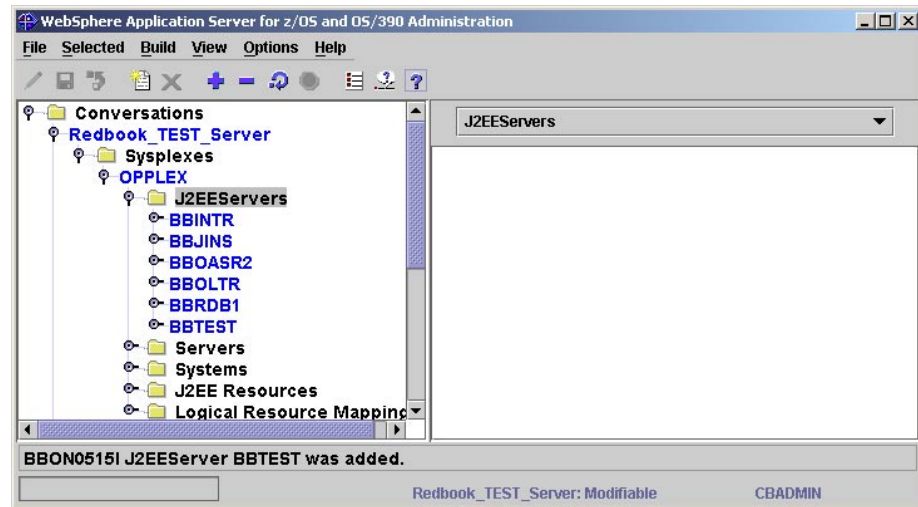


Figure 5-5 Adding a new J2EE server

4. Add a server instance for the J2EE server

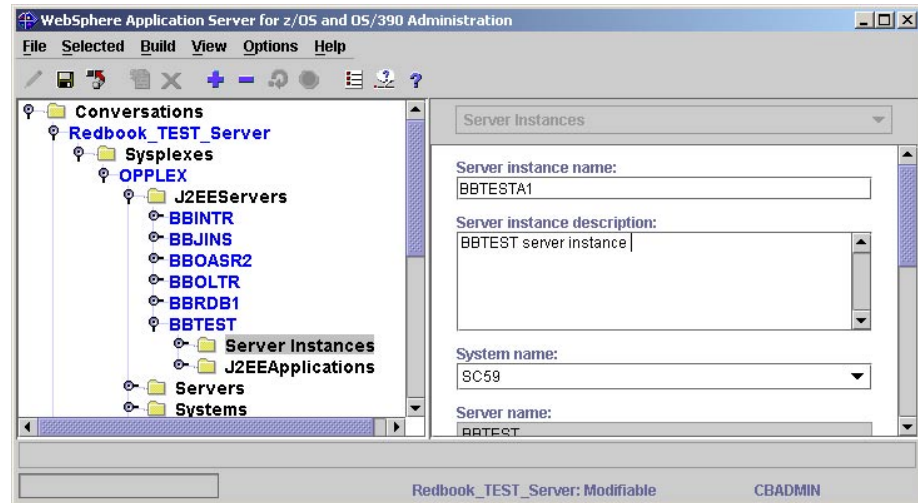


Figure 5-6 Adding a server instance

5. Add a J2EE resource

Our sample application uses JDBC to access DB2 databases. We added the required resource, which is used further on in our steps.

- a. Add a J2EE resource (DB2datasource) and a J2EE resource instance.

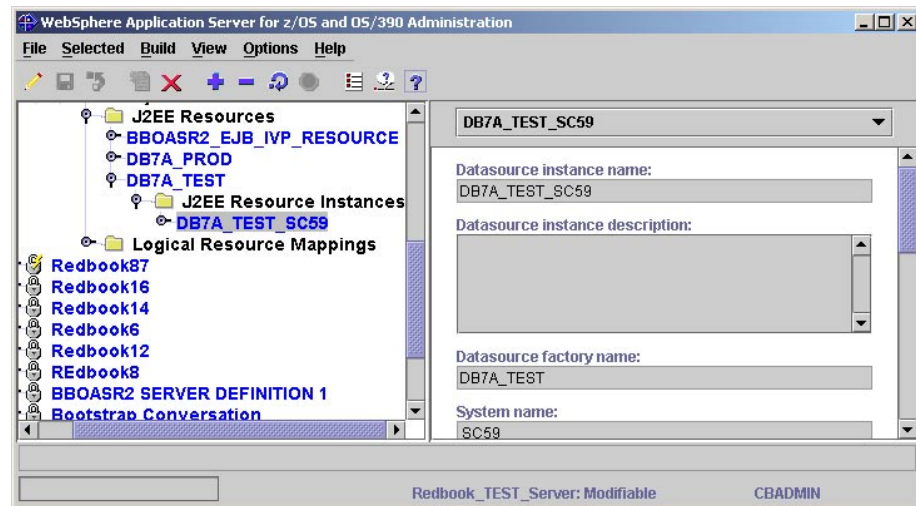


Figure 5-7 Adding a J2EE resource

SMUI-driven application installation

6. Install a J2EE application (note that this step is optional if you are not installing any J2EE application at this time).

During the installation process, we were prompted to perform a series of manual activities:

- a. When we selected “Install J2EE Application”, we were prompted to supply the .ear file location, name and destination (ftp server). The .ear file must have been processed with the Application Assembly Tool for z/OS.

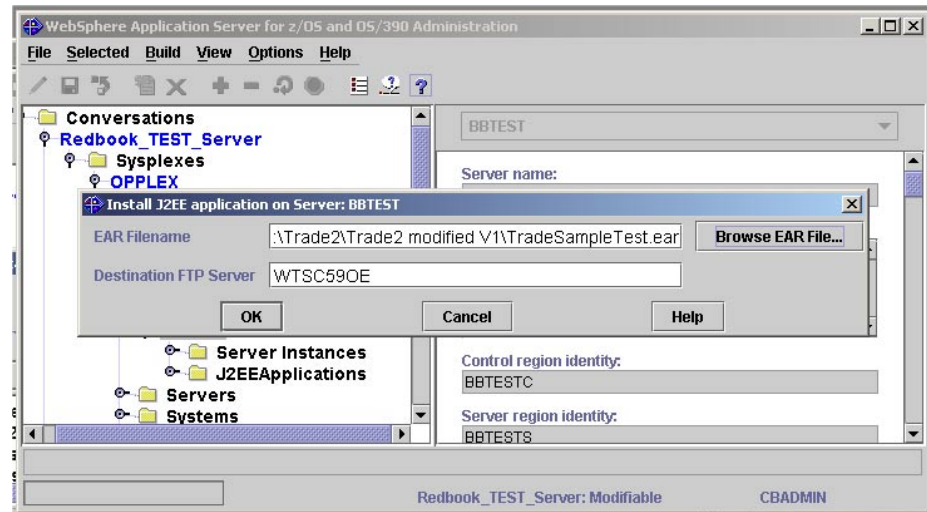


Figure 5-8 Loading a .ear file

- b. After the .ear file is loaded, the reference and resource resolution window is shown. We assigned the default JNDI path and name for all and set the J2EE resource for the ones that required it.

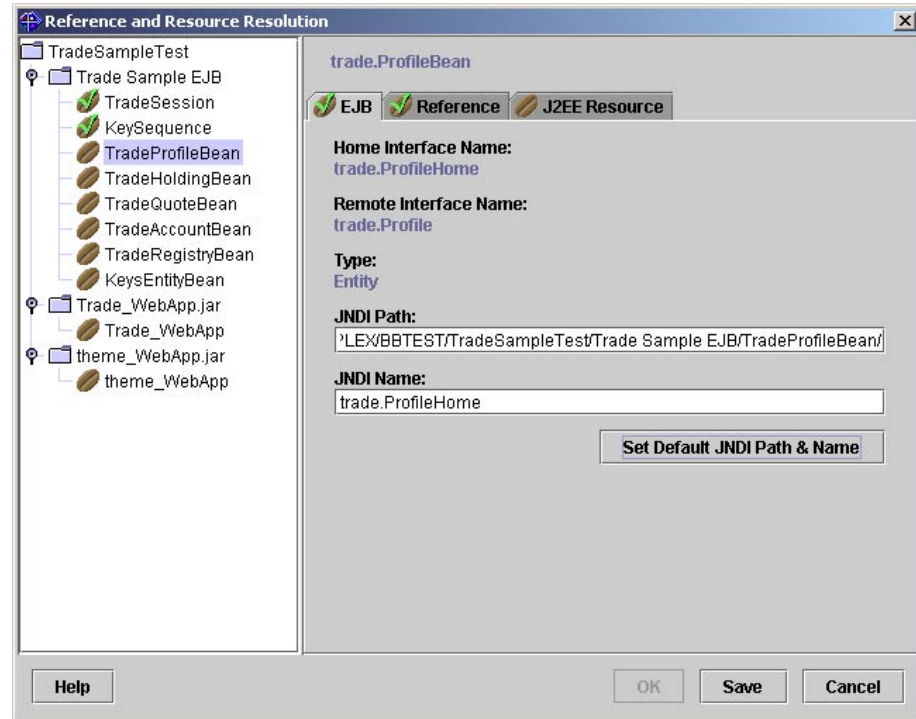


Figure 5-9 Setting the JNDI path and name, and the J2EE resource

For this example, the J2EE resource was defined in step 5., “Add a J2EE resource” on page 88.

When done, we selected **OK** to continue, and the .ear file was transferred to WebSphere Application Server runtime space, in a process that is called by the GUI “Deploying BBTEST”.

The process of installing a J2EE application ends here.

The System Management Enhanced User Interface Administration Application saves a new .ear file in the same place where it read it from (see Figure 5-8 on page 90). In our example, the new name would be:

SampleTradeTest_resolved.ear.

We discuss the importance of saving this new .ear file in 5.3.3, “Scripted application fallback” on page 107.

7. Validate the conversation.

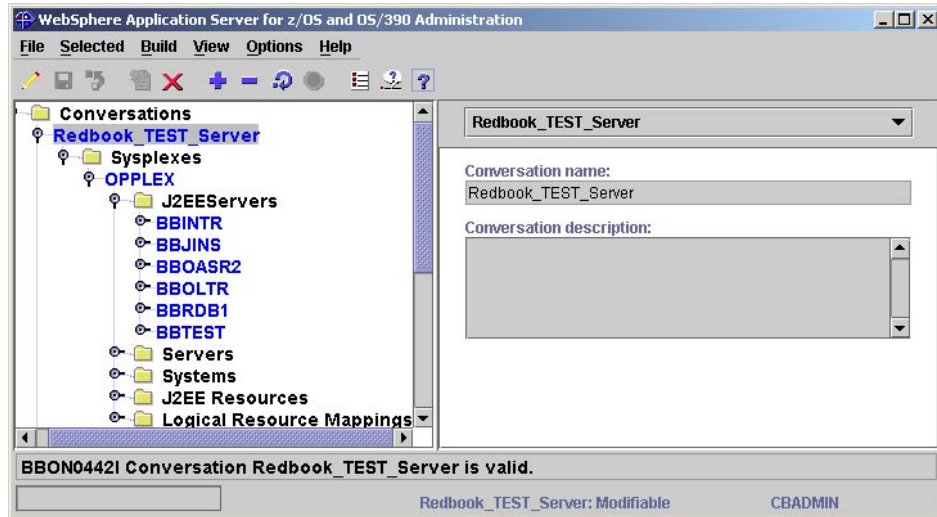


Figure 5-10 Validating the conversation

8. Commit.

The commit process initializes the environment variables based on the ones defined for the sysplex. If these variables cover most of what is needed, you only need to add application- or server-specific variables. Coding specific variables at the application level provides higher granularity and flexibility.

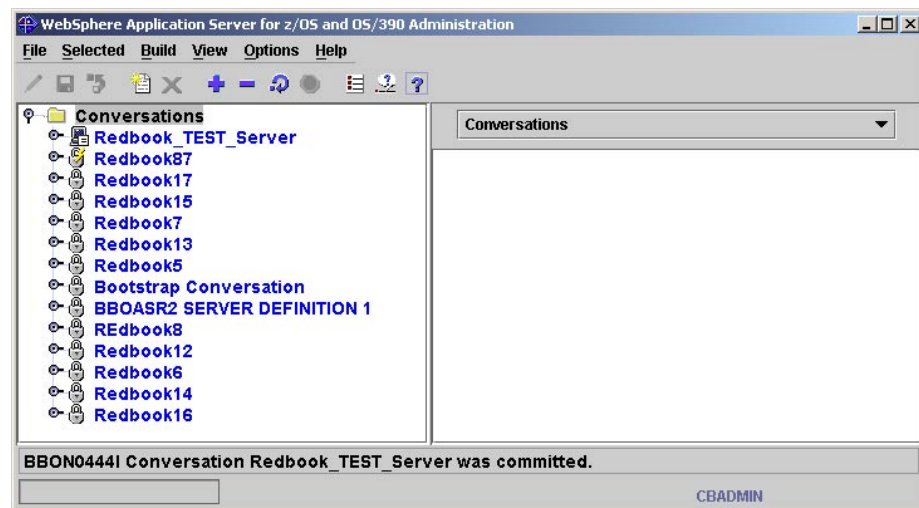


Figure 5-11 Committing the conversation

9. Complete the tasks provided in the instructions:
- Security tasks
 - Workload management (WLM) tasks
 - Automatic restart manager (ARM) tasks
 - Automation tasks
 - Resource management tasks
 - Logstream tasks

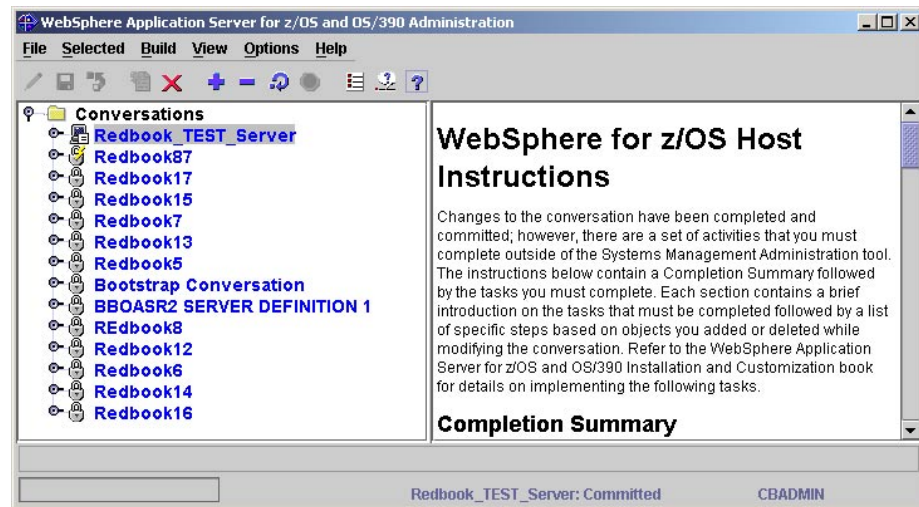


Figure 5-12 WebSphere tasks to perform

10. Activate the new conversation.

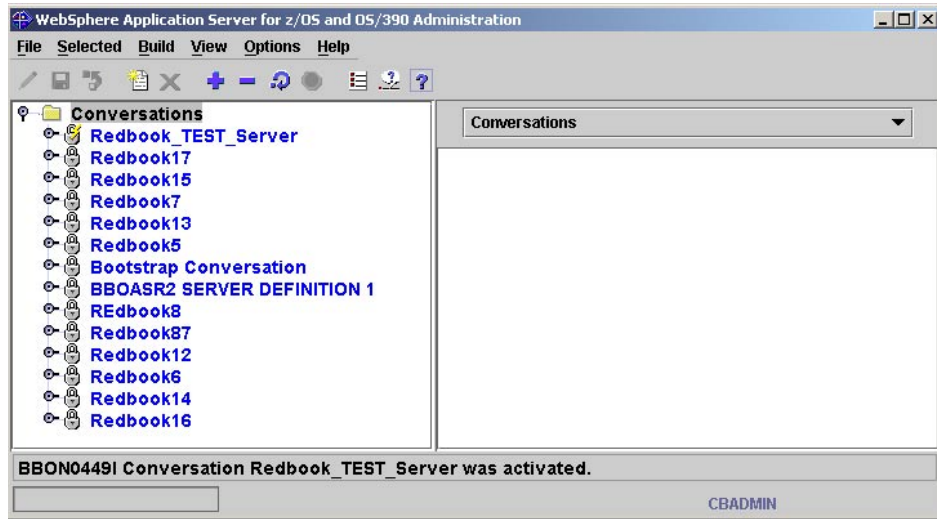


Figure 5-13 Activating a conversation

Manual steps for server and application installation

11. Hostname and tracing information is stored in a flat file that is not managed by the SMUI. There are a few manual steps to do. In our environment, WSAS uses information and the application under path /WebSphere390/CB390/. Our server instance is called BBTESTA1.

In location /WebSphere390/CB390/controlinfo/envfile/OPPLEX/BBTESTA1/ add and edit the following files:

- a. Create a file called webcontainer.conf. There is a sample located at /usr/lpp/WebSphere/bin/.

We changed this statement in the file:

```
host.default_host.alias=wtsc59oe.itso.ibm.com:8110, sc59:8110
```

- b. Enable JVM tracing.

Create a file called trace.dat and add the following property:

```
*=all=enabled (refer to test and production server considerations)
```

- c. Create a file called jvm.properties. This file points the BBTEST server to the two files you just created. Add the following two properties to the file:

```
com.ibm.ws390.wc.config.filename=/WebSphere390/CB390/controlinfo/envfile
/OPPLEX/BBTESTA1/webcontainer.conf (note that this is one line)
com.ibm.ws390.trace.settings=/WebSphere390/CB390/controlinfo/envfile/OPP
LEX/BBTESTA1/trace.dat (note that this is one line)
```


- d. Ensure that all three files have the same ownership and permissions as the current.env file in the control file directory.

12. Configure the HTTP Server plugin.

We configured the BBTEST server plugin under one of our IBM HTTP Server instances.

- a. Change the environment variable file, typically called httpd.envvars.

We added the following paths to the following variables:

Add to LIBPATH:

/usr/lpp/WebSphere/wc/lib

Add to CLASSPATH:

/usr/lpp/WebSphere/wc/lib

Add or change JAVA_HOME:

JAVA_HOME=/usr/lpp/java/IBM/J1.3

- b. Add the plugin directives in the config file, typically called httpd.conf.

We added the following directives for the HTTP server:

ServerInit

/usr/lpp/WebSphere/WebServerPlugIn/bin/was400plugin.so:init_exit

/usr/lpp/WebSphere,/web/bean/was.conf (note that this is one line)

ServerTerm

/usr/lpp/WebSphere/WebServerPlugIn/bin/was400plugin.so:term_exit (note that this is one line)

Service /WebSphereSamples/TradeSample/Test/*

/usr/lpp/WebSphere/WebServerPlugIn/bin/was400plugin.so:service_exit
(note that this is one line)

- c. Optionally, you can add your own plugin config file.

We defined a new plugin config file called was.conf.

13. Start the newly created server BBTEST.

As part of the conversation, following the instructions in step 9., “Complete the tasks provided in the instructions:” on page 93, we created the PROC.

To start the BBTEST server, we issued the following z/OS command:

```
s bbtest.bbtesta1
```

There are some success indicators to monitor:

- a. On z/OS console, look for message:

+Server "BBTESTA1" open for business.

- b. As this is a new server, you should also expect to see messages related to your J2EE application (naming registration). For the Trade2 J2EE application, we checked these messages:

```
+BBOU0697I REGISTERING COMPONENT TradeAccountBean FOR SERVER BBTEST
+BBOU0697I REGISTERING COMPONENT TradeQuoteBean FOR SERVER BBTEST
+BBOU0697I REGISTERING COMPONENT TradeRegistryBean FOR SERVER BBTEST
+BBOU0697I REGISTERING COMPONENT TradeHoldingBean FOR SERVER BBTEST
+BBOU0697I REGISTERING COMPONENT TradeProfileBean FOR SERVER BBTEST
+BBOU0697I REGISTERING COMPONENT KeySequence FOR SERVER BBTEST
+BBOU0697I REGISTERING COMPONENT TradeSession FOR SERVER BBTEST
+BBOU0697I REGISTERING COMPONENT Trade_WebApp FOR SERVER BBTEST
+BBOU0697I REGISTERING COMPONENT theme_WebApp FOR SERVER BBTEST
+BBOU0697I REGISTERING COMPONENT KeysEntityBean FOR SERVER BBTEST
+BBOU0698I REGISTERING SERVER BBTEST
+BBOU0695I NAMING REGISTRATION COMPLETED FOR SERVER BBTEST
BBOU0695I NAMING REGISTRATION COMPLETED FOR SERVER BBTEST
```

14. Start the HTTP server with the plugin enabled.

We started our HTTP server enabling the very verbose (-vv) trace to enable debugging. See also 5.2.2, “Setting up a J2EE test server” on page 97 for specific customizations for the test servers and 5.2.3, “Setting up a production J2EE server” on page 98 for the production servers. It is not recommended to run the HTTP server in production with the -vv tracing option enabled.

A success indicator can be found in your HTTP server joblog:

```
EJS3035I  IBM WebSphere Application Server for OS390 native plugin
initialization went OK :-)
```

Installation verification

15. Access the application.

Trade2 is a complete J2EE application. We verified the Trade2 application by accessing it from the following URL using a Web browser.

```
http://wtsc59oe.itso.ibm.com:8110/WebSphereSamples/TradeSample/Test/TradeDocs/index.html
```

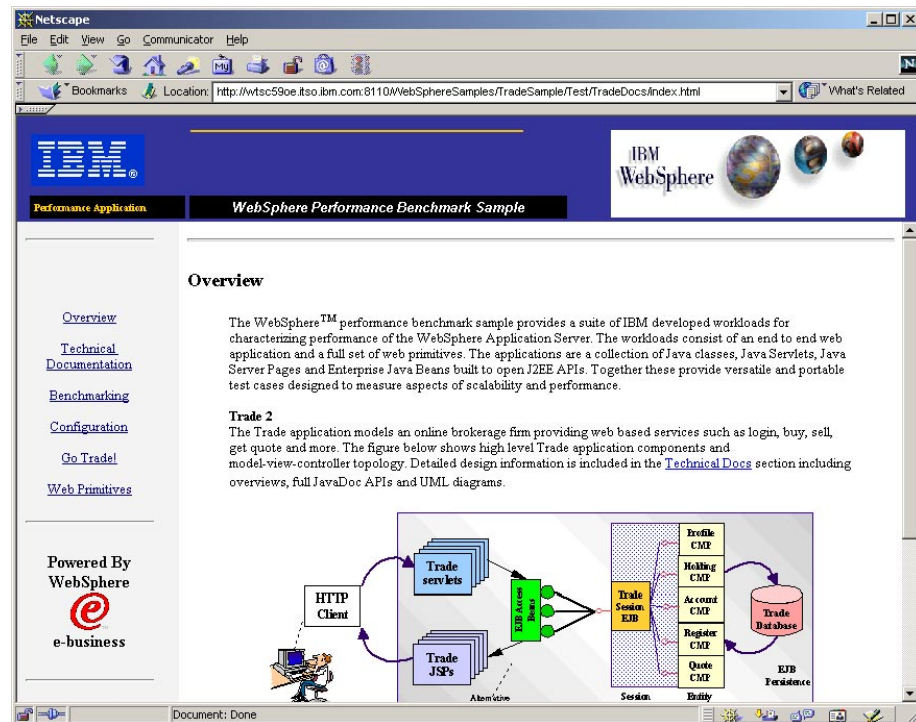


Figure 5-14 BBTEST server and Trade2 J2EE application

5.2.2 Setting up a J2EE test server

When setting up a test J2EE server, we evaluated what typically differentiates test from production in a z/OS environment, or even other platforms. Some of the extra features enabled for a test J2EE server are:

- ▶ Enable debugger and Object Level Trace (see Chapter 8, “Object Level Trace” on page 143).
- ▶ Enable JVM trace (see “Enable JVM tracing.” on page 94).
- ▶ Enable the runtime trace by setting the environment variable TRACEALL=1 for this J2EE server. We also enabled TRACEBUFFLOC=SYSPRINT. For more information about these variables, check *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834.
- ▶ Enable SMF data recording (see *WebSphere Application Server V4.0.1 for z/OS and OS/390: Operations and Administration*, SA22-7835).
- ▶ Use a separate dedicated Web server or separate it from production by running the Web server in scalable mode (see 5.2.4, “Setting up WLM

classification” on page 98). Depending on the application requirements, use the HTTP Transport Handler instead of a Web server plugin (see *WebSphere Application Server V4.0.1 for z/OS and OS/390: Assembling J2EE Applications*, SA22-7836 for possible functional differences).

- ▶ Use WLM to classify workloads to this test server with lower performance goals (5.2.4, “Setting up WLM classification” on page 98).
- ▶ Set the J2EE server replication policy of one per system. The replication policy specifies how many server regions should be started. If you have multiple server regions, which is the normal case, you have to keep your session state externalized.
- ▶ If running a sysplex, keep the test J2EE server running in only one system or LPAR.

5.2.3 Setting up a production J2EE server

When setting up a production J2EE server, we used the same line of thinking as described in the preceding section:

- ▶ Whenever possible, disable most of the tracing, like the ones described in 5.2.2, “Setting up a J2EE test server” on page 97.
- ▶ Use a separate dedicated Web server or separate it from test by using it in scalable mode (see 5.2.4, “Setting up WLM classification” on page 98). Depending on the application requirements, use the HTTP Transport Handler instead of a Web server plugin (see *WebSphere Application Server V4.0.1 for z/OS and OS/390: Assembling J2EE Applications*, SA22-7836).
- ▶ Use WLM to classify traffic to this production J2EE server (5.2.4, “Setting up WLM classification” on page 98).
- ▶ If running the J2EE server in a sysplex, consider running J2EE server instances in more than one system or LPAR. This improves workload management and availability.
- ▶ Use SMF data recording with care (see *WebSphere Application Server V4.0.1 for z/OS and OS/390: Operations and Administration*, SA22-7835). Excessive SMF activity may degrade performance.

5.2.4 Setting up WLM classification

Earlier in this chapter, we recommended using WLM policies and classifications to differentiate a test J2EE server from a production server.

Workload management functions exploited by IBM WebSphere Application Server for z/OS and OS/390 are:

- ▶ Sysplex routing of work requests

For more information about it, refer to *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834, and *TCP/IP in a Sysplex*, SG24-5235. We recommend that you support the WebSphere Application Server workload routing ability in a sysplex environment with any workload distribution solution. This is not discussed in detail in this book.

- ▶ Address space management for work requests

Address space management for work requests

IBM WebSphere Application Server for z/OS and OS/390 propagates the performance context of work requests through the use of WLM enclaves. An enclave is a transaction that can span multiple dispatchable units (SRBs and tasks) in one or more address spaces and is reported on and managed as a unit.

Each transaction has its own enclave and is managed according to its service class. A service class is a group of work that has the same performance goals, resource requirements, or business importance. For workload management, you assign a service goal and optionally a resource group to a service class. When a transaction is classified to a high priority service class, WLM can direct the work to a high priority server instance.

Further discussion of WLM concepts can be found in *OS/390 Workload Manager Implementation and Exploitation*, SG24-5326.

Basically, there are three different ways to classify work with WLM and they depend on how you plan to access your J2EE server:

- ▶ Using WLM subsystem type IWEB with the IBM HTTP server plugin
The IBM HTTP server has to operate in scalable mode to be enabled for WLM workload classification.
- ▶ Within the HTTP transport handler using the
BBOC_HTTP_TRANSACTION_CLASS environment variable to classify workloads
- ▶ Using WLM subsystem type CB
This is the default for all nonexplicit classified workloads and incoming IIOp requests.

Tip: You need to define either the CB default classification, or define and override using IWEB Application Environments, WLM SN enclave support, or the HTTP transport handler WLM classification variable. If you fail to do so your workload will end up with bad performance goals. You are on the safe side if you define usable default goals in the CB rules and assign them to any application server.

You can use RMF reports to analyze your environment. We provide sample RMF JCL and reports in “RMF examples” on page 189.

WLM classification with the IBM HTTP server plugin

One of the options for accessing a J2EE application is to use the HTTP server plugin. The incoming HTTP protocol-based requests can be classified explicitly in the IBM HTTP server. Therefore, the IBM HTTP server has to operate in scalable mode or the standalone mode enclave classification support has to be activated.

For scalable mode classification you need to define WLM Application Environments and route HTTP requests to them using the *AppEnv* directive in *http.conf*. For more details on how to set up an HTTP server in scalable mode, refer to *OS/390 e-business Infrastructure: IBM HTTP Server V5.1 for OS/390*, SG24-5603.

For standalone mode classification you need to apply PTF UQ62753 for APAR PQ46174 and PTF UQ57041 for APAR PQ56008. These service levels give you the ability to WLM classify incoming HTTP requests when the IHS is in standalone (non-scalable) mode. This support is defined using the WLM SN directive in conjunction with the ENCLAVE directive or the ENCLAVE environment variable. The support of this functionality is described in *HTTP Server Planning, Installing, and Using*, SC34-4826.

If you don't use explicit classification, the default CB rules of the WebSphere Application Server in charge would be used.

The HTTP server plugin is the component responsible for dispatching the requests to available J2EE servers. During initialization, the plugin gets information of all the J2EE servers installed and filters out the J2EE servers it is configured to talk to (see Example 5-1 on page 101). At this point, the plugin gets a list of Web applications installed on these servers.

The use of the HTTP server in scalable mode to drive WLM classification allows you to be more granular when assigning performance goals to application requests. See Figure 5-1 on page 83.

Attention: The WLM and other definitions in this section are intended to illustrate our exercises with our server configurations. They may or may not be appropriate for all systems.

In Example 5-1, we list some of the HTTP server directives used in our tests.

Example 5-1 IBM HTTP server directives

```
# prestart queue servers
AppEnvPrestart WEBPROD
AppEnvPrestart WEBTEST
# assign url pattern to specific WLM APPLENV
APPLENV      /WebSphereSamples/TradeSample/Prod/*  WEBPROD
APPLENV      /WebSphereSamples/TradeSample/Test/*  WEBTEST
# set maximum and minimum number of queue servers for the specified WLM APPLENV
AppEnvConfig WEBPROD {
  AppEnvMin 2
  AppEnvMax 4
}
# set maximum and minimum number of queue servers for the specified WLM APPLENV
AppEnvConfig WEBTEST {
  AppEnvMin 1
  AppEnvMax 2
}
```

In Example 5-2 and Example 5-3, we list the application environment definitions for our HTTP server running in scalable mode.

Example 5-2 WEBPROD APPLENV WLM definitions

```
App Environment Name . . WEBPROD
Description . . . . . BBPROD using plugin
Subsystem type . . . . . IWEB
Procedure name . . . . . WEBPROD
Start parameters . . . . IWMSSN=&IWMSSNM,IWMAE=WEBPROD
Limit on starting server address spaces for a subsystem instance:
  No limit
```

Example 5-3 WEBTEST APPLENV WLM definitions

```
App Environment Name . . WEBTEST
Description . . . . . BBTEST using plugin
Subsystem type . . . . . IWEB
Procedure name . . . . . WEBTEST
Start parameters . . . . IWMSSN=&IWMSSNM,IWMAE=WEBTEST
Limit on starting server address spaces for a subsystem instance:
  Single address space per system
```

In Example 5-4, we list the WLM IWEB subsystem classification definitions.

Example 5-4 IWEB WLM subsystem definitions

* Subsystem Type IWEB - WEB Server Test Subsystem

Created by user WELLIE0 on 1996/08/03 at 16:21:11

Last updated by user ANDRE on 2001/10/17 at 09:08:37

Classification:

Default service class is FAST

There is no default report class.

#	Qualifier type	Qualifier name	Starting position	Service Class	Report Class
1	TN	WEBPROD		WASPROD	
1	TN	WEBTEST		WASTEST	

Descriptions:

#	Qualifier type	Qualifier name	Description
1	TN	WEBPROD	
1	TN	WEBTEST	

Descriptions:

#	Qualifier type	Qualifier name	Storage Critical	Manage Region Using Goals Of
1	TN	WEBPROD	NO	TRANSACTION
1	TN	WEBTEST	NO	TRANSACTION

In Example 5-5, we list a sample of the service class definitions used in our exercises. Note that the definition is for service class WASPROD.

Example 5-5 Service class definition sample

* Service Class WASPROD - WEBAPP Prod service class

Created by user ANDRE on 2001/10/02 at 14:48:25

Base last updated by user ANDRE on 2001/10/02 at 14:48:25

Base goal:

CPU Critical flag: NO

#	Duration	Imp	Goal description
---	----------	-----	------------------


```

- -----
1          1    90% complete within 00:00:02.000

```

The result is:

- ▶ The application represented by URI
/WebSphereSamples/TradeSample/Prod/* gets classified to service class WASPROD.
- ▶ The application represented by URI
/WebSphereSamples/TradeSample/Test/* gets classified to service class WASTEST.

These two applications are served from the same HTTP server but use different J2EE servers.

Note: The classification is assigned using IWEB and is passed along with the request. It is then used by WLM to find a suitable J2EE server.

WLM classification using HTTP transport handler or RMI/IIOP

The second way to classify work or transactions is by using WLM CB subsystem classification.

In Example 5-6, work directed to J2EE server BBPROD is classified with the WASPROD service class and work directed to J2EE server BBTEST is classified with the WASTEST service class. All other work gets classified with the WASDFLT service class. A sample service class definition can be found in Example 5-5 on page 102.

Example 5-6 CB subsystem WLM definitions

* Subsystem Type CB - WAS V4 Series classification

Created by user ANDRE on 2001/10/02 at 14:54:45
Last updated by user ANDRE on 2001/10/03 at 16:16:13

Classification:

Default service class is WASDFLT
There is no default report class.

Qualifier #	Qualifier type	Qualifier name	Starting position	Service Class	Report Class
1	CN	BBPROD		WASPROD	
1	CN	BBTEST		WASTEST	

Descriptions:

Qualifier #	Qualifier type	Qualifier name	Description
1	CN	BBPROD	
1	CN	BBTEST	

Descriptions:

Qualifier #	Qualifier type	Qualifier name	Storage Critical	Manage Region Using Goals Of
1	CN	BBPROD	NO	TRANSACTION
1	CN	BBTEST	NO	TRANSACTION

The HTTP Transport Handler currently does not support the following:

- ▶ The authentication policy specified in your Web application's deployment descriptor. As an alternative, your security administrator can define a surrogate user ID under which all requests will execute.
- ▶ HTTPS. This means that your installation cannot use SSL connections for inbound servlet requests.

We expect that this support will be available within the service stream in the second quarter of 2002. Stay tuned for APAR PQ59911 if you need this support soon.

5.3 From test to production

Many of the same considerations that apply to maintaining or upgrading WebSphere Application Server itself also apply to maintaining or upgrading applications. It is equally important, then, to ensure that no errors in the new code adversely affect the production environment. It is also likely that the test will involve new functions, specific groups of users or test data, making isolation between the test and production application code all the more important.

As said before, WebSphere Application Server has full sysplex support, matching or exceeding that of any other application environment. This includes its ability to support replicated instances of itself across the sysplex (or on the same system), with multiple levels of workload balancing, access to shared data, and single system image.

5.3.1 Our test and production environment

In Figure 5-15, we illustrate how we set up our environment:

- ▶ A two-way sysplex called Sandbox
- ▶ A test J2EE server called BBTEST

This server was configured to run only on system SC63.

- ▶ A production J2EE server called BBPROD

This server was configured to run in both SC63 and SC64.

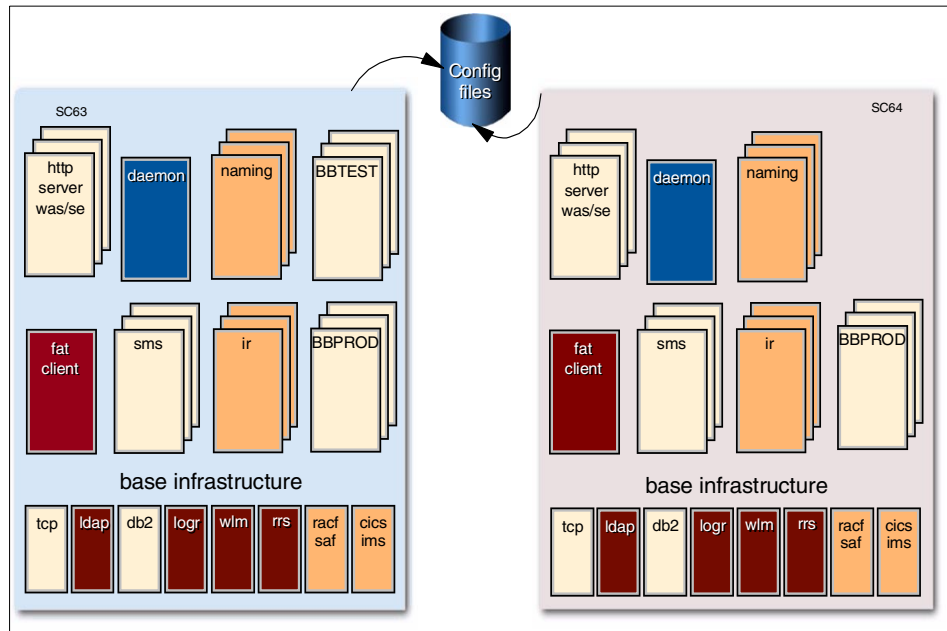


Figure 5-15 Test and production environments

Figure 5-15 shows that if we install an application in BBPROD, it will become part of each of the BBPROD server instances.

5.3.2 Scripted configuration

The scripting interface to System Management is a way to perform functions that are normally performed through the SMUI interface on a PC. You can perform almost all of the functions through the scripting API that you can do with the SMUI. One exception is performing resource and reference resolutions for the .ear file.

Both operations and administration functions can be performed in scripts. The REXX scripts can be run in any shell environment and also in batch.

The SMAPI brings new enhancements into the deployment process. To get started with this new functionality we followed the steps to set up the client environment as explained in *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management Scripting API*, SA22-7839. The instructions in the manual directed us to use *EXPORT* commands in the OMVS environment. This included updates to CLASSPATH, LIBPATH, PATH, and DEFAULT_CLIENT_XML_PATH. We strongly recommend that you put all of these environment variables in your .profile file or into the REXX API script.

Instead of using the **echo** command to check the environment variables as shown in the SMAPI manual, we encourage you to use the **set** command to display your environment variables. It can be used under OMVS and under ISHELL. To run the **set** command under ISH, choose TOOLS on the pull-down menu and enter 2 (Run Shell Command(SH)). Then type **set** on the command line and press Enter. Your environment variables will then be displayed.

The other way to get to the Run Shell Command line is to enter SH on the command line of the first/home panel of ISH. You can then type **set** and press Enter to display environment variables.

There is an environment variable that was not mentioned in the scripting API manual, which must have been an assumption. When executing our first script, we got the error message `java: FSUM7351 not found`. To use Java, the correct Java files must be in the PATH environment variable. Also, you cannot simply use the path names in the manual. You must use the paths that may be unique to your installation. For example, we added the following to the PATH variable:
`/usr/lpp/java213/IBM/J1.3/bin.`

The other preparation task for using SMAPI is to create a new system administrator ID. The SMAPI manual referred us to *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834. We followed the steps there and a new system administrator ID was created. When you are logged on to the SMUI, you can only see the current conversation and conversations created by the current logged on administrator ID. The case will be the same for the user that executes the SMAPI REXX scripts.

To get experience with SMAPI, we used the examples in `/usr/lpp/WebSphere/samples/smapi`. We coded our own routines to perform almost all of the list functions available in the API. This gave us experience with the command interface, configure interface, and the XML calls. This experience is useful before starting the more complex functions that do additions, changes or deletes.

During the writing of the list functions, we would get errors that didn't match with the source. Almost by accident we found data in the file that was to the right of the display screen. Be sure to look to the right of the typical expected 80-byte record. There might be some data there that you don't want.

One script that we designed performs three functions:

1. Creates a new conversation.
2. Deletes an application.
3. Activates the new conversation.

Sometimes when an application is deployed with errors, the application server may fail to start. A script to delete the defective application and get the server back up and running is useful. See 5.3.3, "Scripted application fallback" on page 107 for more details on how to do this.

These scripts write out information about results to a temporary file. Make sure that the user has access to the /tmp/ directory.

5.3.3 Scripted application fallback

In 5.2, "Setting up J2EE servers on z/OS" on page 85, we discussed the creation of new J2EE application servers in test and production environments. We also detailed how to install a J2EE application under a J2EE application server.

In step b. on page 90 we described a point during the J2EE application install when you can save a resolved .ear file. This .ear file is the key for our scripted fallback. The resolved .ear file contains all updates related to references and resource resolutions like the JNDI path and the J2EE DB2 resource. Given that there are no changes to those resources and that there are no other modifications to the resources required by the application represented in the resolved .ear file, this .ear file represents a version of the application that is functioning properly.

We performed the following steps to verify our application fallback procedure:

1. Assumptions
 - We used the naming standards described in Appendix B, "J2EE server naming convention and customization" on page 193.
 - The .ear filename standard is *applicationname_VxRy*, where:
 - *applicationname* is the application name.
 - *x* and *y* are sequential indexes used for reference.
 - .ear files used are:

- TradeSampleProd_V1R0.ear represents the working version of the application.
 - TradeSampleProd_V1R0_resolved.ear for V1R0.
 - TradeSampleProd_V1R1.ear represents the new but faulty version of the application.
 - TradeSampleProd_V1R1_resolved.ear for V1R1.
- A working version of the application is already installed and verified. This application is represented by .ear file TradeSampleProd_V1R0_resolved.ear for fallback purposes.
 - The resolved .ear file for the working version of the application (TradeSampleProd_V1R0_resolved.ear) must be available in the z/OS HFS file system. We placed it under /WebSphere390/EAR/.
2. We followed the instructions described in step 6. on page 89 to install a new version of the application. TradeSampleProd_V1R1.ear contains this version. This is our faulty application as described earlier. We changed the HTML page to show the version number. Figure 5-16 on page 109 shows what it looked like after installing V1R1.

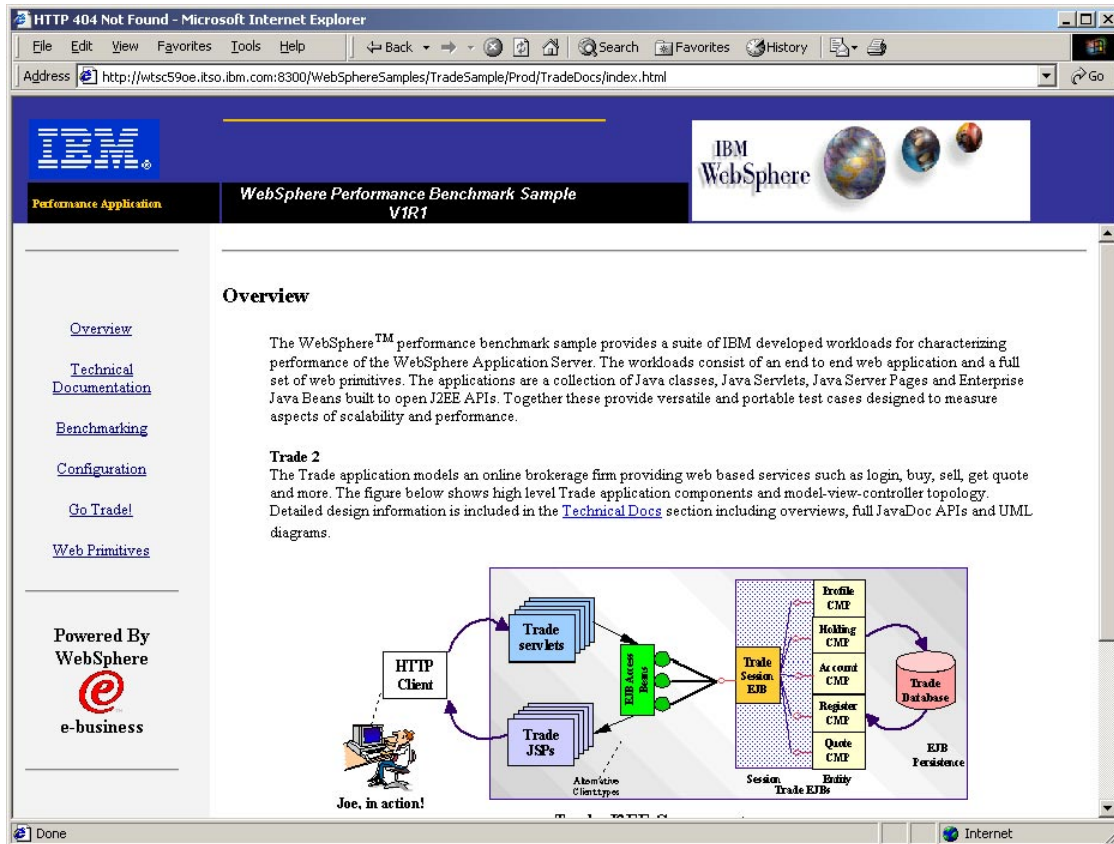


Figure 5-16 TradeSampleProd V1R1

3. We assumed that V1R1 is defective and wanted to fall back to V1R0. In 5.3.2, “Scripted configuration” on page 105, we described some of the SMAPI functionality.

We developed a script to perform the whole process of deleting the offending application (V1R1) and re-install the working application (V1R0). The whole script is listed in Example: C-3, “Fallback SMAPI script” on page 203. The script performs the following actions:

- Creates a first conversation.
- Deletes the offending application in a specific server.
- Processes resolved .ear file to restore V1R0 of the application.
- Commits and activates a second conversation.

Note: After finalizing the scripting code we figured out that you can accomplish this within one conversation.

The results of the script are in Example 5-7.

Example 5-7 Output of the script

```
Starting create/delete/activate/create/add/activate script
Creating conversation Delete ...

Conversation Delete created.
Deleting Application TradeSampleProd_V1R1 ...

Delete application TradeSampleProd_V1R1 complete.
Committing conversation Delete ...

Conversation committed and activated.
Creating conversation ReInstall ...

Conversation ReInstall created.
Begin process ear file /WebSphere390/EAR/TradeSampleProd_V1R0_resolved.ear ...

Process ear file complete.
Committing conversation ReInstall ...

Conversation ReInstall committed and activated.
SMAPI013 script completed!
```

We chose to submit this script in batch mode, and the user executing the script must be defined as administrator as described in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834. We provide a sample of this batch job in “JCL used to run SMAPI samples in batch” on page 202.

4. We verified the fallback procedure by visiting the application homepage. Figure 5-17 on page 111 shows the welcome page, and the version of the application is displayed.

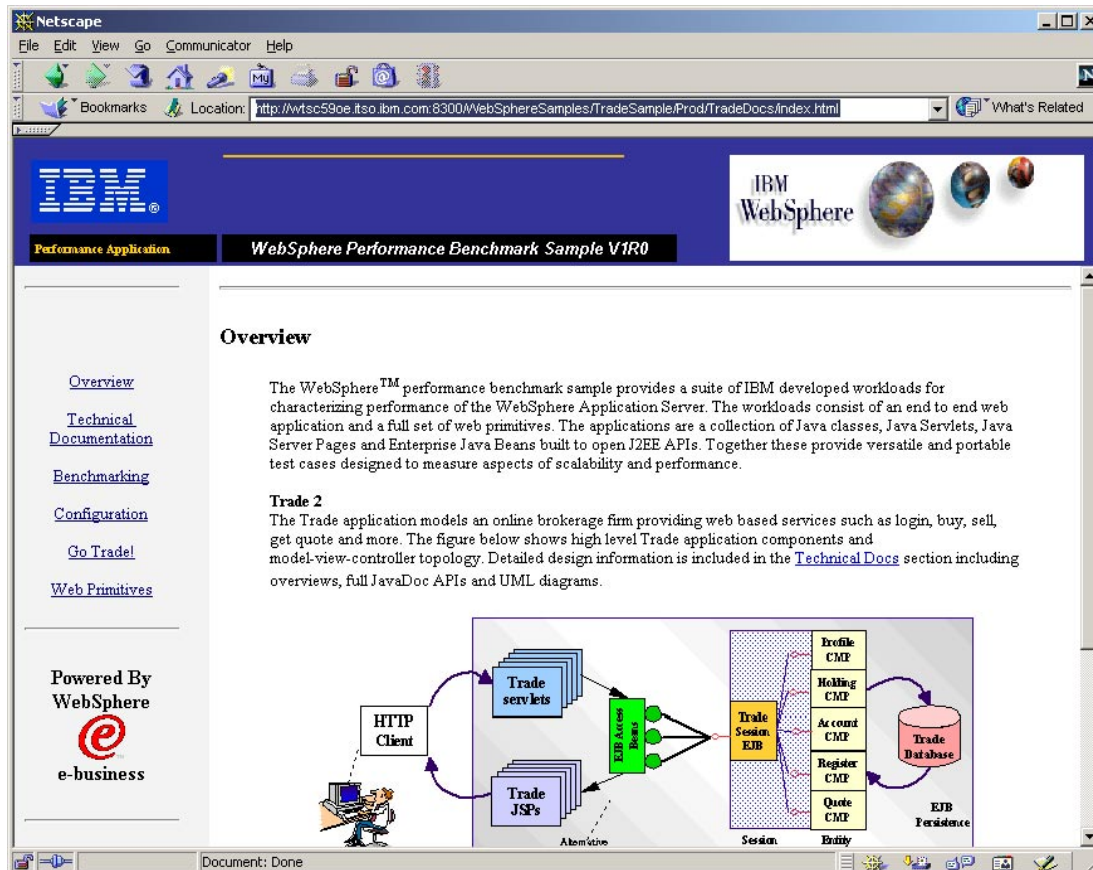


Figure 5-17 Application after successful fallback procedure

Note: We created a two-conversation script because there was no information available whether deleting and adding a J2EE application in the same conversation was supported. However, APAR PQ50956 (PTF UQ57051) addresses this issue now.

This procedure was sufficient for our sample application. Certainly, there are other resources to consider when doing a fallback of an application, like application data that could have been compromised.

IBM WebSphere Application Server for z/OS and OS/390 definitions outside the scope of the J2EE application definitions have to be taken into consideration. One example is the J2EE DB2 data source definitions. According to *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management Scripting API*, SA22-7839, it is possible to define a J2EE resource using the SMAPI. However, we did not verify this during our exercise.

5.3.4 Production infrastructure

We faced some difficulties during our tests using our test and production environment, both in monoplex and sysplex. As we described previously, IBM WebSphere Application Server for z/OS and OS/390 uses many of the advanced features of the operating system and its underlying infrastructure, both hardware and software. Difficulties we encountered are listed below, with a short explanation of how we fixed them.

- ▶ Check DB2's system configuration and change it to the recommended values documented in *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834. The parameters and their recommended values are:
 - IDBACK = 500
 - CTHREAD = 700
- ▶ Evaluate how many LDAP connections you expect on your WebSphere Application Server node, for both production and test. The sample LDAP configuration provided in with IBM WebSphere Application Server for z/OS and OS/390 defaults values that may not suffice in a larger environment.

The LDAP connections can be evaluated by verifying the following points:

- Every client of a Web application server instance uses one LDAP connection each. This does *not* include clients accessing the Web application through a Web browser.
- Every J2EE application server uses at least one LDAP connection per server instance or server region.
- Each System Management Enhanced User Interface instance uses one connection to LDAP.
- Every PLUGIN instance uses one LDAP connection.

The LDAP configuration directives are often located at:

`/WebSphere390/CB390/<sysplex>/etc/ldap/`

and are called:

`<sysname>.bboslapd.conf`

where *sysplex* is the name of the sysplex and *sysname* is the name of the system (LPAR).

In our lab environment, running z/OS 1.2, the only directive changed was:

maxConnections, from 10 to 100. If you leave it at 10, LDAP will automatically change this to the default value of 16384.

If you are are running with an earlier version than z/OS 1.2, you should also review the following directives:

- *maxThreads*, from 10 to 100 (be sure *maxThreads* is at least the same value as *maxConnections*)
- *waitingthreads*

5.3.5 Automatic application verification

After the activation of the application the operator needs to make sure that it is working properly. If it does not work as desired, the operator needs to take appropriate steps. In case the activation was done on a test system, the operator informs the deployer about the unsuccessful test run. If the application was activated on the production system, the operator uses the scripted fallback (5.3.3, “Scripted application fallback” on page 107) to establish a system state prior to the installation of the application.

Our understanding of operator and deployer is discussed in 1.3.3, “Implementation job roles” on page 10.

Verification of a webapplication

The goal of application verification is to ensure, to a certain degree, that this functionality is really available to the remote user. For a complex webapplication it is not possible to test every aspect of every functionality of the application. Therefore, you test a subset of these aspects that represents the whole functionality as well as possible.

To get a picture of what should be in this subset, consider the following questions:

- ▶ Where is the data stored that is used by the application? What data sources are used in which cases?
- ▶ How is session management handled? Which actions make use of this?
- ▶ What security mechanisms are used? What kind of access is granted and what kind is denied?
- ▶ What are the basic functionalities that are typically used by a user?

- ▶ What are high-level functionalities that only work if the majority of low-level functionalities such as data sources, session management, business logic, and security work properly?
- ▶ What is the access point of a user and from where can you access the application under similar conditions?

The answers to these questions enable you to define a test like the one described in the following section.

Verification methods

To verify that a webapplication is working correctly, different means are used on different levels. Unit tests performed on the Java class level ensure that tested Java methods are working correctly.

Functional application tests check whether high-level functionalities are working. These functionalities invoke a number of methods and rely on a subsystem. Due to this fact, the subsystem is partially tested by these tests as well.

Load tests focus on the reliability and performance of the application when put under a heavy load. Sometimes bugs like thread safety issues pop up only when the application is tested this way.

Verification tools

There are plenty of tools that help you perform the kinds of tests mentioned in the previous section. Though we did not use a third-party test tool with the Trade2 sample, we mention two helpful open source tools which can be used for testing webapplications deployed on z/OS.

- ▶ JUnit (see <http://www.junit.org>) is a Java package that is widely used in the industry to perform automated unit tests. Programmers build JUnit test methods into their classes so that these classes can be tested by anyone who performs a JUnit test on the code.
- ▶ Distributed Application Tester (DAT) (see www.alphaworks.ibm.com) is a graphical tool that files HTTP requests onto a webapplication and evaluates the responses. You can record your clicks and navigation through a Web site and replay this, simulating a large number of users. DAT can apply lists with user credentials to the tests, simulating different users that log on to a webapplication.

However, for the Trade2 example we did not use a third-party tool for tests, but a self-written test client. This preserves the opportunity to integrate this test client into batch jobs in z/OS that might be used to deploy and activate the application.

Test client for Trade2

Our test client for Trade2 was an executable Java class that performs six HTTP requests to Trade2 and evaluates the responses. Every response is checked for a keyword or key term that only appears in the response if the application is working properly.

To place a single request and to look for the keyword, the method `CheckAnURL` is used. It returns `True` whenever the desired keyword is found in the response and no exception occurred.

The main method of this executable class accepts the base URL of the current Trade2 deployment as a parameter. It attaches different relative paths to this URL and calls `CheckAnURL` six times. Each time it supplies a different URL and a different keyword.

If all tests were successful, it reports this success to `STDOUT`. If not, it reports the failure to `STDOUT`. The operator uses this information to decide on a fallback to a system status before the new webapplication was made available for use.

In addition to the message to `STDOUT`, the client exits with return code 0 if the test was successful and with return code 8 if it wasn't. Use this mechanism when automating the use of the test client.

See "How to use the test client for Trade2 from a batch job" on page 116 for a way to run the client detached.

How to modify the test client for other Web applications

Modify the test client to test another webapplication by editing the Java source code. Download the code from the supporting Web site of this book or copy it from "The test client source code" on page 178.

Find out which URLs represent the most important parts of your Web application so that the request response indicates whether the application is working correctly or not. Find out which keywords appear in the response whenever the application is running correctly.

Define the common prefix for all URLs. Usually this is the domain name, and eventually also the path to the servlet directory. Split all URLs into the common prefix and the variable extension.

Example:

If your test URLs are

- ▶ `http://demosever.com/webapps/myapp/servlet/Login?username=carl`
- ▶ `http://demosever.com/webapps/myapp/servlet/ShowQuote?symbol=ied`

- ▶ `http://demoserver.com/webapps/myapp/servlet/ResetAccounts`

then your common prefix is

`http://demoserver.com/webapps/myapp/servlet/`

and your variable extensions are

- ▶ `Login?username=carl`
- ▶ `ShowQuote?symbol=ied`
- ▶ `ResetAccounts`

Edit the main method so that you have three test case blocks. In each block assign one variable extension to the string *urlToCall* and the corresponding keyword to *patternToAppear*.

Edit the following assignment:

```
boolean success = testCase1 & testCase2 & testCase3
```

and deploy the class to your test system. Run it by typing

```
java Trade2TestClient http://demoserver.com/webapps/myapp/servlet/
```

Check the console for the results of the test.

How to use the test client for Trade2 from a command line

The best practice is, of course, to run the test client on a system that closely approximates the system of the respective users. Install a JRE 1.3 or higher on that system, place the class `Trade2TestClient` in a directory that is included in the `CLASSPATH`, and run the class supplying the basic servlet path of the Trade2 application as a command line parameter:

```
java Trade2TestClient http://demoserver.com/webapps/trade2/servlet/
```

The test client then performs the tests by calling different URLs of the Trade2 application. If all tests are successful, it writes a simple success message to `STDOUT`. In case one of the tests fails, it writes out an error message.

How to use the test client for Trade2 from a batch job

To automate the application test, we used a batch job to launch a shell script that launches the `Trade2estClient`.

The shell script we used was named `abp1.sh`:

Example 5-8 abp1.sh

```
#-----
```

```

# Application Verification Program Caller
#
# Change JAVA_HOME, URL and the first CLASSPATH occurrence
#
#-----

# Put Java home directory here
export JAVA_HOME=/usr/lpp/java/IBM/J1.3

# Put your Application Verification Program in the classpath
export CLASSPATH=$CLASSPATH:/u/stfan

# Put the URL to check
URL='http://tot14/WebSphereSamples/TradeSample/servlet/'

# Do not change these lines
export PATH=$PATH:$JAVA_HOME/bin

echo 'Starting Application Verification Program on URL '$URL'.'
java Trade2TestClient $URL

```

To call it we used the JCL in Example 5-9:

Example 5-9 JCL example for running the test client

```

//AVPBAT1 JOB (999,POK),'SF',CLASS=A,REGION=OM,MSGCLASS=A,
//          NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//AVPS1 EXEC PGM=IKJEFT01
//SYSEXEC DD DSN=SYS1.SBPXEXEC,DISP=SHR
//STDENV DD PATH='/u/stfan/.profile',PATHOPTS=(ORDONLY)
//SYSTSPRT DD SYSOUT=*
//STDOUT DD PATH='/tmp/avp1.out',PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//          PATHMODE=SIRWXU
//STDERR DD PATH='/tmp/avp1.err',PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//          PATHMODE=SIRWXU
//SYSTSIN DD *
BPXBATCH SH /u/stfan/avp1.sh
/*
//

```

5.4 Nondisruptive upgrade of an application

We did not find any generic safe way to upgrade a J2EE application without interrupting service.

The general considerations for this are:

- ▶ When any modification is made to a J2EE server definition, WebSphere Application Server runtime will restart the server instances of this server when the modification conversation is activated. If the changed server has several active instances at this time, they are restarted one after another: WebSphere Application Server waits until the control region of an instance is up again before restarting another instance of the server, and so on until all instances have been restarted.
- ▶ According to SMUI Help: "If you deleted a J2EE application from a J2EE server and want to reinstall it, you need to delete the J2EE application in one conversation, activate it, and create another conversation to reinstall the J2EE application, then activate it."

As a result, if the application is hosted in only one J2EE server, it will not be available at least during the time this J2EE server instance is recycled.

We divided the more specific considerations according to application structure:

- ▶ Client accesses the application through HTTP.
- ▶ Client is some Java code communicating only with EJBs.

In both cases, you should remember that when you stop a J2EE server with "in-session" session beans, they die and the client code loses any access to them.

You need also to consider whether the application modifications involve some backend modification. In such a case, the service needs usually to be interrupted.

5.4.1 HTTP access-based applications

These applications use servlets/JSPs to manage the presentation layer, and they access J2EE server local EJBs.

As a result, the client is HTTP-based, and the way it knows that the application is alive is that HTTP requests are correctly handled.

So to keep the application uninterrupted, ensure that:

- ▶ The HTTP plug-in finds a J2EE server with a matching Web application context root. You can deploy an upgraded version of the application in one J2EE server instance with the same context root as another instance with the earlier version of the application. The IBM WebSphere Application Server for z/OS and OS/390 plug-in balances the HTTP calls among all the J2EE server instances advertising the same context root. So, there may be a short interval where the plug-in might route balance requests arbitrarily among different levels of the application.

- The user-specific data (session data) is propagated around J2EE servers. In order to save user sessions, you can configure persistent HTTP sessions.

5.4.2 RMI/IIOP access-based applications

These applications are EJBs accessed by a Java client outside the J2EE container.

In this case a strategy could be to code an alternate name for EJB lookups in the client program. If anything goes wrong with an EJB found using the regular name, the client should try to look up the EJB on its alternate name, which will bring it to another J2EE server. With such a mechanism you can have two J2EE servers with different levels of the same application. EJBs from the first server will be reachable from the regular name. EJBs from the second server will be reached using an alternate EJB name.

To choose the application version level you want to use, you would only have to start the right J2EE server.

5.5 Disruptive upgrade of WSAS runtime

By definition, the disruptive upgrade of the WSAS runtime is the only choice on a monoplex. There is no other system to provide service to the client when the monoplex is being updated. In a sysplex configuration with multiple WSAS instances, each WSAS instance is upgraded one at a time, which is often referred to as a rolling upgrade. One or more WSAS instances in the sysplex would always be active servicing requests from clients.

Nondisruptive upgrade of the WSAS runtime is described in 5.6, “Nondisruptive upgrade of the WSAS runtime” on page 120. We describe our experience with a disruptive upgrade of WSAS 4.0 to WSAS 4.01 on a monoplex system.

We started the upgrade process after the SMP/E¹ installation of all the libraries and data sets was completed successfully. The information for the SMP/E process is in the *WebSphere Application Server V4.0.1 for z/OS and OS/390 Program Directory*, G110-0680. The data set names of release 4.0 have the prefix BBO. The data set names of release 4.0.1 have the prefix BBO.V401.PID.

¹ SMP/E (System Modification Product / Extended), is an IBM product for z/OS and OS/390 used to control the software repository or your z/OS and OS/390 system images. SMP/E is able to provide a controlled process for building new product libraries, protects the integrity of your system software libraries, and allows you to query the status of your installed software.

The two manuals that we referenced for this installation are the *WebSphere Application Server V4.0.1 for z/OS and OS/390: Migration*, GA22-7860 and *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834.

Chapter 11 of *WebSphere Application Server V4.0.1 for z/OS and OS/390: Migration*, GA22-7860 describes the basic steps to migrate from WebSphere for z/OS V4.0 to WebSphere for z/OS V4.0.1. One of the key points to watch for is the method type that is required to migrate from one functional level of WebSphere z/OS to another. There are three methods: cold start, warm start, and hot start. The documentation clearly states that the upgrade from V4.0 to V4.0.1 requires a warm start.

All the necessary steps needed for a runtime upgrade are dictated by the WebSphere ISPF dialogs. The dialogs were introduced with WebSphere Application Server Version 4.01 but can be retrofitted to WebSphere Application Server Version 4.0 with the PTF PQ48858.

We provide a sample of the ISPF wizard-generated instructions in Appendix E, “ISPF wizard-driven runtime upgrade” on page 219. Follow the instructions in the dialogs and submit the generated jobs.

After you finished the whole process, run the IVP to ensure a successful runtime upgrade.

5.6 Nondisruptive upgrade of the WSAS runtime

Nondisruptive upgrade of the WSAS runtime is possible only if the WSAS runtime is sysplex. The procedure is similar to what we described in 5.5, “Disruptive upgrade of WSAS runtime” on page 119 but, as we upgrade one system at a time, applications can continue to run on the other systems belonging to this sysplex.

5.7 Change and problem management

This section discusses three related concepts concerning applications installed in WebSphere Application Server for z/OS and OS/390. Content management is the process of creating, managing and maintaining application components separately from their execution or presentation environment.

Software configuration management or change management is the process of managing and controlling changes to applications and their components, especially in the execution or presentation environment. Ideally, an end-to-end software configuration management process encompasses both content management and change management, ensuring that only managed content becomes part of the execution environment, and that this occurs only in the scope of procedures that are well understood and auditable.

Not very many years ago, application components consisted primarily of source code that was compiled into object code and linked into load modules or load libraries. In some cases, these applications were linked with data in other types of files, such as panels. In this environment, it was fairly straightforward to envision a software configuration management (SCM) tool that would include not only library management of the components, but comprehensive build and install processes.

The invention of the World Wide Web and the Java programming language complicated the picture somewhat by introducing new types of executable code and displayable data. But with the addition of a few new data types to the content repository and a few new processing rules, this new type of content was manageable.

Java 2 Enterprise Edition (J2EE) introduces a new programming model that has several advantages over its predecessors. It is truly platform neutral and portable. It is dynamic, allowing the deferment of linkages between application components (even components deployed geographically and across heterogeneous platforms) until execution time. It also provides new standards for describing the configuration and characteristics of applications. These new features present challenges that software configuration management tools have not fully caught up with.

This section discusses how these challenges can be addressed with current tools and capabilities for applications deployed in WebSphere Application Server for z/OS and OS/390. We also suggest how tooling and interfaces might be enhanced in the future to further automate the SCM process.

5.7.1 Deployment principles

We explain J2EE deployment principles in this section because the deployment is a crucial part of the change process. Note that you can also activate a deployed application using the WLM environment. We don't discuss this possibility in detail.

Figure 5-18 is a pictorial representation of the development, deployment and test process. Individual components are developed with an editor or in a development environment such as WebSphere Studio Application Developer. Web applications, consisting of components such as servlets, Java server pages (JSPs), text (html), images (.gif, .jpeg, etc.) and so forth can be created by integrating these components, typically with a tool such as WebSphere Studio or WebSphere Studio Workbench. For a J2EE Web application, a configuration descriptor (WEB-INF/web.xml) is created to describe the content and structure of the application. The result is a Web application repository or .war file.

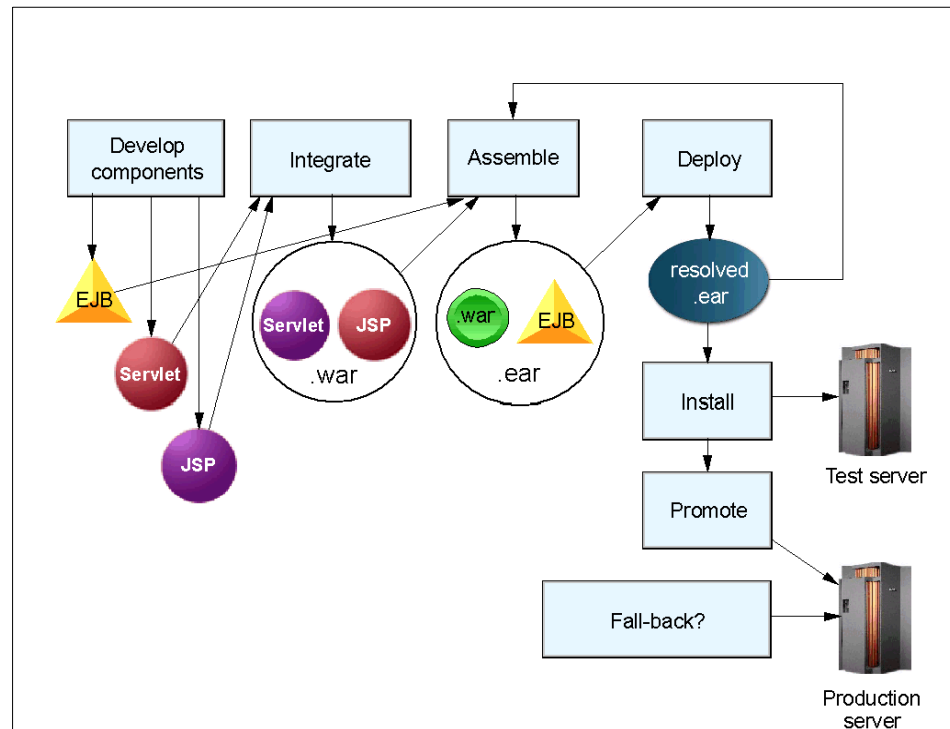


Figure 5-18 Development, deployment, and test process

Web applications (.war files) can then be combined with EJB applications (.jar files) into an enterprise application repository or .ear file. For deployment in the IBM WebSphere Application Server for z/OS and OS/390, this function is performed by the Application Assembly Tool (AAT). The AAT allows editing of the EJB deployment descriptor files and Web application configuration descriptor files. This includes, for example, resolving references in the .war files to EJBs in the .jar files. It also includes setting descriptor values, such as method permissions and transaction attributes, that the J2EE specification determines belong to the application assembly phase.

The AAT can also create stub and tie classes, as well as persistence classes, that are necessary for the J2EE application to function within the z/OS WebSphere container.

The .ear file must then be processed by System Management Enhanced User Interface Administration Application for z/OS. This is necessary in order to resolve references to resources outside of the .ear file, such as data sources and other EJBs. Once the references are resolved (using a GUI interface called the Administration Application), the resolved .ear file is saved on the workstation and a copy is sent, via ftp, to the WebSphere server.

The resolved .ear file can later be fed back into the AAT, and modified .jar and .war file components can be introduced as necessary for fixes or upgrades.

The resolved .ear file can be installed into a test server, and after testing is successful, it can be installed (potentially with minor changes) into a production server. Earlier versions of the resolved .ear file must be maintained in order to allow fall-back in case there is a failure of the new version in production (refer to 5.3.3, “Scripted application fallback” on page 107 for more information on this).

- ▶ The paramount unit of change for J2EE applications is the .ear file. J2EE applications may also use common utility Java classes packaged separately in .jar files. The only supported way an .ear file can be deployed into a J2EE server is via the System Management server. It cannot be installed, nor can its contents be interrogated or manipulated, by end users or other tools.
- ▶ There is only one level of protection for the System Management server. Within a WSAS node, any user that has the authority to access the System Management server can perform all functions related to all aspects of the configuration.
- ▶ Only a conversation that was created based on the current active conversation can be activated. In other words, when a conversation is activated, all other conversations become obsolete and useless. Only the current conversation and conversations created by that user can be seen or accessed.
- ▶ Each component (EJB, Web application, resource, etc.) must have a JNDI name that is unique within the name space. The name space includes the WSAS node and any federated name spaces.

5.7.2 Recommendations for a controlled deployment

To get the deployment and activation process under change management control you have to follow some guidelines:

- ▶ First and foremost, library management must be introduced into the picture. This means introducing an SCM repository (such as the IBM SCLM for z/OS)

into one or more stages to manage .ear files and the utility .jar files that their applications depend on through the test, production and fall-back phases of the cycle.

- ▶ Accordingly, we recommend that once reference resolution is accomplished in the system management administration application (each component has a green check mark), the resolved .ear file should be saved in the SCM repository (refer to step b. on page 90 for more on the resolved .ear file).
- ▶ Likewise, the resolved .ear file should not be installed into the J2EE server using the System Management Enhanced User Interface Administration Application GUI. Instead, the SCM tool should be invoked to install the application by obtaining the .ear file from the SCM repository and invoking a batch program to process it via the SMAPI described below.
- ▶ The installation can decide at what other points an SCM repository should be utilized. The accompanying diagram is modified to show the output of each development phase being stored in an SCM repository. The installation may choose to skip some of these repository points or to use another repository for some stages.

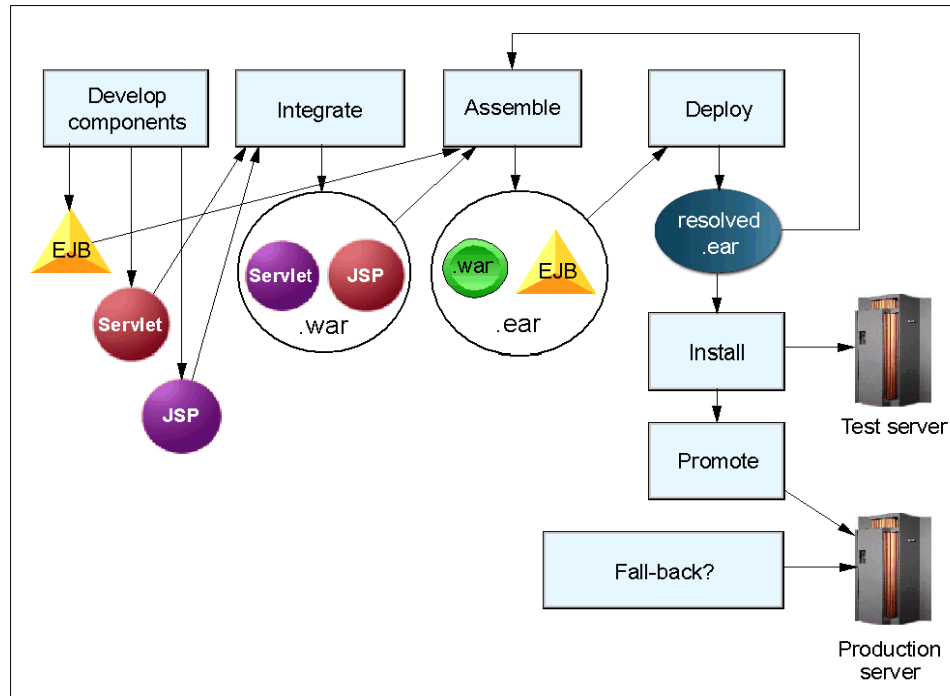


Figure 5-19 Modified cycle including SCM

- ▶ For each WSAS node, identify a WSAS administrator job responsibility assigned to a limited number of people. Only administrators (and perhaps their supervisors) will know the userid/password necessary to access the System Management server in that node. This responsibility must match a defined change control process that will ensure that only approved changes to the WSAS configuration are allowed and only approved software changes are made. Ensure that the files and directories referenced by each server's classpath are protected from modification except by WSAS administrators.
- ▶ Insofar as possible, make all configuration and application changes using the System Management Scripting API, including the following steps, in a single operation:
 - Create a new conversation.
 - Make configuration modifications and process .ear files.
 - Validate the conversation.
 - Commit the conversation.
 - Indicate tasks completed.
 - Activate the conversation.

Note that for this to be successful, offline tasks must be accomplished separately, preferably before starting the above change operation.

5.7.3 Deployment integration into change management

To get WebSphere application deployment under change management control, you need to integrate the deployment process into your software management infrastructure. If you like to follow our approach, proceed as described in this section; see Figure 5-20 on page 126.

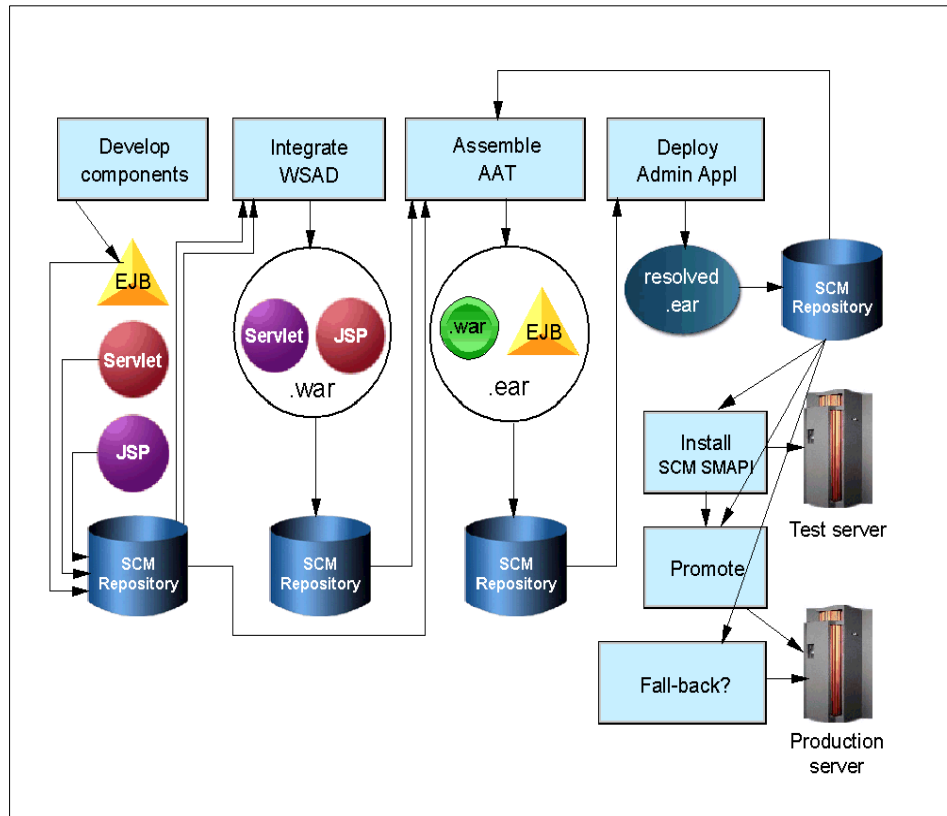


Figure 5-20 Outline of change procedures

1. Defining/modifying a test server
 - a. Create control region and server region proclib entries.
 - b. Add or update RACF profiles (CBIND, STARTED, etc.) as necessary.
 - c. Create a new WLM application definition.
 - d. Use a batch job to create a server as a single operation (see the DefTest.rexx sample).
 - e. Use a batch job to list the test server to an XML file and store it under SCM control.
2. Other configuration changes

Use a batch job to perform configuration changes using scripts stored under SCM control. Each script should be accompanied by an “undo” script in order to maintain a back-out capability.
3. Installing an application in a test server

Use the SCM build/install function to fetch appropriate .ear and .jar files from the repository and invoke a batch job to process .ear files in a single operation.

4. Defining a production server

Use a batch job to fetch the test server XML file from the SCM, modify it as necessary, store it under SCM control, and invoke a server create script. The source could be created by using export and import functions of the System Management Enhanced User Interface Administration Application GUI or by using some sort of template script edited to reflect the production server. Refer to “Create new application server” on page 209 for an example.

5. Promoting an application from test to test (within a node)

This should be the same as installing an application in a test server, utilizing the SCM's built-in promote logic.

6. Promoting an application from test to production

This should be the same as installing an application in a test server, utilizing the SCM's built-in promote logic.

7. Determining what's installed

At this time, only the SCM would have access to the required information.

8. Backing out configuration changes in production

The SCM will maintain multiple levels of the configuration scripts.

9. Backing out application changes in production

The SCM needs to maintain fall-back levels of .ear and .jar files and to be able to use the existing install application logic to install these levels when necessary.

5.7.4 Problems not addressed (and directions for future tooling)

- It would be useful if, following the application assembly (AAT) and reference resolution (part of the system management administration application) we could somehow isolate the configuration and deployment descriptor files so that they could be re-used after a modification.

For example, if the component provider (application developer) ships a new version of an EJB, that EJB contains a deployment descriptor without any of the resolutions or modifications that were made in the AAT to the previous version. It would be good to merge the new DD with the old one in some intelligent way (or at least to use the old one in the case where there were no changes required) in order to eliminate tedious manual and error-prone work. Similar considerations apply to reference resolution at deployment time.

- ▶ In a similar vein, it might be useful to have a batch, automated, or command-driven way to do reference resolution (e.g., a batch editing capability for the DD). This would be especially useful in cases of large applications (e.g., lots of EJBs), where we plan to use mostly defaults. Manual reference resolution is tedious.
- ▶ Because the application assembly and deployment logic is embedded in special-purpose tools, such as the AAT and the system management administration application, it is currently difficult for SCM tools to do automated builds of applications or to maintain an understanding of the relationship between individual components and the resulting applications. Richer programmatic interfaces into AAT and system management administration application could allow integration into frameworks such as WebSphere Studio Workbench. SCM tools could plug in to such frameworks and thus obtain broader control over the end-to-end process.



Management summary

eBusiness workloads, as represented by WebSphere Application Servers, are beginning to impact your organization. Application departments now have the tools to create portable J2EE applications.

But how will other parts of your organization react to these new applications? Data processing has always been an industry that has had to react to change, sometimes radical change. You must ask whether the latest changes are so radical and so different that your current structure cannot absorb it without massive stress and unacceptable incremental costs. Can your current workloads, personnel, disaster recovery processes and other established business processes cope effectively with the challenges and opportunities represented by WebSphere?

6.1 Operational view

By their nature, z/OS environments have a well-coordinated operations organization. At many installations, operational support is already at a level of 24x7x365. The skills and tasks for negotiating and writing service level agreements, monitoring workloads, notifications and escalation of problems are well established.

Operations will be able to understand WebSphere as a new workload and incorporate it into the existing workload. The disciplines that operations already have will not have to be modified for WebSphere in any significant way. Obviously, some new terms and technology will be learned in the process of including a WebSphere application into the production workload. The disciplines of production—in the z/OS sense—are extremely valuable to impose on and implement with WebSphere applications. In fact, this is one of the benefits of hosting or moving an application to z/OS or OS/390. The quality of service provided by operations is precisely what you are looking for to add to WebSphere workloads.

6.2 Technical view

As a technical manager of OS/390 systems, you deal with a wide range of disciplines. You are much like a conductor of a large orchestra. It is rare for a conductor to play any instrument as skillfully as the actual performers in the orchestra. There are usually some instruments that the conductor cannot play at all. But the conductor, and the technical manager, must have a keen eye and ear for the coordination of the individual players.

This is the challenge and opportunity of the z/OS environment. The complexity of this environment, which is also robust and flexible, requires some special skills and the coordination of those skills. In fact, coordination of technical skills is required to a larger extent with WebSphere than with any other workload in the past.

The emergence of e-business has put new requirements on technicians and their managers. Deploying and running WebSphere applications is another new workload that can be added to the mix of business applications that are already running successfully on your system. There are new technologies to be learned for sure, but they are no more complex than the ones previously learned and implemented.

There are new programmers and a new programming language: Java; and it runs in a different environment: WebSphere. There is a new instrument in the orchestra. The challenge is to learn how to make music with the new players. There is nothing more grand than a well tuned and conducted orchestra. We have given you some tips in this redbook on just how to do that.

Hopefully, this redbook has shown you some of the characteristics of the new instrument or technology. Our experience has shown that an application written for other environments and deployed on other environments does migrate to z/OS. If applications are written to standards and follow some very simple portability guidelines, they are surprisingly easy to deploy on z/OS.

6.3 Total cost of ownership vs. cost of migration

The total cost of implementing an e-business application on multiple, small servers is often perceived as less than the costs incurred with implementing the same application on a robust and redundant OS/390 server. When costs (or other reasons) indicated that migrating an application to OS/390 is the best solution, the costs of rewriting and porting used to be prohibitive. Migration used to entail massive rewrites of code. Costs had the potential to become astronomical.

With the standards, architecture and promises of portable J2EE applications now a reality, as our experience during this project attests, there are no longer any massive rewrites. The freedom of choice of where an application runs is now a reality. Applications can easily be deployed on different platforms, inheriting different runtime characteristics.

If your business processes are currently running on z/OS, you can be assured that emerging e-business technologies will also run there. If your complex, transactional e-business processes are not currently running on z/OS, you now have a clear choice on where to host your WebSphere applications.



Part 3

Tools for WebSphere Application Server on z/OS

After installing your application under IBM WebSphere Application Server for z/OS and OS/390, you'll need specialized tools to adjust it for a production environment. In this part of the book, we discuss those tools.



Jinsight

Jinsight is a tool for visualizing and analyzing the execution of Java programs. It is useful for performance analysis, memory leak diagnosis, debugging, or any task in which you need to better understand what your Java program is really doing.

7.1 Overview

Jinsight brings together a range of techniques that let you explore many aspects of your program:

- ▶ Visualization

Visualizations let you understand object usage and garbage collection, and the sequence of activity in each thread, all from an object-oriented perspective.

- ▶ Patterns

Pattern visualizations extract the essential structure in repetitive calling sequences and complex data structures, thus letting you analyze large amounts of information in a concise form.

- ▶ Information exploration

Using Jinsight, you may specify filtering criteria to focus your study, or drill down from one view to another to explore details. You can create your own units that precisely match features you are studying, and then use them as an additional dimension in many of the views.

- ▶ Measurement

You can study measurements of execution activity or memory summarized at any level of detail, along call paths, and along two dimensions simultaneously.

- ▶ Memory leak diagnosis

Special features are provided to help you diagnose memory leaks.

Jinsight provides instrumentation (profiler agent for Java 2) for making trace data, and visualizes the trace data on a 100% Pure Java visualizer.

On z/OS you will create a trace using Jinsight instrumentation (controlled by a servlet), then you will analyze this trace on a workstation after having downloaded it with FTP.

7.2 Jinsight 2.1

In the following sections, we discuss downloading, installing, hosting, and deploying Jinsight.

7.2.1 Installation of the IBM JRE for Windows 1.3

The Jinsight visualizer needs IBM SDK1.3 for Windows, which can be downloaded from the following site:

<http://www7b.boulder.ibm.com/wsdd/wspvtdevkit-info.html>

Under IBM Developer Kit for Windows(R), Release 1.3.0, the needed files are the Installer and Runtime Environment installable packages. These two executable files will expand in the install.exe executable file for installing IBM Java Runtime. By default, the JRE is installed into C:\Program Files\IBM\Java13.

7.2.2 Download of Jinsight

Jinsight is located at the following site:

<http://www.alphaworks.ibm.com/tech/jinsight>

You should download the jinsight2.1-os390-java2.zip file.

7.2.3 Installation of Jinsight

After unzipping jinsight2.1-os390-java2.zip to a PC directory, we choose to install it directly under c:\. As a result, Jinsight was installed under the c:\jinsight2.1 directory.

To finish the setup of the Jinsight visualizer, in the file C:\jinsight2.1\jinsight.bat we modified the value of JAVA2_HOME variable to point to our jre. The modified line was:

```
set          JAVA2_HOME=C:\Progra~1\IBM\Java13\jre
```

The installation instructions were located under the following URL:

<file:///C:/jinsight2.1/docs/jump.htm#Installing>

At the time of writing, there were no instructions describing how to install Jinsight inside IBM WebSphere Application Server for z/OS and OS/390 Version 4.

We went to the directory C:\jinsight2.1, where we found the code for OS/390 in the file jinsight2.1-os390-java2.tar. We uploaded this file in binary mode to OS/390 in the directory /usr/lpp/jinsight, and ran tar to unwind it under OS/390 USS:

```
cd /usr/lpp/jinsight
tar -xvf /usr/lpp/jinsight/jinsight2.1-os390-java2.tar
```

7.2.4 Preparing a J2EE server to host Jinsight

To install Jinsight into a J2EE server, we needed to modify the environment file of the J2EE server. Using the SMUI, we had to do the following:

1. Append the Jinsight directory /usr/lpp/jinsight/jinsight2.1 to the LIBPATH environment variable.
2. Add the environment variable JVM_EXTRA_OPTIONS with the value -XrunjinsightPA.
3. Add (or modify) the variable JAVA_COMPILER to no. (Keep in mind that this will disable the Java just-in-time compiler, so in your case, do not keep it if the server goes to production.)
4. Add the environment variable JINSIGHT_TRACEFILE_NAME with the path and file name of the file in which we want to record traces. We also had to ensure that the path to that file is writable by the instrumented server.

Under the SMUI, we had to check the “Debugger Allowed” field.

We also added a new Service line in your httpd.conf pointing to the URI we planned to define under AAT (Context Root files) for the Jinsight Web application.

7.2.5 Packaging Jinsight code to be deployed into a J2EE server

Before running the Jinsight servlet, it must be placed inside a war file.

The jar file containing the servlet code needs to be downloaded back to the workstation. In your case, use your favorite FTP client to download /usr/lpp/jinsight/jinsight2.1/jintrace.jar in binary mode.

After a copy of C:\jinsight2.1\jintrace.htm has been made, modify it in order to be able to access it using any URI:

```
window.document.location = "http://" +
document.traceControlForm.serverName.value +
"/servlet/jintrace?command=" + command;
```

Those lines need to be changed to the following:

```
window.document.location = "jintrace?command=" + command;
```

The INPUT field declaration must also be deleted because it will not be used.

We then used WebSphere Studio to package the Jinsight application in a war file; see Figure 7-1.

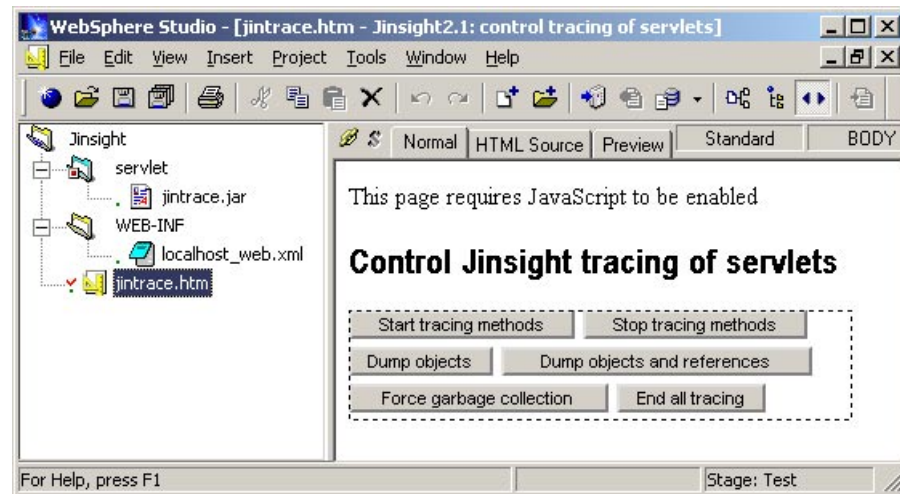


Figure 7-1 Jinsight project under WebSphere Studio

We filled the XML descriptor file with the servlet definition and URL mapping, as shown in Figure 7-2.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE web-app (View Source for full doctype...)>
- <web-app>
  <display-name>Jinsight2.1-WebApplication</display-name>
  <welcome-file>jintrace.htm</welcome-file>
- <servlet>
  <servlet-name>jintrace</servlet-name>
  <description>THE Jinsight servlet</description>
  <servlet-
    class>com.ibm.jinsight.tracing.TraceControlServlet</servlet-class>
  </servlet>
- <servlet-mapping>
  <servlet-name>jintrace</servlet-name>
  <url-pattern>/jintrace</url-pattern>
  </servlet-mapping>
</web-app>
```

Figure 7-2 Jinsight Web application XML descriptor

7.2.6 Installing Jinsight in the J2EE Server

After creating the war file, we deployed under the AAT and installed into a J2EE server. When we started the J2EE server, we checked that the J2EE server region SYSOUT contained the Jinsight start messages:

```
Jinsight2.1, build 20010629
Jinsight Profiler is Licensed Materials - Property of IBM
Copyright (c) IBM Corp. 2000
IBM is a Trademark of International Business Machines
*****
Authors:  Jeaha Yang      Fereydoun Maali
*****
  OS/390 : use CTRL-V for tracing options
*****
Number of segments used will be  32
Size of a segment used will be  524288
Total buffer size used will be  16777216
```

We then started tracing from the menu at:

```
http://ourhost/ourContextRoot/jintrace.htm
```

We ran the JSP sample named simpleJSP from the sample Web application and stopped the trace. After the trace was downloaded in binary mode to the Jinsight visualizer, we checked the trace content; see Figure 7-3 on page 141.

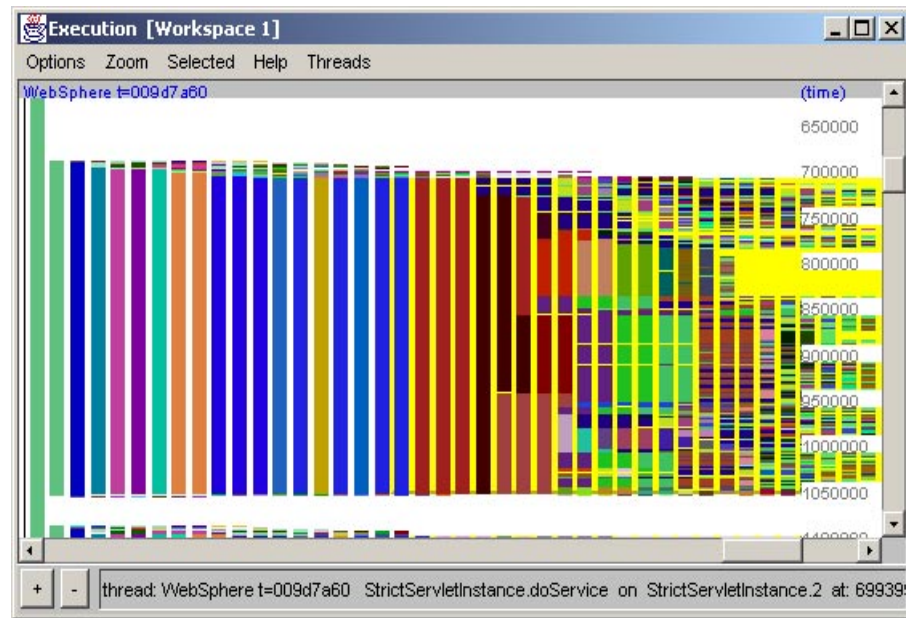


Figure 7-3 Jinsight visualizer, application execution view

7.2.7 Additional information on Jinsight

Following is the Web site for the Jinsight project:

<http://www.research.ibm.com/jinsight>

There is also a w3 IBM Internal Web site for Jinsight.

Extensive documentation on the usage of the visualizer is contained in the download packages and is also available online at:

<http://www.research.ibm.com/jinsight/docs/refman/refman.htm#Visualizing>

Although Jinsight is an unsupported tool, the Jinsight project members will try to answer questions. They appreciate your feedback to jinsight@us.ibm.com.



Object Level Trace

Object Level Trace allows you to trace and debug distributed application code execution from a workstation (for example, from the developer's Integrated Desktop Environment). Object Level Trace generates visual images of application execution flow, provides application performance analysis capabilities, enables method calls level debugging, and manages source level debugging of distributed applications.

This chapter describes how we installed the Object Level Trace facility and used it to trace the Trade2 application running in IBM WebSphere Application Server for z/OS and OS/390.

8.1 Overview

WebSphere's runtime is OLT-enabled and sends debugging information to an IP-based server which is installed at the developer's workstation. OLT uses either trace or debug mode. In this chapter we describe the OLT setup to work together with VAJ. The WebSphere Studio Application Developer IDE also comes with a Remote Debugger that follows the same basic concepts. This setup is briefly outlined in Appendix 8.4, "WSAD Remote Debugger quick install" on page 148.

In trace mode, OLT displays the following:

- ▶ Graphical representation of the interactions between Java clients, EJBs, JSPs, and Servlets.
- ▶ Method calls flow
- ▶ Time spent in a method

Figure 8-1 shows the Object Level Trace Viewer.

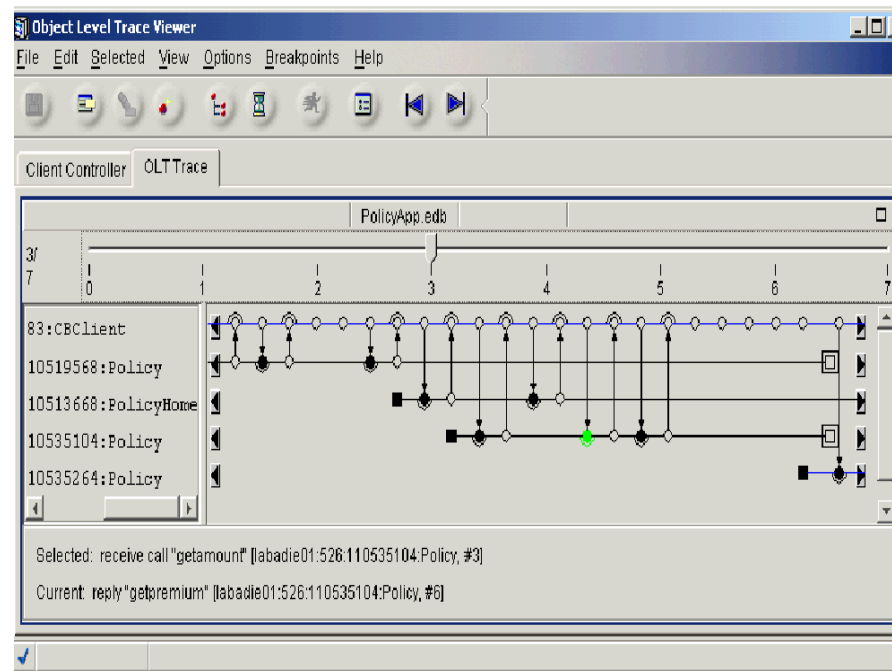


Figure 8-1 OLT Trace Viewer

The debugging mode simultaneously displays both source and executed bytecode, with the opportunity to use debugging facilities such as breakpoints setting, step-by-step mode and so on.

8.2 OLT Implementation

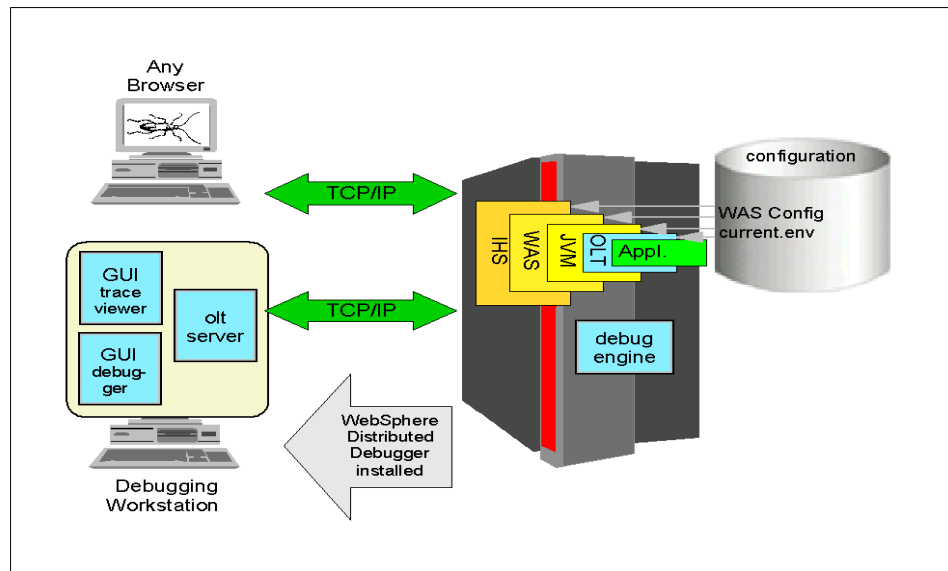


Figure 8-2 OLT and WebSphere Application Server

The OLT implementation consists of the following elements:

1. The OLT runtime runs in the same JVM as the traced code. It gets the data from the runtime environment and sends it to the OLT server. In a WebSphere environment, the OLT runtime is installed within IBM WebSphere Application Server for z/OS and OS/390. The code of the OLT runtime is provided with WebSphere Application Server's runtime, so no additional installation has to be done. The Java code needs to be compiled with the debug option (-g) in order to be debuggable by the OLT client.
2. The OLT viewer is used to display the data gathered by the OLT client. It binds to a TCP/IP port to receive the tracing information. It can also save current trace data to be inspected later. It has to be installed on a workstation.
3. The Debugger Daemon, which is started automatically by OLT when you register a client executing in Trace and Debug or Debug Only mode. This daemon launches a debug session when you choose to step into a debuggable method.

The OLT viewer supports different execution modes:

- ▶ No trace and debug: used when not tracing or debugging the application.
- ▶ Trace only (default): as the application runs, OLT monitors events and creates a trace. Trace only mode has a lesser impact on memory and performance

than any of the debugging modes. Once a trace is analyzed, method breakpoints can be set on selected events, then the execution mode can be changed to Trace and debug mode and the application rerun.

- ▶ **Debug only:** this mode enables you to debug applications without producing a trace. In this mode, the Debugger steps into every debuggable method without first prompting. While debugging, no other users can trace or debug using the same application server.
- ▶ **Trace and debug:** this mode provides access to both the trace facility and the Distributed Debugger. It also provides the greatest control over the debugging process. If Options > Step-by-step debug mode is selected in the OLT viewer (it is selected by default), OLT stops the application at each debuggable method. At that point, the programmer can choose to step into or over the debuggable code.

8.3 OLT workstation installation

Attention: We tested the OLT with WebSphere Application Server 4.01 ESP, and this pre-GA code did not contain all of the OLT runtime code. As a result, we got a temporary fix to be used with this early version and the GA code settings may be slightly different; refer to the WebSphere Application Server 401, Chapter 18 "Collecting data about J2EE application activity" of SA22-7836. Note that the instructions provided in Distributed Debugger help menus are not updated to WebSphere Application Server z/OS V4.

The OLT code is delivered with WebSphere Studio and VisualAge for Java with the Distributed Debugger. When one of these product installation processes is launched, a panel asks the installer if he wants to install the product or the Distributed Debugger, as shown in Figure 8-3 on page 147:

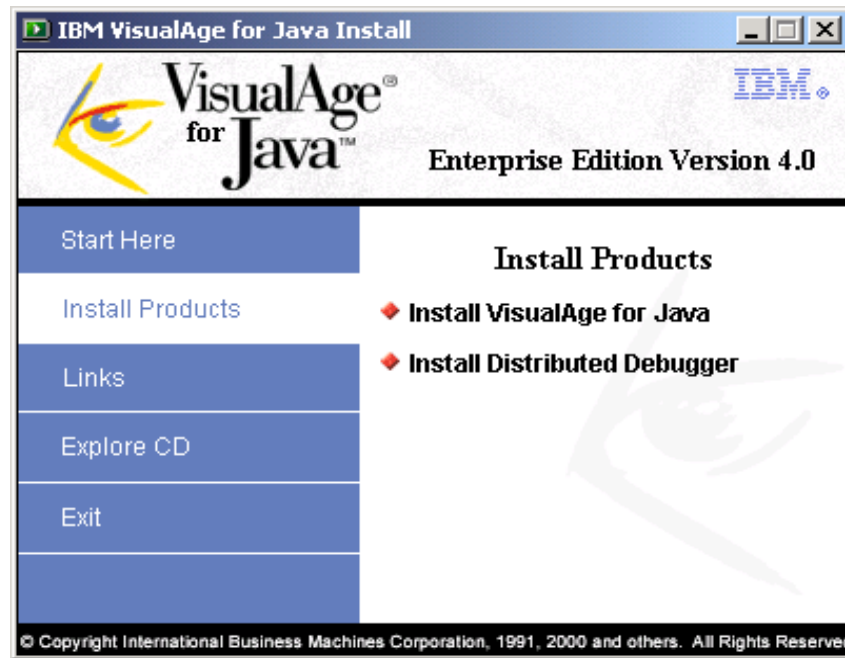


Figure 8-3 The VisualAge OLT installation screen

You should then see the OLT installation menu shown in Figure 8-4:

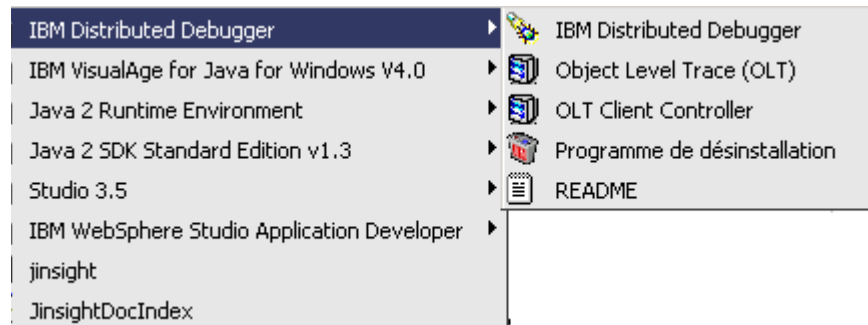


Figure 8-4 The OLT installation menu

If you do not get to this menu, just rerun WebSphere Studio or VisualAge for Java installation and choose **Distributed Debugger** in the first installation process panel.

8.4 WSAD Remote Debugger quick install

WebSphere Studio Application Developer IDE also comes with a Remote Debugger. In this section we list and describe the basic implementation steps:

1. In the application server region, add the environment property using SMEUI to enable the debugger by specifying:

```
JVM_DEBUG_PORT=nnnn
```

where nnnn is any available TCP/IP port.

2. Start the Server Region.

You'll notice the following message in the server log, which indicates that the debugger is enabled and listening on the specified debug port:

```
jdwp port is set to <nnnn>
```

3. Start the WebSphere Studio Application Developer debug session. From the WebSphere Studio Application Developer Debug Perspective:

- Select **Debug Remote Java Application**.
- Enter the IP address of the Application Server.
- Enter the jdwp port number mentioned above.
- Select **Finish** and verify that the debug process starts up in the WSAD debug pane.

4. Debugging procedure

Make sure you have the appropriate Java source code in WebSphere Studio Application Developer corresponding to the WebSphere Application Server for the z/OS application being debugged.

- Enter breakpoints in the source code.
- Invoke the method to debug on WebSphere Application Server z/OS to hit the breakpoints and wait for the WebSphere Studio Application Developer debugger window to open the source code and highlight the line with the breakpoints.
- Use the WebSphere Application Developer debug window controls to navigate the source code and inspect the runtime information.

8.5 WebSphere for z/OS installation

There is no additional code to install with WebSphere for z/OS 4.01 to run the OLT; the agent OLT runtime has been merged in the WebSphere runtime.

You need to verify that you have at least the IBM SDK 1.3 SR6 (the date returned by 'java -fullversion' should be at least July 2001).



Introscope

Introscope from Wily Technology is a tool dedicated to monitoring Java applications.

It can monitor and manage:

- ▶ Java application performance
- ▶ JVM usage and performance
- ▶ WebSphere Application Server performance

9.1 Introduction

Introscope from Wily Technology is a Java Web Application Management solution. It monitors the performance of any Java Web application in real time production and QA environments, isolating bottlenecks quickly by monitoring every component and its interactions from within the Java application and the Web Application Server.

Using Introscope, you can monitor and manage:

- ▶ Java application performance
 - Overall application performance and response times
 - JSP and Servlet frontend response times and transaction rates
 - Enterprise Java Bean (EJB) usage and performance
 - Individual method-level performance statistics
- ▶ JVM usage and performance
 - Internal activity: Memory usage, garbage collection, CPU stats, thread counts
 - External connections: Socket traffic and File I/O
 - J2EE services: JDBC, JNDI, RMI, or any standard J2EE method
- ▶ WebSphere Application Server performance
 - WebSphere Admin Server and Managed Server within the JVM
 - Internal component performance
 - J2EE services provided by WebSphere

Additional modules are available from Wily to manage Java Web Application connections to IBM Application Framework for e-business components such as the CICS Transaction Gateway and the IBM MQSeries Connector, as well as Java application interaction with SQL databases such as DB2.

Introscope currently supports any combination of JVM and JDK on all IBM WebSphere platforms including AIX, z/OS, OS/390, AS/400, NT, and Solaris. For more information on IBM WebSphere Application Server Advanced Edition, refer to Wily Technology at:

<http://www.wilytech.com/>

9.2 Introscope architecture

Introscope consists of three main components:

- ▶ Introscope Agent

The Agent runs in the same JVM as the managed Java application. It collects and summarizes performance data on the application, and sends the data to the Introscope Enterprise Manager for analysis. WebSphere for z/OS and OS/390 Version 3.5 and 4 APAR PQ53605 includes the Wily AutoProbe interface, which makes performance data collection automatic for applications running on those platforms.

Note: At the time of writing, APAR PQ53605 was still under development; check the PTF availability status.

► Introscope Enterprise Manager

The Enterprise Manager is the management server for Introscope. It receives performance data from managed Java applications via Introscope Agents, performs analysis and calculations, sends processed data to the Introscope Workstation for display, and optionally stores performance data to a relational database for historical trend and capacity planning analysis.

► Introscope Workstation

The Workstation is the graphical user interface for viewing performance data gathered by the Enterprise Manager. It consists of two main windows, the Introscope Explorer and the Console:

- The Explorer shows all performance data collected by the Introscope Agents, and allows users to create management modules to manage this data.
- The Console allows users to create customized dashboards to display the performance data.

Figure 9-1 on page 154 illustrates the flow of Introscope data.

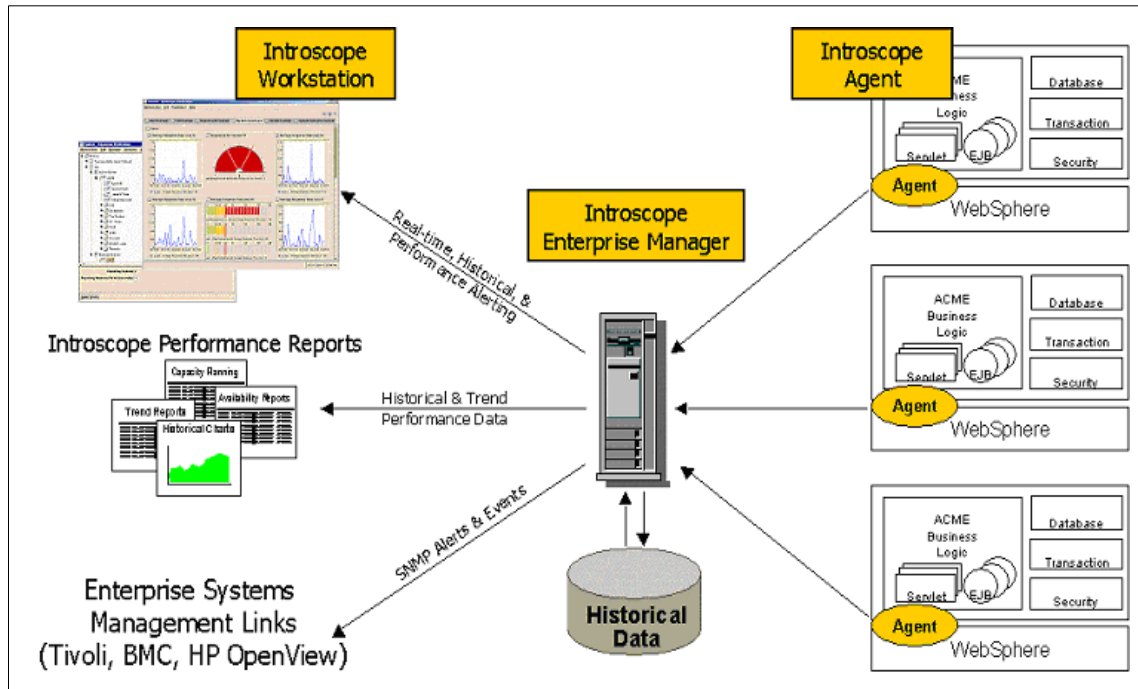


Figure 9-1 Introscope data flow

The Introscope Workstation is intended to run on any NT workstation platform. The Enterprise Manager and managed application (Agent) can be placed on either a z/OS or a distributed platform server.

9.2.1 Enabling an application for Introscope management

Introscope's Agent technology works by monitoring the execution of Java bytecode. Introscope relies on probes that work with the JVM and report on the Java application as it is running.

Introscope provides two methods for enabling the monitoring of applications, neither of which requires source code changes to the application or platform: Introscope ProbeBuilder, and Introscope AutoProbe:

- Option 1 - Introscope ProbeBuilder

The ProbeBuilder is an Introscope utility that creates a copy of Java bytecode (.class or .jar), scans the bytecode for methods/classes described in ProbeBuilder Directives files, and adds Probes to the methods/classes. These Probes allow the Introscope Agent to collect performance data from the Java application.

The “probed” bytecode will be used to run the application. The Introscope ProbeBuilder can be executed using the ProbeBuilder GUI Wizard, or a shell command under OMVS.

However, the manual ProbeBuilder process has two shortcomings: It cannot monitor JSPs which are compiled on the fly by application servers, and it needs to be invoked every time a new application or code is installed. Introscope AutoProbe does not have these shortcomings.

► Option 2 - Introscope AutoProbe

Introscope's AutoProbe feature avoids these two issues by integrating the ProbeBuilder with an application server's class loader. This allows ProbeBuilder to dynamically insert probes into the Java application bytecode as it is loaded by the class loader. Introscope's AutoProbe integration with WebSphere 3.5.2 and higher makes it easy to monitor any WebSphere Java applications.

9.3 Using Introscope 2

We describe the use of Introscope 2 for monitoring the Trade2 application. (Introscope 3, with AutoProbe functionality, is now generally available. You can find installation instructions in 9.4, “Using Introscope 3.0” on page 163.

9.3.1 Download

An evaluation copy of Introscope can be requested from the Wily Web site at:

<http://www.wilytech.com>

Two files will be needed:

- installintroscope.exe: (the Windows installation file)
- introscope2.6.1os390install.tar: (the z/OS installation file)

We also downloaded the *Wily Technology Introscope User Guide* from:

http://www.wilytech.com/support_productsupport.html

And we received a temporary license file from Wily.

9.3.2 Workstation installation

We chose to install both the workstation interface and Enterprise Manager on a Windows server. The Enterprise Manager was configured to keep historical data in an UDB/NT 7.1 database.

Note: Although we decided to install the Enterprise Manager under Windows, it may be more efficient to host this component within z/OS in order to minimize network load.

We executed the `installintroscope.exe`, and then chose the option Complete installation.

Because we wanted to keep data in a database in order to have access to historical views, we had to do some additional configuration, as follows:

- ▶ We created a new database named INTROSC with the UDB Control Center.
- ▶ We created Introscope tables using the sample DDL from `C:\Introscope\examples\database\db2.sql`. We had to:
 - Put each SQL command on one line.
 - Suppress the semi-colon (;) character at the end of each line.
 - Insert, as the first command: `CONNECT TO INTROSC`.

Note: You can find the modified version of this file in the Additional Materials of this book as: `db2Introscope.sql`.

- ▶ We configured `C:\Introscope\config\IntroscopeEnterpriseManager` properties directives for historical recording:
 - `introscope.enterprisemanager.db.driver=COM.ibm.db2.jdbc.app.DB2Driver`
 - `introscope.enterprisemanager.db.url=jdbc:db2:INTROSC`
 - `introscope.enterprisemanager.db.username=myUser`
 - `introscope.enterprisemanager.db.password=myPassword`

Finally, we copied the license file in the `C:\Introscope\license` subdirectory.

9.3.3 z/OS Agent installation

For this installation, we proceeded as follows:

1. Created an HFS file (50 Megabytes).
2. Created mount point `/usr/lpp/Introscope`.
3. FTP'ed, in binary mode, `introscope2.6.1os390install.tar`.
4. Exploded `introscope2.6.1os390install.tar` on z/OS as follows:

```
cd /usr/lpp/Introscope
tar -xvf introscope2.6.1os390install.tar
```
5. We exploded the needed file:

```
tar -xvf introscope2.6.1os390.tar.Z
```

9.3.4 J2EE server configuration

We configured the J2EE where we intended to put the monitored application as follows:

1. Created a J2EE server using SMUI and added the following in the CLASSPATH environment variable:
`/usr/lpp/Introscope/introscope2.6.1/lib/Agent.jar`
2. Created a jvm.properties file in the J2EE server configuration directory:

```
touch jvm.properties
chown CBSYMSR1:CBCFG1 jvm.properties
edit jvm.properties
```

We used the Introscope Installation guide to customize this file. At a minimum, the following variables need to be defined:

```
com.wily.introscope.agentProfile=/WebSphere390/CB390/controlinfo/envfile/OPPLEX/BBINTRA1/Agent.profile
```

```
com.wily.introscope.agent.logfile=/WebSphere390/CB390/logs/introscope.bb intra1.log
```

3. Copied the Introscope Configuration file into the J2EE server configuration directory:

```
cp /usr/lpp/Introscope/introscope2.6.1/examples/ExampleAgent.profile
/WebSphere390/CB390/controlinfo/envfile/OPPLEX/BBINTRA1
cd /WebSphere390/CB390/controlinfo/envfile/OPPLEX/BBINTRA1
chown CBSYMSR1:CBCFG1 Agent.properties
```

4. Customized these server-specific Agent.properties. We modified the following properties:

```
com.wily.introscope.agent.verbose=true
introscope.agent.enterprisemanager.transport.tcp.host=IP address of the
workstation hosting the Enterprise Manager
introscope.agent.enterprisemanager.transport.tcp.port=5001
introscope.agent.defaultProcessName=BBINTRA1
```

9.3.5 DB2 classes instrumentation

In order to monitor jdbc calls, the jdbc driver jar file has to be processed by ProbeBuilder.

We did this under OS/390 Unix System Services.

We created a little shell command file (pbjar) in order to more easily call the ProbeBuilder class:

```
CP='-classpath /usr/lpp/Introscope/introscope2.6.1/lib/ProbeBuilder.jar'
CL='com.wily.introscope.api.IntroscopeProbeBuilder'
DR='-directives
/usr/lpp/Introscope/introscope2.6.1/config/systempbd/websphere.pb1'
java $CP $CL $DR -origjar $1 -destjar $2 -verbose
```

To run ProbeBuilder, we just called this shell file using the input file as the first parameter and the output file as the second parameter.

During our tests, we found that the zip file name for the DB2 7.1 jdbc driver needed to stay db2j2classes.zip; using another name provides jdbc errors about not finding DSNJDBC_JDBCProfile.ser file. So we duplicated the whole /usr/lpp/db2/db2710/classes directory into /usr/lpp/db2/db2710/classesi.

We then ran ProbeBuilder on the jdbc driver sited in this directory:

```
mv /usr/lpp/db2/db2710/classesi/db2j2classes.zip
/usr/lpp/db2/db2710/classesi/db2j2classes.original.zip
```

```
pbjar /usr/lpp/db2/db2710/classesi/db2j2classes.original.zip
/usr/lpp/db2/db2710/classesi/db2j2classes.zip
```

```
10/2/01 10:53:28 AM EDT [I] [ProbeBuilder] [Introscope ProbeBuilder]
Introscope ProbeBuilder Release 2.6.1 (Build 617)
10/2/01 10:53:28 AM EDT [I] [ProbeBuilder] [Introscope ProbeBuilder] Using
Java VM version 1.3.0 from IBM Corporation
10/2/01 10:53:28 AM EDT [I] [ProbeBuilder] [Introscope ProbeBuilder] Wily
Technology (tm) Introscope (tm) Version 2
10/2/01 10:53:28 AM EDT [I] [ProbeBuilder] [Introscope ProbeBuilder]
Copyright (c) 1998 - 2001 Wily Technology, Inc. All Rights Rese
rved.
Processing /usr/lpp/db2/db2710/classesi/db2j2classes.zip
Log file created at
/usr/lpp/db2/db2710/classesi/db2j2classes.zip.probebuilder.log
Processing directives file:
/usr/lpp/Introscope/introscope2.6.1/config/systempbd/websphere.pb1
Converting db2j2classes.zip
Processing com/ibm/db2/jcc/DB2BaseDataSource.class
Processing com/ibm/db2/jcc/DB2CharacterEncodings.class
...
Processing COM/ibm/db2os390/sqlj/util/DB2SQLJProperties.class
Processing COM/ibm/db2os390/sqlj/util/DB2SQLJTrace.class
Finished db2j2classes.zip
Results Summary:
    Total Classes Encountered:    718
    Total Classes Modified:      56
```



```
Total Classes Skipped:      0
Total Number Of Archives:   1
Total Number Of Files Copied: 45
Total Number Of Files Skipped: 0
Total Number Of Errors:     0
Processing successful.
Please refer to the ProbeBuilder log file at
/usr/lpp/db2/db2710/classesi/db2j2classes.zip.probebuilder.log for more
details.
Adding probes finished.
Total time adding probes: 303 seconds
```

To have JDBC calls monitored, we then just had to switch the CLASSPATH variable in the J2EE server to point to
`/usr/lpp/db2/db2710/classesi/db2j2classes.zip`

9.3.6 Trade2 instrumentation

In order to monitor the J2EE application inside Introscope, we ran ProbeBuilder on the already deployed .ear file.

Unlike for the DB2 jdbc classes, we found it easier to use the ProbeBuilder installed on the workstation. Its advantage is that it is a graphical user interface application.

Although Agent.jar was in the J2EE server CLASSPATH, we then found out that we needed to include it in the .ear file at two other places:

- ▶ In the root of the .ear to be found by EJBs
- ▶ In the WEB-INF/lib path of the Web application war file to be found by servlets

We had to include the Agent.jar *after* ProbeBuilder because it needs to be out of ProbeBuilder instrumentation.

To include the Agent.jar, we proceeded with the following steps:

- ▶ Download Agent.jar to your PC in subdirectory C:\temp\WEB-INF\lib.
- ▶ Start AAT.
- ▶ Import the instrumented .ear.
- ▶ Select the new imported application.
- ▶ Right-click **Modify**.
- ▶ Select the Files folder and press **Import**.
- ▶ Go to select the Agent.jar and press **OK** without modifying the proposed relative path after import.

- ▶ Save the modification.
- ▶ Select the Web application.
- ▶ Right-click **Modify**.
- ▶ Select the Files folder and press **Import**.
- ▶ Go to select the Agent.jar.
- ▶ Select the WEB-INF/lib/Agent.jar relative path after import.

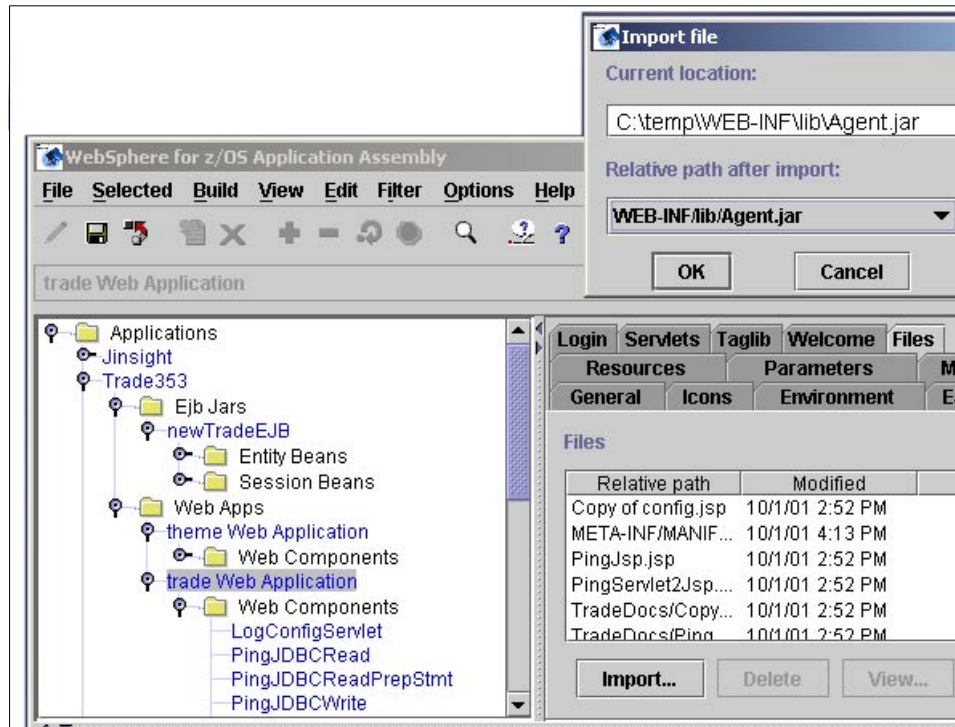


Figure 9-2 Importing Agent.jar in a war file

- ▶ Press **OK**.
- ▶ Save the modification.
- ▶ Redeploy the .ear file.

At the end of this process, we had to obtain an .ear file ready to be installed on the host with Introscope monitoring inside. We simply installed it under the SMUI as we did for any regular .ear file.

9.3.7 Checking the installation

Before starting the J2EE server, we checked whether the Enterprise Manager program was already started.

Then, after starting the J2EE server, we browsed the file defined previously in `jvm.properties` with the `com.wily.introscope.agent.logfile` variable. It looked like this:

```
Introscope Agent Release 2.6.1 (Build 617)
Using Java VM version 1.3.0 from IBM Corporation
Trying to load profile based on system property
"com.wily.introscope.agentProfile"
Trying to load profile (based on system property) from:
/WebSphere390/CB390/controlinfo/envfile/OPPLEX/BBINTRA1/Agent.profile
Loaded profile (based on system property) from:
/WebSphere390/CB390/controlinfo/envfile/OPPLEX/BBINTRA1/Agent.profile
Looking for Platform Monitor for "z/OS"...
No Platform Monitor found.
Introscope Agent startup complete....
The Agent will connect to the Introscope Enterprise Manager using: TCP
target: 9.12.6.141:5001 read buffer size: 4096 write buffer size: 4096
Using Agent protocol revision 15.0
Connected Agent BBINTRA1 to the Introscope Enterprise Manager at
9.12.6.141:5001.
```

At the same time, the Enterprise Manager log (in the workstation) appeared:

```
10/1/01 6:40:47 PM EDT [I] [Manager] [Agent] Connected to
Agent"wtsc59oe|BBINTRA1|175"
```

So we started the workstation interface, and after playing around with the Trade2 application, we used the workstation interface explorer to look at response times.

We then obtained the screen shot shown in Figure 9-3 on page 162.

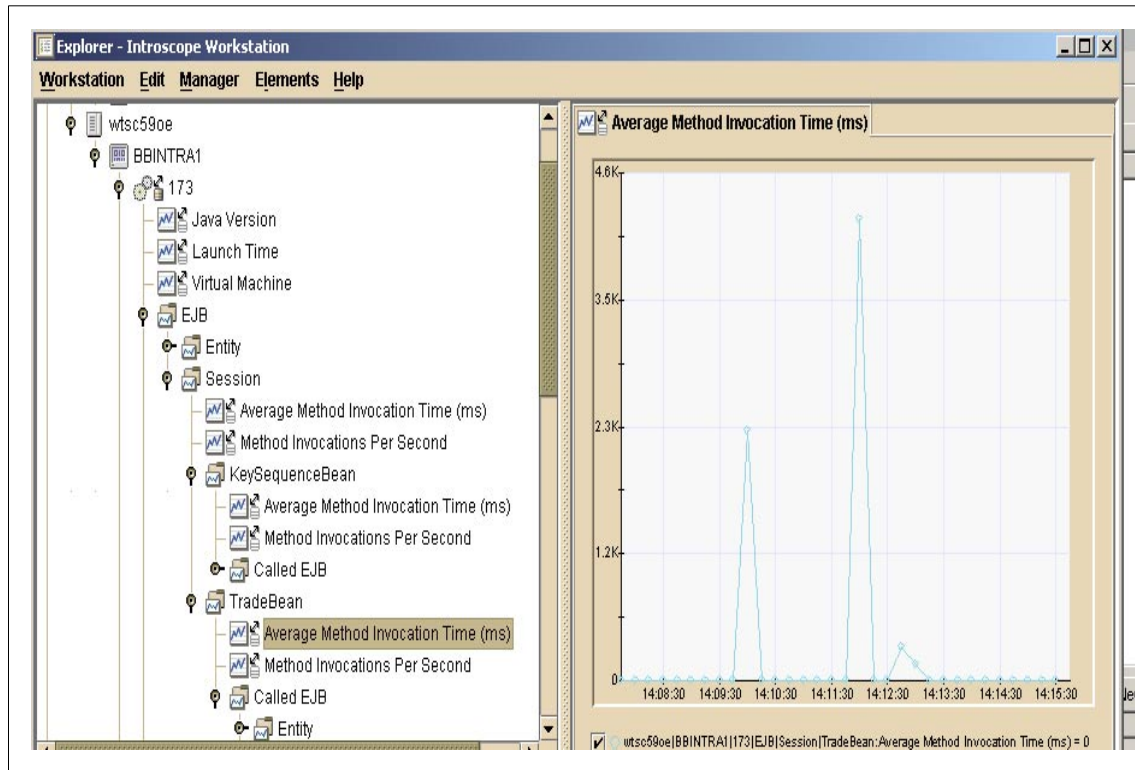


Figure 9-3 Sample Introscope display

9.3.8 Disable Introscope monitoring

If you do not want your application to be monitored, you will have to install the .ear file before ProbeBuilder instrumentation.

9.4 Using Introscope 3.0

9.4.1 Downloading and Installing Introscope

The installer to be used for OS/390 is named *introscope3.0os390install.tar*.

Refer to the Introscope User Guide for instructions on downloading and installing.

After you have uncompressed the AutoProbe installer into the appropriate directory, you need to configure AutoProbe to work with z/OS.AutoProbe on WebSphere 4.0.1 Enterprise Edition for OS/390.

WebSphere configuration

1. Introscope integration with WebSphere 4.01 for OS/390 requires the following PTF, supplied by IBM: PTF UQ61610. This PTF should be included with the WebSphere Application Server 4.0.1 OS/390 release.
2. Create a J2EE Server using SMUI.
3. Add <autoprobe.dir>/Agent.jar and <autoprobe.dir>/ProbeBuilder.jar to the CLASSPATH variable of this new server instance's current.env file.
4. Edit (creating if necessary) the jvm.properties file in the J2EE server configuration directory. You should ensure that this file has root permissions.

```
touch jvm.properties
chown CBSYMSR1:CB CFG1 jvm.properties
edit jvm.properties
```

5. At least the following properties need to be defined:

```
com.wily.introscope.agentProfile=<autoprobe.dir>/ AutoProbe.properties
com.wily.introscope.agent.logfile=<your.path>/your.file
com.wily.introscope.probebuilder.websphere.homeDirectory=<autopr obe.dir>
com.ibm.websphere.classloader.plugin=com.wily.introscope.api.web
sphere.WASAutoProbe
```

6. Customize the AutoProbe.properties file in your AutoProbe installation to point the Introscope Agent to the correct location of the Introscope Enterprise Manager. Edit the following properties in <autoprobe.dir>/AutoProbe.properties. You will need to do this for each managed instance of WebSphere:

```
com.wily.introscope.agent.verbose=true
introscope.agent.enterprisemanager.transport.tcp.host=<IP address of Enterprise
Manager's host>
introscope.agent.enterprisemanager.transport.tcp.port=<Enterpris e Manager
port>
introscope.agent.defaultProcessName=<Name of this managed WebSphere instance>
```

Database configuration for OS/390

Wily provides scripts for either creating a new version 3.0 database for DB2 on OS/390, or upgrading a version 2.6.x database to version 3.0. The scripts are found in the <Introscope>/examples/database directory.

- ▶ To create a new database, use the script db2_os390.sql.
- ▶ To upgrade your previous database, use the script v3_upgrade_db2_os390.sql

Complete instructions for creating a new database schema or upgrading your current database can be found in the Introscope User Guide Version 3.0.

Running Introscope Enterprise Manager as a z/OS batch job

To facilitate use of Introscope in the z/OS environment, it is desirable to run the product in a manner familiar to z/OS operations and development staff—either as a z/OS batch job or as a started task. You can also run Introscope Enterprise Manager as a shell process if desired. The following list contains a brief description of each section:

- ▶ Overview briefly describes the OS/390 environment and the BPXBATCH utility. This section describes software prerequisites, security, authorization, and address space size requirements.
- ▶ Installation describes the tasks necessary to customize the sample JCL and other files in preparation for running the Introscope Enterprise Manager as a batch job on the z/OS platform.
- ▶ Running Introscope Enterprise Manager as a OS/390 Batch Job describes the activities required to run the Introscope Enterprise Manager as a batch job or started task on the z/OS platform.
- ▶ DB2 Considerations discusses some of the most important considerations for deploying the Introscope Enterprise Manager database on DB2 for OS/390.

9.4.2 Overview

Prerequisites

The infrastructure required to run the Introscope Enterprise Manager as an z/OS batch job or started task is straightforward and easily realized. The z/OS utility program, BPXBATCH, enables users to invoke UNIX System Services to run shell commands, shell scripts and executable files in MVS batch. Specifically, the Introscope Enterprise Manager can be started using the BPXBATCH utility.

Software Prerequisites

The specific software prerequisites to run Java (and therefore the Introscope Enterprise Manager) on the OS/390 can be found in the Java for OS/390 Program Directory GI10-0614.

Security and Authorization

Authorization is determined by the RACF profile associated with a user's USERID. To successfully start the Introscope Enterprise Manager, a user must have authority to access the following resources:

- ▶ UNIX System Services as specified by the OMVS parameter in the RACF profile
- ▶ Introscope Enterprise Manager DB2 Tables
- ▶ EXEC PGM= BPXBATCH
- ▶ EXEC WILYPROC

It is important to understand how the user is identified in various contexts:

Running Introscope as a Batch Job or Started Task **Batch Job**

When a user submits a batch job through the TSO/E or ISPF interface, the user's TSO UserID and password are propagated to the batch job. Provided the submitted Batch JOB CARD does not explicitly contain a User and Password, the batch job acquires the same access authority as the TSO user. If User and Password are coded on the Batch JOB CARD, they override the UserID and Password of the TSO/E user.

Started Task

If the Introscope Enterprise Manager is initiated as a started task, it runs under the authorization of the STC user ID. A Site-specific algorithm may be used to relate the name of the started PROC to a particular UserID. You must make sure that the algorithm produces a UserID with the correct authorization to run the Introscope Enterprise Manager.

Tip: If the OMVS address space is not large enough, an out of memory condition will occur when an attempt is made to run the Introscope Enterprise Manager. The OMVS ASSIZEMAX(address-space-size) parameter in the RACF profile for the owner of the Introscope Enterprise Manager job or started task must be large enough to execute the Introscope Enterprise Manager. The maximum value for this parameter is 2GB

.DB2

To access JDBC and SQLJ from UNIX Systems Services, you must set the following environment variables:

- ▶ STEPLIB must include the SDSNEXIT and SDSNLOAD data sets
- ▶ LIBPATH and LD_LIBRARY_PATH must include the DLL libraries for the JDBC and SQLJ drivers

In order to run the Introscope Enterprise Manager as a OS/390 job, the *runem.sh* shell script must contain the following statements where DSNXXX denotes the High Level Qualifier of your DB2 libraries:

```
export STEPLIB=DSNXXX.SDSNEXIT:DSNXXX.SDSNLOAD
export LIBPATH=/usr/lpp/db2/db2510/lib:$LIBPATH
export LD_LIBRARY_PATH=/usr/lpp/db2/db2510/lib:$LD_LIBRARY_PATH
```

In addition, the CLASSPATH must include the JDBC and SQLJ driver classes. The classpath in the *runem.sh* shell script should include the following file:

```
/usr/lpp/dsn610/db2/db2610/classes/db2sqljclasses.zip
```

The character string “610” in dsn610 and db2610 refers to the Version and Release of DB2 being used. If another version of DB2 is being used, these directories must be changed

JDBC

JDBC for OS/390 is based on the DB2 Call Level Interface (CLI). You must ensure that the DB2 CLI has been set up as follows:

1. The DB2 CLI application plan (the default name is DSNACLI) must be bound before you can use the DB2 CLI. A sample bind job can be found in DSNXXX.SDSNSAMP(DSNTIJCL) where DSNXXX denotes the high level qualifier of your DB2 installation.
2. The DB2 CLI application plan must be public. Execute the following command from SPUFI or from a batch job:

```
GRANT EXECUTE ON PLAN DSNACLI TO PUBLIC
```


3. Set DSNAOINI to point to the CLI initialization file. The CLI initialization file provides information about DB2 subsystem names and additional configuration parameters. You can place the file either in an HFS file or in an MVS data set using one of the following statements:

```
export DSNAOINI=/usr/lpp/db2/db2510/dsnaoini
export DSNAOINI=DSNxxx.CLIINI
export DSNAOINI=DSNxxx.CLIINI(CONF1)
```

4. Make sure that the DB2 subsystem referenced in the CLI initialization file references the DB2 subsystem that you wish to connect to. DB2 subsystem names usually take the form DB2n where n is a unique qualifier.

DASD

The Introscope Enterprise Manager DB2 database constitutes the only significant DASD requirement.

9.4.3 Installation

Overview of Installation Tasks

Before the Introscope Enterprise Manager can run as an OS/390 batch job, the following files must be copied into the OS/390 environment and customized to ensure compatibility with site-specific requirements.

- ▶ IntroscopeEnterpriseManager.properties – the Introscope Enterprise Manager properties file, which specifies information pertinent to the operation of the Introscope Enterprise Manager
- ▶ The following three files must be downloaded from the Wily support page at <http://www.wilytech.com/support.html>.
 - runem.sh – the shell script used to launch the Introscope Enterprise Manager process
 - WILYPROC – the OS/390 PROC used to execute OS/390 BPXBATCH program that launches the shell script
 - DB2_OS390.ddl – the DDL to define the required DB2 tables, indexes and views and to populate selected Introscope Enterprise Manager tables.

Copying and Customizing Introscope Enterprise Manager Files

EM.PRO and runem.sh are UNIX files that must be copied into the HFS. These files will be used by UNIX System Services and the Introscope Enterprise Manager respectively. Required changes to these files should be made using a UNIX-based editor such as vi. Do not use the ISPF editor because it does not properly handle insertion of characters that increase line length.

WILYPROC and DB2SQL are OS/390 files that will be used by the JES2 component of z/OS and by DB2 for OS/390. Changes to these files should be made using the ISPF editor.

Copying the Introscope Enterprise Manager UNIX files to HFS Files

Copy the IntroscopeEnterpriseManager.properties and runem.sh files into the Introscope installation directory, as shown in the following examples, where “xxxx” is the UID assigned to Wily Technology.

```
/u/xxxx/introscope3.0/runem.sh  
/u/xxxx/introscope3.0/config/ IntroscopeEnterpriseManager.properties
```

Customizing the Introscope Enterprise Manager Properties File

1. Open the u/xxxx/introscope3.0/config/IntroscopeEnterpriseManager.properties file.
2. Find the “# Database Settings” heading. Under this heading, you should find the following statements:

```
introscope.enterprisemanager.db.driver=COM.ibm.db2os390.sqlj.jdbc  
c.DB2SQLJDriver  
introscope.enterprisemanager.db.url=jdbc:db2os390:  
introscope.enterprisemanager.db.username=myUser  
introscope.enterprisemanager.db.password=myPassword
```

Verify that the driver is the correct one for this connection to DB2.

Note: You do not have to change the username and password because they will not be used by DB2 for authorization.

3. Find the “# Interactive Mode” heading and make sure the value for the following property is set to false .
4. Comment out the following statements that redirect statements for STDERR and STDOUT (the files are allocated in the OS/390 JCL):

```
introscope.enterprisemanager.disableInteractiveMode=false  
  
#introscope.enterprisemanager.debug.output.redirect.path=/u/wily/  
Introscope Enterprise Manager_out.txt  
#introscope.enterprisemanager.debug.error.redirect.path=/u/wily/ Introscope  
Enterprise Manager_err.txt
```

Customizing runem.sh – the UNIX Shell Script

Verify the version and release of DB2. The DB2 files in the runem.sh shell script reference DB2 V6.1.

Note: If you are connecting to a DB2 subsystem other than version 6.1, you must change all instances of “610” to the appropriate character string for the version being used.

The following is a list of settings in the runem.sh script that must be changed:

```
In the CLASSPATH, /usr/lpp/dsn610/db2/db2610/classes/ db2sqljclasses.zip
export STEPLIB=DSN610.SDSNEXIT:DSN610.SDSNLOAD
#export LIBPATH=/usr/lpp/dsn610/db2/db2610/lib:$LIBPATH
#export LD_LIBRARY_PATH=/usr/lpp/dsn610/db2/db2610/lib:$LD_LIBRARY_PATH
```

Note: If LIBPATH and LD_LIBRARY_PATH have been commented out, remove the #.

Copying Introscope Enterprise Manager files to a Partitioned Dataset

Copy WILYPROC and DB2SQL into the PDS

1. Using ISPF, create a partition dataset named userid.JCL.CNTL, (where userid is the userid assigned to WILY) with fixed block (FBA) 80 byte records.
2. Copy the IntroscopeEnterpriseManager.properties and runem.sh files into the Introscope installation directory, as shown in the following examples, where userid is the userid assigned to Wily and userid.JCL.CNTL is the name of the previously created PDS.

```
put WILYPROC 'userid.JCL.CNTL(WILYPROC)'
put DB2SQL 'userid.JCL.CNTL(DB2SQL)'
```

3. Do not try to copy the shell script, runem.sh , or the properties file, IntroscopeEnterpriseManager.properties , into this PDS. These files have lines that are longer than 80 bytes. An attempt to copy them will either fail or lines will be truncated.

Customizing the OS/390 PROC That Invokes BPXBATCH

1. Edit WILYPROC using the ISPF editor:
 - On the ISPF command line, type CAPS OFF and press Enter.
 - On the ISPF command line, type NUM OFF and press Enter.
2. The following statement is coded so that the batch job and the UNIX shell script will end within 20 seconds:

```
// PARM='SH nohup /u/wily/introscope3.0/runem.sh & sleep 20
```

In the unlikely event that the OS/390 is too busy to complete the initiation of the Introscope Enterprise Manager within 20 seconds, the Introscope Enterprise Manager will not be started. If you encounter this problem, try increasing sleep time.

3. Modify the STDERR and STDOUT file names contained in the following statements to satisfy installation naming conventions:

```
//STDOUT DD PATH='/u/wily/testout',
```

```
//STDERR DD PATH='/u/wily/testerr',
```

Copy WILYPROC into an active PROCLIB

Using ISPF option 3, copy the PROC to xxxx.PROCLIB where xxxx represents an site-dependent high level qualifier. If WILYPROC is not an acceptable name in your installation, rename the PROC accordingly.

Customizing the DDL

Customization of the sample DDL is described in the Introscope User Guide.

9.4.4 Running the Introscope Enterprise Manager

Executing as a Started Task

You may run the Introscope Enterprise Manager as an MVS started task (STC). A started task is a familiar, operator-friendly environment that allows the Introscope Enterprise Manager to be started, monitored and cancelled from the MVS console. In this case, the Introscope Enterprise Manager runs under the authorization of the STC user ID, thereby allowing assignment of specific authorities to the started task.

To run the Introscope Enterprise Manager as a started task, type the following command on the MVS console or on the command line of SDSF:

```
S WILYPROC
```

Running as a Batch Job

The Introscope Enterprise Manager can also run as a batch job. To run the Introscope Enterprise Manager as a batch job, submit the following JCL:

```
//jobname JOB installation jobcard parameters,NOTIFY=&SYSUID  
ISCOPE EXEC WILYPROC  
/*
```

You must execute the member name of the PROC stored in an active PROCLIB. If you were required to change the name WILYPROC, you must use the new name on the EXEC statement.

The OS/390 batch job and the OMVS Shell will end shortly after the Introscope Enterprise Manager processes have started. If you submitted the batch JCL from your ISPF session and your JOB card contains NOTIFY=&SYSUID, your ISPF session will receive notification when your batch jobs ends.

Verifying That the Introscope Enterprise Manager Is Running

The most accurate way to determine that the Introscope Enterprise Manager has started correctly is to view the STDOUT file. To view STDOUT, you must telnet into the OS/390 and view the STDOUT file defined in the WILYPROC. The last line of the file should indicate that the Introscope Enterprise Manager has started.

You can also determine that the Introscope Enterprise Manager has started from SDSF by following one of the procedures below:

- ▶ If you started the Introscope Enterprise Manager as a started task, on the DA screen you should see at least two active started tasks whose owner is the owner associated with the Introscope Enterprise Manager PROC.
- ▶ If you started the Introscope Enterprise Manager by submitting a batch job from your TSO session, look for two active tasks with your TSO USERID as the owner.

You may also see the job that invoked the WILYPROC as well as the OMVS shell script. These two entries will disappear from the list, once the Introscope Enterprise Manager has started.

The owner of your TSO session is also your TSO/E USERID. Do not mistake your TSO/E session for one of the Introscope Enterprise Manager tasks.

Starting the Introscope Enterprise Manager automatically

To can start the Introscope Enterprise Manager automatically when JES2 starts:

1. Place the following statement in JES2PARM:

```
$VS,'start WILYPROC
```

2. You must type the member name of the PROC stored in an active PROCLIB. If you were required to change the name of WILYPROC, you must use the new name in the z/OS *START* command.

You can also run the Introscope Enterprise Manager as a batch job using whatever job scheduling package you use to manage your batch workload.

9.4.5 DB2 Considerations

DB2 for OS/390 provides many installation parameters and customization options that are designed to leverage the reliability, availability and serviceability of the OS/390 platform as well as the special features of DB2. Users of DB2 for OS/390 generally augment these parameters and options with site-specific

standards, naming conventions, operator procedures and OEM products. For this reason, the assumptions and suggestions contained in the DDL obtained from Wily Technology should be thoroughly and carefully reviewed prior to installation of the Introscope Enterprise Manager.

Assumptions

Table Usage

The database contains seven tables and one view. There is a single unique index defined on each table. The following tables are small, non-volatile, code tables:

```
WT_VERSIONLIST
WT_INDEX
WT_PROPERTY
WT_AGENT
WT_RESOURCE
WT_METRIC_NAME
WT_METADATA
WT_TMP_Q_1
WT_TMP_Q_2
WT_TMP_Q_3
WT_TMP_Q_4
WT_TMP_Q_5
```

Currently, the DDL for these tables takes all defaults except for “IN database”.

WT_METRIC

The WT_Metric table is the largest table in the Introscope Enterprise Manager database and the most heavily used. The implementation of the Introscope Enterprise Manager described below represents the anticipated upper bound for number of agents deployed and number of metrics collected. Consequently, this implementation represents the anticipated upper bound for the size of the WT_Metric table.

Environment

The Introscope Enterprise Manager is the primary means of accessing the WT_Metric table. Only rarely will SQL queries be executed against the WT_Metric table outside of the Introscope Enterprise Manager application.

Database Implementation Considerations

Sample database creation script is based on the assumptions listed above, particularly the assumptions related to the WT_Metric table. Review these assumptions carefully. If your environment will be substantially different, you should reevaluate associated database creation script parameters.

Customizing the DDL

Using the ISPF editor, customize the database creation script provided in DB2_OS390.ddl.

Customizing the tablespaces and the sample CREATE TABLE statements

When customizing the tablespaces, follow these rules:

- ▶ The WT_Metric table data should have its own segmented tablespace.
- ▶ The WT_Metric_U1 index should also have its own dedicated tablespace.
- ▶ The WT_Metric table and its index should be placed on high-speed devices. The physical path to the data should be on a different path from the physical path to the index.

These tablespaces may be defined as either DB2 managed or user managed.

Tables and indexes

All tables and indexes should be defined in the Wily database – not the default database. All tables should be assigned to a specific storage group or tablespace. All CREATE INDEX statements should contain a “USING” clause designating either a specific storage group or VCAT.



Part 4

Appendixes



Sample code and JCL

Here are the samples developed for this redbook:

- ▶ Trade2 test client
- ▶ DB2 tables for Trade2 on z/OS
- ▶ RMF and SMF samples

The test client source code

To give the operator a fast way to test drive your Web application after activating it, you supply a test client. The aim of this testing is to make sure that the application is running properly and accessible through the network. In 5.3.5, “Automatic application verification” on page 113 we discuss that in detail.

The following Java source code represents a runnable Java class that performs six http requests to Trade2 and evaluates the responses. Every response is checked for a keyword or a key term that appears in the response only if the application is working.

To place a single request and to look for the keyword, the method `CheckAnURL` is used. It returns `True` whenever the desired keyword was found in the response and no exception occurred.

The method of this runnable class is to accept, the base URL of the current Trade2 deployment as a parameter. It attaches different relative paths to this URL and calls `CheckAnURL` six times, each time supplying a different URL and a different keyword.

If all tests were successful, it reports this success to the `StandardOut`. It also reports any failure to `StandardOut`. The operator uses this information to decide about a fallback to a system status prior to the activation of the new Web application.

The following source code and the resulting class file can be downloaded from the supporting Web site.

Example: A-1

```
import java.net.*;
import java.io.*;

/**
 * This class is an example for a simple testclient which can be called
 * from a script to find out if an application is running.
 *
 * All tests and paths are hardcoded to keep the class as simple and as
 * intuitive as possible.
 */

public class Trade2TestClient {
    /**
     * This method calls a given URL and looks for a pattern String
     * within the response. If the pattern appears in at least one
     * line of the response it returns true. In the other case it
```

```

* returns false.
*
* @return boolean: true if the pattern was found, false else
* @param urlToCall java.lang.String: The URL which leads to the response
* @param patternToAppearInResponse java.lang.String: any string pattern which
is assumed to be in the response
*/

private static boolean checkAnURL(String urlToCall, String
patternToAppearInResponse) {

    boolean success = false; // pessimistic assumption of the result

    URLConnection urlConnection;
    InputStream istr;

    try {
        // open the connection and obtain the stream containing the response
        URL url = new URL(urlToCall);
        urlConnection = url.openConnection();
        istr = ((URLConnection) urlConnection).getInputStream();

        // read the response line by line, use ISO-8859 encoding for this
        BufferedReader reader = new BufferedReader(new
InputStreamReader(istr,"ISO-8859-1"));
        String line;

        while ((line = reader.readLine()) != null) {

            // look for the pattern in the current line
            if (line.indexOf(patternToAppearInResponse) >= 0) { // The pattern
is found
                success = true;
            }
        }

        } catch (Throwable t) { // if any exception is raised we consider the test
as failed
            success = false; // even if the pattern was found before
            t.printStackTrace();
        }

        return success;
    }
    /**
    * This method calls a number of urls to the trade2 application and
    * evaluates the responses in order to find out wether the application
    * is working or not.

```

```

*
* The url to the servlet directory in which Trade2 resides is supplied
* as the first a command line argument.
* If an http proxy server is used to test the application the address
* of this server is transferred as the second command line parameter
* while the port is supplied as the thrid command line parameter.
*
* If the tests are successful it returns [put in here] otherwise
* it returns.
*
*/

public static void main(String[] args) {

    String urlToCall;
    String patternToAppear;
    String httpProxyAddress = "";
    String httpProxyPort = "";
    String urlPrefix = args[0]; // url to the servlet directory
    int exitCode = 8; // 8 indicates an error

    // set a proxy if parameters for that have been supplied
    if (args.length > 2) { // the proxy parameters have been supplied
        httpProxyAddress = args[1]; // optional http proxy server address
        httpProxyPort = args[2]; // optional http proxy server port
        System.getProperties().put("proxySet", "true");
        System.getProperties().put("proxyHost", httpProxyAddress);
        System.getProperties().put("proxyPort", httpProxyPort);
    }

    // testcase 1
    // try logging in with an invalid username
    urlToCall = urlPrefix +
"TradeAppServlet?uid=invaliduid%3A1&passwd=xxx&action=login";
    patternToAppear = "Could not find"; // appears only when login could not
    proceed
    boolean testCase1 = checkAnURL(urlToCall,patternToAppear);

    // testcase 2
    // try logging in with a valid username
    urlToCall = urlPrefix +
"TradeAppServlet?uid=uid%3A1&passwd=xxx&action=login";
    patternToAppear = "Welcome"; // appears only on successful login
    boolean testCase2 = checkAnURL(urlToCall,patternToAppear);

    // testcase 3
    // write to database using a prepared statement
    urlToCall = urlPrefix + "PingJDBCWritePrepStmt";

```

```

        patternToAppear = "Update Information"; // appears only on successful
operation
        boolean testCase3 = checkAnURL(urlToCall,patternToAppear);

        // testcase 4
        // perform a calculation using a stateless SessionEJB
        urlToCall = urlPrefix + "PingServlet2Session";
        patternToAppear = "Investment Return Information"; // appears only on
successful operation
        boolean testCase4 = checkAnURL(urlToCall,patternToAppear);

        // testcase 5
        // get a row from the database through a cmp SessionEJB
        urlToCall = urlPrefix + "PingServlet2Entity";
        patternToAppear = "Quote Information"; // appears only on successful
operation
        boolean testCase5 = checkAnURL(urlToCall,patternToAppear);

        // testcase 6
        // tests the full servlet to Session EJB to Entity EJB path
        urlToCall = urlPrefix + "PingServlet2Session2Entity";
        patternToAppear = "Quote Information"; // appears only on successful
operation (same as in last case 5)
        boolean testCase6 = checkAnURL(urlToCall,patternToAppear);

        // check if all tests were successful
        boolean success = testCase1 & testCase2 & testCase3 & testCase4 & testCase5
& testCase6;

        // report the success
        if (success) {
            System.out.println("Success: All tests of the Trade2 test client
succeeded. The application is running.");
            exitCode = 0; // 0 indicates success

        } else {
            System.out.println("Error: The Trade2 test client could not verify that
the application is running.");
        }

        System.exit(exitCode);
    }
}

```

DB2 DDL job for Trade2 database

The sample job to create DB2 tables for our Trade2 application is shown in Example A-2.

Note: this job reflects the environment we used in our lab. You have to customize it to reflect your environment.

The DB2 definitions regarding ALIAS and prefixes follow the naming convention described in Appendix B, “J2EE server naming convention and customization” on page 193.

Example: A-2 DB2 sample job for Trade2

```
//DEFTRADE JOB (999,POK),'ANDRE',CLASS=A,REGION=OM,
//          MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//*****
/* This sample JCL can be used to create Trade2 tables
/* NOTE:
/* - This job have to be customized to reflect runtime environment
/* - The table aliases are necessary due to Trade2 design as it uses
/*   USERID/PW in servlet TradeAppServlet (initparm). We used ANDRE
/*   for this particular server BBPROD
/* - prefix used was BBPRODS which is the Server Region identity
/*
//*****
/* Step1 - Cleanup old Trade2 tables, databases, spaces, storgroups and
/*          aliases
//*****
//DROPTAB EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DB7A)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA71) -
      LIB('DB7AU.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
DROP Database TRADEDDB;
commit;
DROP alias ANDRE.TradeQuoteBean;
DROP alias ANDRE.TradeHoldingBean;
DROP alias ANDRE.TradeProfileBean;
DROP alias ANDRE.TradeRegistryBean;
DROP alias ANDRE.KeysEntityBean;
DROP ALIAS ANDRE.TradeAccountBean;
/*
//*****
```



```

/* Step2 - Delete linear datasets used in old tables and recreate them
/*****
//DEFVSAM EXEC PGM=IDCAMS,COND=(8,LT)
//SYSPRINT DD SYSOUT=*
DELETE DB7AU.DSNDBC.TRADEDB.ACCTBSPC.I0001.A001
DELETE DB7AU.DSNDBC.TRADEDB.ACCOUNDX.I0001.A001
DELETE DB7AU.DSNDBC.TRADEDB.HOLDBSPC.I0001.A001
DELETE DB7AU.DSNDBC.TRADEDB.HOLDINDX.I0001.A001
DELETE DB7AU.DSNDBC.TRADEDB.PROFBSPC.I0001.A001
DELETE DB7AU.DSNDBC.TRADEDB.PROFINDX.I0001.A001
DELETE DB7AU.DSNDBC.TRADEDB.QUOTBSPC.I0001.A001
DELETE DB7AU.DSNDBC.TRADEDB.QUOTENDX.I0001.A001
DELETE DB7AU.DSNDBC.TRADEDB.REGBSPC.I0001.A001
DELETE DB7AU.DSNDBC.TRADEDB.REGISNDX.I0001.A001
DELETE DB7AU.DSNDBC.TRADEDB.KEYSSPC.I0001.A001
DELETE DB7AU.DSNDBC.TRADEDB.KEYSNDXB.I0001.A001
SET MAXCC = 0
DEFINE CLUSTER-
    (NAME(DB7AU.DSNDBC.TRADEDB.ACCTBSPC.I0001.A001)) -
    LINEAR -
    REUSE -
    VOLUMES(OP1DB3) -
    RECORDS(10000 2500) -
    SHAREOPTIONS(3 3) -
    DATA -
    (NAME(DB7AU.DSNDBC.TRADEDB.ACCTBSPC.I0001.A001))
DEFINE CLUSTER-
    (NAME(DB7AU.DSNDBC.TRADEDB.ACCOUNDX.I0001.A001)) -
    LINEAR -
    REUSE -
    VOLUMES(OP1DB3) -
    RECORDS(10000 2500) -
    SHAREOPTIONS(3 3) -
    DATA -
    (NAME(DB7AU.DSNDBC.TRADEDB.ACCOUNDX.I0001.A001))
DEFINE CLUSTER-
    (NAME(DB7AU.DSNDBC.TRADEDB.HOLDBSPC.I0001.A001)) -
    LINEAR -
    REUSE -
    VOLUMES(OP1DB3) -
    RECORDS(10000 2500) -
    SHAREOPTIONS(3 3) -
    DATA -
    (NAME(DB7AU.DSNDBC.TRADEDB.HOLDBSPC.I0001.A001))
DEFINE CLUSTER-
    (NAME(DB7AU.DSNDBC.TRADEDB.HOLDINDX.I0001.A001)) -
    LINEAR -
    REUSE -
    VOLUMES(OP1DB3) -

```

```

        RECORDS(10000 2500) -
        SHAREOPTIONS(3 3) ) -
DATA -
    (NAME(DB7AU.DSNDBD.TRADEDB.HOLDINDX.I0001.A001))
DEFINE CLUSTER-
    (NAME(DB7AU.DSNDBC.TRADEDB.PROFBSPC.I0001.A001) -
    LINEAR -
    REUSE -
    VOLUMES(OP1DB3) -
    RECORDS(10000 2500) -
    SHAREOPTIONS(3 3) ) -
DATA -
    (NAME(DB7AU.DSNDBD.TRADEDB.PROFBSPC.I0001.A001))
DEFINE CLUSTER-
    (NAME(DB7AU.DSNDBC.TRADEDB.PROFINDX.I0001.A001) -
    LINEAR -
    REUSE -
    VOLUMES(OP1DB3) -
    RECORDS(10000 2500) -
    SHAREOPTIONS(3 3) ) -
DATA -
    (NAME(DB7AU.DSNDBD.TRADEDB.PROFINDX.I0001.A001))
DEFINE CLUSTER-
    (NAME(DB7AU.DSNDBC.TRADEDB.QUOTBSPC.I0001.A001) -
    LINEAR -
    REUSE -
    VOLUMES(OP1DB3) -
    RECORDS(4000 2500) -
    SHAREOPTIONS(3 3) ) -
DATA -
    (NAME(DB7AU.DSNDBD.TRADEDB.QUOTBSPC.I0001.A001))
DEFINE CLUSTER-
    (NAME(DB7AU.DSNDBC.TRADEDB.QUOTENDX.I0001.A001) -
    LINEAR -
    REUSE -
    VOLUMES(OP1DB3) -
    RECORDS(4000 2500) -
    SHAREOPTIONS(3 3) ) -
DATA -
    (NAME(DB7AU.DSNDBD.TRADEDB.QUOTENDX.I0001.A001))
DEFINE CLUSTER-
    (NAME(DB7AU.DSNDBC.TRADEDB.REGBSPC.I0001.A001) -
    LINEAR -
    REUSE -
    VOLUMES(OP1DB3) -
    RECORDS(10000 2500) -
    SHAREOPTIONS(3 3) ) -
DATA -
    (NAME(DB7AU.DSNDBD.TRADEDB.REGBSPC.I0001.A001))

```

```

DEFINE CLUSTER-
  (NAME(DB7AU.DSNDBC.TRADEDB.REGISNDX.I0001.A001) -
  LINEAR -
  REUSE -
  VOLUMES(OP1DB3) -
  RECORDS(10000 2500) -
  SHAREOPTIONS(3 3) ) -
  DATA -
  (NAME(DB7AU.DSNDBD.TRADEDB.REGISNDX.I0001.A001))
DEFINE CLUSTER-
  (NAME(DB7AU.DSNDBC.TRADEDB.KEYSSPC.I0001.A001) -
  LINEAR -
  REUSE -
  VOLUMES(OP1DB3) -
  RECORDS(10000 2500) -
  SHAREOPTIONS(3 3) ) -
  DATA -
  (NAME(DB7AU.DSNDBD.TRADEDB.KEYSSPC.I0001.A001))
DEFINE CLUSTER-
  (NAME(DB7AU.DSNDBC.TRADEDB.KEYSNDXB.I0001.A001) -
  LINEAR -
  REUSE -
  VOLUMES(OP1DB3) -
  RECORDS(10000 2500) -
  SHAREOPTIONS(3 3) ) -
  DATA -
  (NAME(DB7AU.DSNDBD.TRADEDB.KEYSNDXB.I0001.A001))
/*****
/* Step3 - Create Trade2 tables, databases, spaces, storgroups and
/* aliases
/*****
//DEFTAB EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(8,LT)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DB7A)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA71) -
LIB('DB7AU.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
CREATE database TRADEDB
BUFFERPOOL BPO;
CREATE TABLESPACE QuotBSp IN TRADEDB
BUFFERPOOL BPO
LOCKSIZE ROW
SEGSIZE 32
USING VCAT DB7AU
PCTFREE 15 ;

```

```

CREATE TABLESPACE AcctBSpc IN TRADEDDB
    BUFFERPOOL BP0
    LOCKSIZE ROW
    SEGSIZE 32
    USING VCAT DB7AU
    PCTFREE 15 ;

CREATE TABLESPACE HoIdBSpc IN TRADEDDB
    BUFFERPOOL BP0
    LOCKSIZE ROW
    SEGSIZE 32
    USING VCAT DB7AU
    PCTFREE 15 ;

CREATE TABLESPACE ProfBSpc IN TRADEDDB
    BUFFERPOOL BP0
    LOCKSIZE ROW
    SEGSIZE 32
    USING VCAT DB7AU
    PCTFREE 15 ;

CREATE TABLESPACE RegBSpc IN TRADEDDB
    BUFFERPOOL BP0
    LOCKSIZE ROW
    SEGSIZE 32
    USING VCAT DB7AU
    PCTFREE 15 ;

CREATE TABLESPACE KeysSpc IN TRADEDDB
    BUFFERPOOL BP0
    LOCKSIZE ROW
    SEGSIZE 32
    USING VCAT DB7AU
    PCTFREE 15 ;

CREATE TABLE BBPRODS.TradeQuoteBean (
    symbol      VARCHAR(251) NOT NULL,
    details     VARCHAR(251),
    price       FLOAT,
    PRIMARY KEY (symbol)) in TRADEDDB.QuotBSpc;
commit ;

CREATE Unique INDEX BBPRODS.QuoteNDXnTb1
    on BBPRODS.TradeQuoteBean (
    symbol ASC)
    Using
        VCAT DB7AU;
commit ;

```

```

CREATE TABLE BBPRODS.TradeAccountBean (
    userid VARCHAR(251) NOT NULL,
    balance FLOAT,
    transactions INTEGER,
    PRIMARY KEY (userid)) in TRADEDDB.AcctBSPc;
commit ;

CREATE Unique INDEX BBPRODS.AccouNDXeanTb1
    on BBPRODS.TradeAccountBean (
        userid ASC)
    Using
        VCAT DB7AU;
commit ;

CREATE TABLE BBPRODS.TradeHoldingBean (
    userid VARCHAR(128) NOT NULL,
    symbol VARCHAR(251),
    price FLOAT,
    quantity FLOAT,
    indx INTEGER NOT NULL,
    PRIMARY KEY (userid,indx)) in TRADEDDB.HoldBSPc;
commit ;

CREATE Unique INDEX BBPRODS.HoldiNDXeanTb1
    on BBPRODS.TradeHoldingBean (
        userid ASC, indx ASC)
    Using
        VCAT DB7AU;
commit ;

CREATE TABLE BBPRODS.TradeProfileBean (
    userid VARCHAR(251) NOT NULL,
    fullname VARCHAR(251),
    address VARCHAR(251),
    email VARCHAR(251),
    creditcard VARCHAR(251),
    PRIMARY KEY (userid)) in TRADEDDB.ProfBSPc;
commit ;

CREATE Unique INDEX BBPRODS.ProfiNDXeanTb1
    on BBPRODS.TradeProfileBean (
        userid ASC)
    Using
        VCAT DB7AU;
commit ;

CREATE TABLE BBPRODS.TradeRegistryBean (
    userid VARCHAR(251) NOT NULL,
    password VARCHAR(251),

```

```

        status INTEGER,
        PRIMARY KEY (userid)) in TRADEDB.RegBSpc;
commit ;

CREATE Unique INDEX BBPRODS.RegisNDXBeanTbl
    on BBPRODS.TradeRegistryBean (
        userid ASC)
    Using
        VCAT DB7AU;
commit ;

CREATE TABLE BBPRODS.KeysEntityBean
(name VARCHAR(251) NOT NULL,
 value INTEGER,
 PRIMARY KEY (name)) in TRADEDB.KeysSpc;
commit ;

CREATE Unique INDEX BBPRODS.KeysNDXBeanTbl
    on BBPRODS.KeysEntityBean (
        name ASC)
    Using
        VCAT DB7AU;
commit ;

insert into BBPRODS.KeysEntityBean( name, value)
VALUES ('holdingKey', 0);

GRANT DELETE, INSERT, SELECT, UPDATE
    ON TABLE BBPRODS.TradeQuoteBean
    TO PUBLIC AT ALL LOCATIONS;
commit ;

GRANT DELETE, INSERT, SELECT, UPDATE
    ON TABLE BBPRODS.TradeAccountBean
    TO PUBLIC AT ALL LOCATIONS;
commit ;

GRANT DELETE, INSERT, SELECT, UPDATE ON TABLE BBPRODS.TradeHoldingBean
    TO PUBLIC AT ALL LOCATIONS;
commit ;

GRANT DELETE, INSERT, SELECT, UPDATE ON TABLE BBPRODS.TradeProfileBean
    TO PUBLIC AT ALL LOCATIONS;
commit ;

GRANT DELETE, INSERT, SELECT, UPDATE ON TABLE BBPRODS.TradeRegistryBean
    TO PUBLIC AT ALL LOCATIONS;
commit ;

```

```

GRANT DELETE, INSERT, SELECT, UPDATE ON TABLE BBPRODS.KeysEntityBean
    TO PUBLIC AT ALL LOCATIONS;
commit ;

CREATE alias ANDRE.TradeQuoteBean for BBPRODS.TradeQuoteBean;
CREATE alias ANDRE.TradeHoldingBean for BBPRODS.TradeHoldingBean;
CREATE alias ANDRE.TradeProfileBean for BBPRODS.TradeProfileBean;
CREATE alias ANDRE.TradeRegistryBean for BBPRODS.TradeRegistryBean;
CREATE alias ANDRE.KeysEntityBean for BBPRODS.KeysEntityBean;
CREATE ALIAS ANDRE.TradeAccountBean for BBPRODS.TradeAccountBean;
/*

```

RMF examples

SMF extract job

Example: A-3 SMF extract sample job

```

//RMFEXT JOB (999,P0K),'ANDRE',CLASS=A,REGION=0M,MSGCLASS=H,
//      NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//*****
/* THIS SAMPLE JCL CAN BE USED TO EXTRACT SMF RECORDS
/* NOTE:
/* - THIS JOB HAVE TO BE CUSTOMIZED TO REFLECT RUNING ENVIRONMENT
//*****
/* EXTRACT: EXTRACT SMF RECORDS
//*****
//EXTRACT EXEC PGM=IFASMFD
//INSMFA DD DSN=SYS1.SC59.MANA,DISP=SHR
//INSMFB DD DSN=SYS1.SC59.MANB,DISP=SHR
//SMFDATA DD DSN=ANDRE.LSM059.SMFTARN,
//      DCB=(RECFM=VBS,LRECL=32760),
//      SPACE=(CYL,(150,100)),
//      UNIT=SYSALLDA,
//      DISP=(NEW,CATLG)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
          INDD(INSMFA,OPTIONS(DUMP))
          INDD(INSMFB,OPTIONS(DUMP))
OUTDD(SMFDATA,TYPE(0:255))
      DATE(2001291,2001292)
/*

```

RMF report job

Example: A-4 Sample RMF report generation job

```
//RMFREP JOB (999,POK),'ANDRE',CLASS=A,REGION=0M,MSGCLASS=H,
// NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//*****
//* THIS SAMPLE JCL CAN BE USED TO CREATE RMF REPORTS
//* NOTE:
//* - THIS JOB HAVE TO BE CUSTOMIZED TO REFLECT RUNING ENVIRONMENT
//*****
//* RMFSORT: SORT SMF RECORDS
//*****
//RMFSORT EXEC PGM=SORT,REGION=0M
//SORTIN DD DISP=SHR,DSN=ANDRE.LSM059.SMFTRAN
//SORTOUT DD DISP=(NEW,PASS),
// DSN=&&SORTOUT,
// UNIT=SYSALLDA,
// SPACE=(CYL,(100,100)),
// DCB=*.RMFSORT.SORTIN
//SORTWK01 DD DISP=(NEW,DELETE),
// DSN=&&WK1,
// UNIT=SYSALLDA,
// SPACE=(CYL,(100,250))
//SORTWK02 DD DISP=(NEW,DELETE),
// DSN=&&WK2,
// UNIT=SYSALLDA,
// SPACE=(CYL,(100,250))
//SORTWK03 DD DISP=(NEW,DELETE),
// DSN=&&WK3,
// UNIT=SYSALLDA,
// SPACE=(CYL,(100,250))
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
SORT FIELDS=(11,4,CH,A,7,4,CH,A),EQUALS
MODS E15=(ERBPPSRT,500),E35=(ERBPPSRT,500)
//*****
//* RMFPOST1: REPORTS HTTP
//*****
//RMFPOST1 EXEC PGM=ERBRMFPP
//MFPINPUT DD DSN=*.RMFSORT.SORTOUT,DISP=(OLD,PASS)
//SYSIN DD *
RTOD(0000,2400)
STOD(0000,2400)
SUMMARY (INT)
REPORTS (HTTP)
SYSOUT(T)
//*****
```




```
//* RMFPOST1: REPORTS WLM WAS* SERVICE CLASSES
//*****
//RMFPOST2 EXEC PGM=ERBRMFPP
//MFPINPUT DD DSN=*.RMFSORT.SORTOUT,DISP=(OLD,PASS)
//SYSIN DD *
RTOD(0000,2400)
STOD(0000,2400)
SUMMARY (INT)
SYSRPTS (WLMGL(SCLASS(WAS*)))
SYSOUT(T)
//*
```

Sample partial RMF report

Example: A-5 Sample RMF report for WAS* service classes

REPORT BY: POLICY=DAYTIME		WORKLOAD=WEBAPP		SERVICE CLASS=WASPROD			
				CRITICAL		=NONE	
				DESCRIPTION		=WEBAPP Prod	
TRANSACTIONS		TRANS.-TIME	HHH.MM.SS.TTT	--DASD I/O--	---SERVICE----		
AVG	0.03	ACTUAL	163	SSCHRT	0.0	IOC	0
MPL	0.03	EXECUTION	163	RESP	0.0	CPU	22654
ENDED	72	QUEUED	0	CONN	0.0	MSO	0
END/S	0.17	R/S AFFINITY	0	DISC	0.0	SRB	0
#SWAPS	0	INELIGIBLE	0	Q+PEND	0.0	TOT	22654
EXCTD	0	CONVERSION	0	IOSQ	0.0	/SEC	55
AVG ENC	0.03	STD DEV	353				
REM ENC	0.00						
MS ENC	0.00						
REPORT BY: POLICY=DAYTIME		WORKLOAD=WEBAPP		SERVICE CLASS=WASTEST			
				CRITICAL		=NONE	
				DESCRIPTION		=WEBAPP Test	
TRANSACTIONS		TRANS.-TIME	HHH.MM.SS.TTT	--DASD I/O--	---SERVICE----		
AVG	0.00	ACTUAL	5	SSCHRT	0.0	IOC	0
MPL	0.00	EXECUTION	5	RESP	0.0	CPU	15377



J2EE server naming convention and customization

When we started to run our tests, we needed to have some common naming conventions for our J2EE servers. There are a lot of RACF definitions, so having a naming convention allowed us to use a REXX exec to put these definitions in place and ease the creation of system tasks in PROCLIB.

This appendix describes the conventions we shared and the REXX exec to update the RACF database.

J2EE Server relative names

Each J2EE server name was defined by a group of alphanumeric characters which appear as XXXX in the following explanations. To help manage servers on a system having both a test and a production server, the first letter can be used to specify the type of server (T for test, V for validation, P for production).

Table 9-1 J2EE naming conventions

Item	Name
Server name	BBXXXX
WLM application environment name	BBXXXX
Server instance name	BBXXXXSN where - S is a letter representing the number of the system in the sysplex - N is the number of the instance on a specific system For example, the second instance of a server on the second image of a sysplex will be BBXXXXB2.
UserId for Control Region	BBXXXXC
STC for Control Region	BBXXXXC
Group ID for Control Region	BBXXXXG
UserId for Server Region	BBXXXXS
STC for Server Region	BBXXXXS
Group ID for Server Region	BBXXXXR
Default remote userid	BBXXXXI
Default local userid	BBXXXXD
Group ID name for default userids	BBXXXXP

RACFJ2EE REXX

“We used the following REXX program to set up RACF for each of our J2EE servers. We created it reworking the BBO.SBBOEXEC(BBOCBRAC) EXEC in order to fit our installation and naming conventions. This REXX program configures only one instance of a J2EE server at a time so you need to run it once per J2EE server instance. To use it, read the comments in the heading. You need to modify the code:

- ▶ The appl_id to the 4-character project identifier .
- ▶ The appl_instance to the correct value.
- ▶ If needed, set the UID and GID values to something meeting your installation standards.

Example: B-1 RACFJ2EE REXX code

```
/* REXX ----- */
/*
/* Basic Assumptions -
/*
/* The proc name for the control region and server region is the
/* same as the userid associated with the region.
/*
/* Here is the naming scheme:
/*
/* CBserver name           = BBXXXX
/* - APPLENV name          = BBXXXX
/* CBserver instance name  = BBXXXXAY
/* - ORBSrvname default value = BBXXXXAY
/* Userid for control region = BBXXXXC
/* - PROC for control region = BBXXXXC
/* Group id for control region = BBXXXXG
/* Userid for server region = BBXXXXS
/* - PROC for server region = BBXXXXS
/* Group id for server region = BBXXXXR
/* Default remote userid    = BBXXXXI
/* Default local  userid    = BBXXXXD
/* Group id for default ids = BBXXXXP
/*
/* So basically, the application server is distinguished from all
/* other servers by characters 2-6 (i.e., 'XXXX'). The server
/* instance, one or more on each system, is distinguished by
/* the 8th character (i.e., 'Y')
/*
/* Here are the userids/UIDs, change as desired.
/* BBXXXXC 0 - do not change.
/* BBXXXXS 1100
/* BBXXXXD 1101
/*
```

```

/*      BBXXXXI 1102                                */
/*      HerE are the groups/GIDS,  change as desired. */
/*      BBXXXXG 1000                                */
/*      BBXXXXR 1001                                */
/*      BBXXXXP 1002                                */
/*                                                    */
/*                                                    */
/*      Customization:                               */
/*                                                    */
/*      Set the appl_id to the 4-character project identifier */
/*      the appl_instance to the correct value.           */
/*                                                    */
/*      If desired, set the UID and GID values to something meeting */
/*      your installation standards.                     */
/*                                                    */
/*      ----- */

```

trace r

```

default_logstream_name           = "WAS.ERROR.LOG"

appl_id                          = "PROD"
appl_instance                    = "1"

default_appl_CR_proc_name        = "BB" || appl_id || "C"
say default_appl_CR_proc_name
default_appl_CR_userid           = default_appl_CR_proc_name
default_appl_CR_group            = "BB" || appl_id || "G"
default_appl_CR_UID              = "2170"
default_appl_CR_GID              = "2270"

default_appl_SR_proc_name        = "BB" || appl_id || "S"
default_appl_SR_userid           = default_appl_SR_proc_name
default_appl_SR_group            = "BB" || appl_id || "R"
default_appl_SR_UID              = "2171"
default_appl_SR_GID              = "2271"

default_appl_generic_server_name = "BB" || appl_id
default_appl_server_instance_name = "BB" || appl_id || "A" || ,
                                   appl_instance

default_appl_unauth_local_userid = "BB" || appl_id || "D"
default_appl_unauth_local_uid    = "2172"
default_appl_unauth_remote_userid = "BB" || appl_id || "I"
default_appl_unauth_remote_uid    = "2173"
default_appl_unauth_group         = "BB" || appl_id || "P"
default_appl_unauth_GID           = "2272"

default_CB_admin_group           = "CBADMGP"

```

```

default_CB_CFG_group          = "CBCFG1"
default_CB_CR_group           = "CBCTL1"

```

ADDRESS TSO

```

/* ----- */
/*
/* Create RACF groups for the Application control and server
/* regions and for the default client ids.
/*
/* ----- */
if appl_id = 'XXXX' then do
    say 'Error... configure the appl_id value and re-try.'
    exit(-1)
end
if appl_instance = 'Y' then do
    say 'Error... configure the appl_instance value and re-try.'
    exit(-1)
end
if default_logstream_name = '' then do
    say 'Error... configure the default_logstream_name and re-try.'
    exit(-1)
end
say 'Adding groups for Application control and server regions'
"ADDGROUP " || default_appl_CR_group || ,
  " OMVS(GID(" || default_appl_CR_GID || "))"

"ADDGROUP " || default_appl_SR_group || ,
  " OMVS(GID(" || default_appl_SR_GID || "))"

"ADDGROUP " || default_appl_unauth_group || ,
  " OMVS(GID(" || default_appl_unauth_GID || "))"

/* ----- */
/*
/* Define the user ids for Application Control and server regions.
/* Connect these ids to default_CB_CFG_group so HFS can be read.
/*
/* ----- */

say 'Adding users for IVP Control and Server Regions'

"ADDUSER " || default_appl_CR_userid || ,
  " DFLTGRP(" || default_appl_CR_group || ,
  " ) OMVS(UID(" || default_appl_CR_UID || ,
  " ) HOME(/tmp) PROGRAM(/bin/sh)) NAME('CB390 " || appl_id || " CR') "

"ADDUSER " || default_appl_SR_userid || ,
  " DFLTGRP(" || default_appl_SR_group || ,

```

```

    ") OMVS(UID(" || default_appl_SR_UID || ,
    ") HOME(/tmp) PROGRAM(/bin/sh)) NAME('CB390 " || appl_id || " SR') "

say 'Connecting IVP CR and SR userids to the CB configuration group.'

"CONNECT " || default_appl_CR_userid || ,
  " group(" || default_CB_CFG_group || ")"

"CONNECT " || default_appl_CR_userid || ,
  " GROUP(" || default_CB_CR_group || ")"

"CONNECT " || default_appl_SR_userid || ,
  " group(" || default_CB_CFG_group || ")"

/* ----- */
/*
/* Define the default userids for local and remote non-
/* authenticated users
/*
/* ----- */

say 'Adding default users for installation'

"ADDUSER " || default_appl_unauth_local_userid || ,
  " DFLTGRP(" || default_appl_unauth_group || ,
  ") OMVS(UID(" || default_appl_unauth_local_UID || ,
  ") HOME(/tmp) PROGRAM(/bin/sh)) NAME('CB390 " || appl_id || ,
  " LOCAL USER') "

"ADDUSER " || default_appl_unauth_remote_userid || ,
  " DFLTGRP(" || default_appl_unauth_group || ,
  ") OMVS(UID(" || default_appl_unauth_remote_UID || ,
  ") HOME(/tmp) PROGRAM(/bin/sh)) NAME('CB390 " || appl_id || ,
  " REMOTE USER') "

/* ----- */
/*
/* Assigning userids to X.SERVER to control region
/*
/* ----- */

say 'Assigning userids to started tasks'

"RDEF STARTED " || default_appl_CR_proc_name || ,
  ".* STDATA(USER(" || default_appl_CR_userid || ,
  ") GROUP(" || default_appl_CR_group || ,
  ") TRACE(YES))"

"RDEF STARTED " || default_appl_SR_proc_name || ,

```



```

        ".* STDATA(USER(" || default_appl_SR_userid || ,
        ") GROUP(" || default_appl_SR_group || ,
        ") TRACE(YES))"

/* ----- */
/*
/* Allowing access to CBIND, SERVER and SOMDOBJs profiles
/*
/* ----- */

say 'Defining CBIND CB.BIND.generic_server with control access.'

"RDEFINE CBIND CB.BIND." || default_appl_generic_server_name || ,
  " UACC(READ)"

"Permit CB.BIND." || default_appl_generic_server_name || ,
  " CLASS(CBIND) ID(" || default_CB_CR_group || ,
  ") ACC(CONTROL)"

say 'Defining CBIND CB.generic_server with read access.'

"RDEFINE CBIND CB." || default_appl_generic_server_name || ,
  " UACC(READ)"

say 'Defining SOMDOBJs servername.*.* with alter access.'

"RDEFINE SOMDOBJs " || default_appl_generic_server_name || ,
  ".*.* " || ,
  " UACC(ALTER) "

say 'Defining SERVER CB.server_instance.generic_server w/ read access.'

"RDEFINE SERVER CB.*." || ,
  default_appl_generic_server_name || ,
  " UACC(NONE) "
"PERMIT CB.*." || ,
  default_appl_generic_server_name || ,
  " CLASS(SERVER) ID(" || default_appl_SR_userid || ,
  ") ACC(READ)"

/* ----- */
/*
/* Allowing CR and SR to write to the logstream
/*
/* ----- */

say 'Defining BPX.SERVER with update access.'

```

```

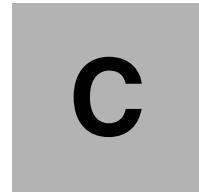
"PERMIT" default_logstream_name "CLASS(LOGSTRM)" || ,
      "ID(" || default_appl_CR_group || ") ACCESS(UPDATE)"
"PERMIT" default_logstream_name "CLASS(LOGSTRM)" || ,
      "ID(" || default_appl_SR_group || ") ACCESS(UPDATE)"

/* ----- */
/* Optional Step:                               */
/* Allowing our Web server to bind to the J2EE Server */
/* ----- */
/* "PE CB.BIND.BB" || appl_id,                  */
/*   " CLASS(CBIND) ",                          */
/*   " ID(WEBSRV) ",                            */
/*   " ACCESS(CONTROL)"                        */
/* ----- */
/* ----- */
/* Wrap it up.                                */
/* ----- */

say 'Refreshing the world'

"SETR RACLIST(CBIND)    GENERIC(CBIND)    REFRESH"
"SETR RACLIST(FACILITY) GENERIC(FACILITY) REFRESH"
"SETR RACLIST(SERVER)   GENERIC(SERVER)   REFRESH"
"SETR RACLIST(SOMDOBJ)  GENERIC(SOMDOBJ)  REFRESH"
"SETR RACLIST(STARTED)  GENERIC(STARTED)  REFRESH"

```



System Management scripting API

Here we provide sample documents and files used during our tests using the SMAPI referenced in the book.

JCL used to run SMAPI samples in batch

In order to automate the use of the SMAPI samples as much as possible, we used the BPXBATCH facility to submit these scripts in batch, instead of running them in the foreground using USS shell or Telnet.

Using BPXBATCH

Example: C-1 Using BPXBATCH

```
//APIBAT04 JOB (999,POK),'ANDRE',CLASS=A,REGION=OM,MSGCLASS=H,
//          NOTIFY=ANDRE,MSGLEVEL=(1,1)
//*
/*
/*****
/* Step1 - Run a the SMAPI in batch
/*****
//STEP1 EXEC PGM=BPXBATCH,
// PARM='SH /u/andre/SMAPI013'
/*-----
//STDIN DD DUMMY
//STDENV DD DUMMY
//STDOUT DD PATH='/tmp/&SYSUID..out',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//STDERR DD PATH='/tmp/&SYSUID..err',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
/*****
/* Step2 - Send results to SYSOUT
/*****
//STEP2 EXEC PGM=IKJEFT01
/*-----
//SYSTSPRT DD SYSOUT=*
//HFSOUT DD PATH='/tmp/&SYSUID..out'
//HFSERR DD PATH='/tmp/&SYSUID..err'
//STDOUTL DD SYSOUT=*,DCB=(RECFM=VB,LRECL=133,BLKSIZE=137)
//STDERRL DD SYSOUT=*,DCB=(RECFM=VB,LRECL=133,BLKSIZE=137)
//SYSPRINT DD SYSOUT=*
//SYSTSIN DD *
OCOPY INDD(HFSOUT) OUTDD(STDOUTL)
OCOPY INDD(HFSERR) OUTDD(STDERRL)
/*
```

Using TSO in batch

Example: C-2 Using TSO in batch

```
//APIBAT04 JOB (999,POK), 'ANDRE',CLASS=A,REGION=OM,MSGCLASS=H,
//          NOTIFY=ANDRE,MSGLEVEL=(1,1)
//*
//*
//*****/
/* Step1 - Run a the SMAPI in TSO batch */
//*****/
//APIBAT EXEC PGM=IKJEFT01,REGION=OM,DYNAMNBR=20
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//STDOUT DD PATH='/tmp/&SYSUID..out',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//STDERR DD PATH='/tmp/&SYSUID..err',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//SYSTSIN DD *
BPXBATCH SH /u/andre/SMAPI013
/*
//*****/
/* Step2 - Send results to SYSOUT
//*****/
//STEP2 EXEC PGM=IKJEFT01
//*-----
//SYSTSPRT DD SYSOUT=*
//HFSOUT DD PATH='/tmp/&SYSUID..out'
//HFSERR DD PATH='/tmp/&SYSUID..err'
//STDOUTL DD SYSOUT=*,DCB=(RECFM=VB,LRECL=133,BLKSIZE=137)
//STDERRL DD SYSOUT=*,DCB=(RECFM=VB,LRECL=133,BLKSIZE=137)
//SYSPRINT DD SYSOUT=*
//SYSTSIN DD *
OCOPY INDD(HFSOUT) OUTDD(STDOUTL)
OCOPY INDD(HFSERR) OUTDD(STDERRL)
/*
```

Fallback SMAPI script

Sample used in 5.3.3, “Scripted application fallback” on page 107.

Example: C-3 Fallback SMAPI script

```
/* REXX ----- */
/* ===== */
/* */
```

```

/* ITS0 2001 - Additional material */
/* RedBook: SG24-6521 */
/* */
/* FILENAME: /u/louis/SMAPI013 */
/* Sample of a backout script */
/* */
/* FUNCTION: */
/* 1) Creates a first conversation */
/* 2) Deletes an application in a specific server */
/* 3) Commits/Activates the first conversation */
/* 4) Creates a second conversation */
/* 5) Process ear file to restore a version of the application */
/* 6) Commits/Activates second conversation */
/* */
/* ===== */
/* */

call syscalls 'ON'
signal on error

say "Starting create/delete/activate/create/add/activate script"

/* ===== */
/* */
/* Customize: */
/* - ServerName */
/* - AppIDel */
/* - ResolvedEar */
/* ===== */
ConversationName1 = date() "time()" Delete"
ConversationName2 = date() "time()" ReInstall"
ServerName = "BBPROD"
AppIDel = "TradeSampleProd_V1R1"
ResolvedEar = "/WebSphere390/EAR/TradeSampleProd_V1R0_resolved.ear"

/* 1. Step - Create 1st conversation */
say "Creating conversation Delete ..."

name. = 0
name.1 = "conversationname"
name.2 = "conversationdescription"

val. = 0
val.1 = ConversationName1
val.2 = "Delete offending application ."

rc = 0
i = 1

```

```

do while(name.i <> '0')
    rc = XMLGEN("tempin" name.i val.i)
    if (rc == 4) then do
        say "Define server script createconversation failed while XMLGEN"
        exit
    end
    i = i+1
end;
rc = CB390CFG("-action 'createconversation'
              -xmlinput 'inputcreateconversation.xml'
              -input 'tempin' -output 'LEH013S1'")
if (rc == 4) then do
    say "Define script createconversation failed"
    exit
end
say "Conversation Delete created."
/* 1. Step - End*/

/* Step 2. Delete application */
say "Deleting Application " ApplDel " ..."

name. = 0
name.1 = "conversationname"
name.2 = "j2eeservername"
name.3 = "j2eeapplicationname"

val. = 0
val.1 = ConversationName1
val.2 = ServerName
val.3 = ApplDel

rc = 4
i = 1

do while( i <> '4')
    rc= XMLGEN("tempin" name.i val.i)
    if (rc == 4) then do
        say "LEH13S2 failed while XMLGEN"
        exit
    end
    i = i + 1
end;

rc = CB390CFG("-action 'deletej2eeapplication'
              -xmlinput 'inputdeletej2eeapplication.xml'
              -input 'tempin' -output 'LEH013S2'")

```

```

if (rc > 0) then do
    say "Script to delete J2EE application failed"
    exit
end
say "Delete application " ApplDel " complete."
/* 2. Step - End*/

/* 3. Step - Commit and activate conversation" */
say "Committing conversation Delete ..."
name. = 0
name.1 = "conversationname"

val. = 0
val.1 = ConversationName1

rc = 0
i = 1

do while(name.i <> '0')
    rc = XMLGEN("tempin" name.i val.i)
    if (rc == 4) then do
        say "Define server script commitconversation failed while XMLGEN"
        exit
    end
    i = i+1
end;

rc = CB390CFG("-action 'commitconversation'
              -xmlinput 'inputcommitconversation.xml'
              -input 'tempin' -output 'LEH013S3'")
if (rc == 4) then do
    say "Define server script commitconversation failed"
    exit
end
say "Conversation committed and activated."
/* 3. Step - End*/

/* 4. Step - Create 2nd conversation"*/
say "Creating conversation ReInstall ..."

name. = 0
name.1 = "conversationname"
name.2 = "conversationdescription"

val. = 0
val.1 = ConversationName2

```



```

val.2 = "Fall back application ."

rc = 0
i = 1

do while(name.i <> '0')
    rc = XMLGEN("tempin" name.i val.i)
    if (rc == 4) then do
        say "Define ceate conversation ReInstall failed while XMLGEN"
        exit
    end
    i = i+1
end;
rc = CB390CFG("-action 'createconversation'
              -xmlinput 'inputcreateconversation.xml'
              -input 'tempin' -output 'LEH013S4'")
if (rc == 4) then do
    say "Define script create conversation ReInstall failed"
    exit
end
say "Conversation ReInstall created."

/* 4. Step - End"*/

/* 5. Step - Process ear file to add application " */
say "Begin process ear file " ResolvedEar "...

name.=0
name.1 ="conversationname"
name.2 ="j2eeservername"
name.3 ="earfilename"
val.=0
val.1 = ConversationName2
val.2 = ServerName
val.3 = ResolvedEar
rc =4
i =1

/*Generate XML Input */
do while(name.i <> '0')
    rc =XMLGEN("tempin" name.i val.i)
    if (rc ==4)then do
        say "Process ear file failed while XMLGEN."
        exit
    end
    i =i+1
end;

```

```

rc =CB390CFG("-action 'processearfile'
             -xmlinput 'inputprocessearfile.xml'
             -input 'tempin' -output 'LEH013S5'")

if (rc ==4)then do
    say "Process ear file failed"
    exit
end
say "Process ear file complete."

/*Parse the result */
rc = XMLPARSE("processearfile" "ALL")
if (rc ==4)then do
    say "Process ear file failed while XMLPARSE"
    exit
end
/* 5. Step - End*/

/* 6. Step - Commit and activate conversation */
say "Committing conversation ReInstall ..."
name. = 0
name.1 = "conversationname"

val. = 0
val.1 = ConversationName2

rc = 0
i = 1

do while(name.i <> '0')
    rc = XMLGEN("tempin" name.i val.i)
    if (rc == 4) then do
        say "Define server script commitconversation failed while XMLGEN"
        exit
    end
    i = i+1
end;

rc = CB390CFG("-action 'commitconversation'
             -xmlinput 'inputcommitconversation.xml'
             -input 'tempin' -output 'LEH013S6'")
if (rc == 4) then do
    say "Define server script commitconversation failed"
    exit
end
say "Conversation ReInstall committed and activated."
/* 6. Step - End*/

```

```

say "SMAPI013 script completed!"
exit

error:
say "Error" rc "at line" sigl
say sourceline(sigl)
exit

```

Create a server and install a J2EE application

Example: C-4 Create new application server

```

/* REXX ----- */
/* ===== */
/*
/*   ITS0 2001 - Additional material
/*   RedBook: SG24-6521
/*
/* FILENAME: /u/andre/SMAPI014
/*
/* FUNCTION:
/* Defines a J2EE server and one server instance
/*
/* ===== */
/* This script defines a new J2ee server.
/* (RACF, PROCS and WLM is done outside of this script.
/* First a new conversation called "<date> <time>" will be added.
/* Then the server "BBTEST" will be added. Next, a server
/* instance, called BBTESTA1" will be added for conversation
/* Finally, the conversation will be validated, committed and
/* activated.
/*
/* ===== */

call syscalls 'ON'
signal on error

say "Starting define server script"

/* 1. Step - Create new conversation */
say "Creating conversation..."

ConversationName = date() " " time()
name. = 0
name.1 = "conversationname"
name.2 = "conversationdescription"

```

```

val. = 0
val.1 = ConversationName
val.2 = "Define new J2EE server BBTEST."

rc = 0
i = 1

do while(name.i <> '0')
  rc = XMLGEN("tempin" name.i val.i)
  if (rc == 4) then do
    say "Define server script ceateconversation failed while XMLGEN"
    exit
  end
  i = i+1
end;
rc = CB390CFG("-action 'createconversation'
              -xmlinput 'inputcreateconversation.xml'
              -input 'tempin' -output 'teriout1'")
if (rc == 4) then do
  say "Define script createconversation failed"
  exit
end
say "Conversation created"
/* 1. Step - End"*/

/* 2. Step - Define server"*/
say "Defining server..."

sval. = 0
sname. = 0

name. = 0
name.1 = "conversationname"
name.2 = "j2eeservername"
name.3 = "allownonauthenticatedclients"
name.4 = "allowuseridpasswd"
name.5 = "identityofthecontrolregion"
name.6 = "identityoftheserverregion"
name.7 = "isolationpolicy"
name.8 = "j2eeserverdescription"
name.9 = "localidentity"
name.10 = "procname"
name.11 = "remoteidentity"
name.12 = "serverregionstacksize"
name.13 = "security"
name.14 = "environment"
name.15 = "environment"
name.16 = "environment"

```

```

name.17 = "environment"
name.18 = "environment"

val. = 0
val.1 = ConversationName
val.2 = "BBTEST"
val.3 = 'Y'
val.4 = 'Y'
val.5 = "BBTESTC"
val.6 = "BBTESTS"
val.7 = 'Multiple_Transactions_Per_Server_Region'
val.8 = "Test Server"
val.9 = "BBTESTD"
val.10 = "BBTESTC"
val.11 = "BBTESTI"
val.12 = "0"
val.13 = "ISM_UserID_Password"
val.14 = "LIBPATH='/usr/lpp/db2/db2710/lib:" ||,
"/usr/lpp/java/IBM/J1.3/bin:/usr/lpp/java/IBM/J1.3/bin/classic:" ||,
"/usr/lpp/WebSphere/lib'"
val.15 = "CLASSPATH='/usr/lpp/WebSphere/lib/ws390srt.jar:" ||,
"/usr/lpp/WebSphere/lib/xerces.jar:" ||,
"/usr/lpp/WebSphere/lib/waswebc.jar:" ||,
"/usr/lpp/db2/db2710/classes/db2j2classes.zip:'"
val.16 = "JVM_LOGFILE='/tmp/jvm.log'"
val.17 = "TRACEALL='0'"
val.18 = "DB2SQLJPROPERTIES=" ||,
"'/usr/lpp/db2/db2710/classes/db2sqljjdbc.properties'"

rc = 0
i = 1
l = 1

do while(name.i <> '0')
    rc = XMLGEN("tempin" name.i val.i)
    if (rc == 4) then do
        say "Define Server Script createj2eeserver failed while XMLGEN"
        exit
    end
    i = i+1
end;

rc = CB390CFG("-action 'createj2eeserver'
              -xmlinput 'inputcreatej2eeserver.xml'
              -input 'tempin' -output 'teriout2'")
if (rc == 4) then do
    say "Define script createj2eeserver failed"
    exit
end

```

```

say "Server defined"
/* 2. Step - End"*/

/* 3. Step - Define server instance"*/
say "Defining server instance..."

sval. = 0
sname. = 0

name. = 0
name.1 = "conversationname"
name.2 = "servername"
name.3 = "serverinstancename"
name.4 = "serverinstancedescription"
name.5 = "systemname"

val. = 0
val.1 = ConversationName
val.2 = "BBTEST"
val.3 = "BBTESTA1"
val.4 = "BBTESTA1 on System SC59"
val.5 = "SC59"

rc = 0
i = 1
l = 1

do while(name.i <> '0')
    rc = XMLGEN("tempin" name.i val.i)
    if (rc == 4) then do
        say "Define Script creatserverinstance failed while XMLGEN"
        exit
    end
    i = i+1
end;

rc = CB390CFG("-action 'createserverinstance'
              -xmlinput 'inputcreateserverinstance.xml'
              -input 'tempin' -output 'teriout3'")
if (rc == 4) then do
    say "DefTest script createserverinstance failed"
    exit
end

say "Server instance defined"
/* 3. Step - End"*/

```

```

/* 4. Step - Commit and activate conversation"*/
say "Committing conversation..."
name. = 0
name.1 = "conversationname"

val. = 0
val.1 = ConversationName

rc = 0
i = 1

do while(name.i <> '0')
    rc = XMLGEN("tempin" name.i val.i)
    if (rc == 4) then do
        say "Define server script commitconversation failed while XMLGEN"
        exit
    end
    i = i+1
end;

rc = CB390CFG("-action 'commitconversation'
              -xmlinput 'inputcommitconversation.xml'
              -input 'tempin' -output 'teriout4'")
if (rc == 4) then do
    say "Define server script commitconversation failed"
    exit
end
say "Conversation committed and activated"
/* 4. Step - End"*/

say "Define server script completed"
exit

error:
say "Error" rc "at line" sigl
say sourceline(sigl)
exit

```



Multiple WSAS nodes in a sysplex

Here we provide a sample description on how to enable multiple WSAS nodes in the same sysplex.

Multiple WSAS nodes in a sysplex

Here is an approach to allow running of multiple WSAS nodes in a single sysplex.

Important: At the time we set up the infrastructure, a multi-node approach was not officially supported by IBM. However, the support has now been introduced and is available in some code and documentation updates. If you are able to apply the IBM-supported fix, we strongly recommend that you follow the approach that is documented in “IBM support for multiple nodes in a sysplex” on page 218.

For this discussion, let's assume a sysplex, SANDBOX, with four member systems: PRD1, PRD2, PRD3, and TST1, all of which are in a DB2 datasharing group.

The main WSAS node spans PRD1 and PRD2

It has the following properties:

- ▶ Uses DB2 in datasharing mode.
- ▶ Has a generic IP host name (i.e., prdcb.company.com).
- ▶ Uses 900 for the resolve IP port, and 5555 for the daemon IP port.
- ▶ Has a dedicated LDAP running on PRD1 using port 1389, not replicated on other systems. The suffixes are prefixed with “CB”.
- ▶ All WSAS servers start with the prefix “CB”, all profiles start with “CB”.
- ▶ The configuration HFS is mounted at /WebSphere390/CB390/.

The “test” WSAS node is only on the TST1

We made everything separate that we could: application environments, port usage, error log streams, configuration HFSs, etc.

The following can all be configured with the ISPF customization dialogues:

- ▶ Create a new DB2 subsystem, not a member of the DB2 datasharing group that supports the main WAS node.
- ▶ Use a system-specific daemon IP name (tst1.company.com).
- ▶ Use port 1900 and port 15555 for the resolve IP port and daemon IP port, respectively.
- ▶ Create a dedicated LDAP server on TST1 using port 2389. The suffixes are prefixed with “TB”.

- ▶ Use DASD-only logging for the error log, which has a different name from the production error log.
- ▶ All test WAS servers start with a prefix of “TB”, all profiles start with “TB”.
- ▶ The configuration HFS is mounted at /WebSphereTest/TB390/.

The following modifications must be made:

- ▶ Before running the RACF definition job, all occurrences of CBGUEST were changed to TBGUEST in the BBOWBRAC script.
- ▶ Before running the bootstrap (but after the BBOMCFG job) specify DATASHARING=0 in /WebSphereTest/TB390/PRDPLEX/initial/configuration.env.
- ▶ Before the bootstrap (but after BBOMCFG) change all occurrences of CBGUEST to TBGUEST in /WebSphereTest/TB390/PRDPLEX/initial/configuration.xml.

Common names and definitions shared by both nodes

- ▶ Sysplex name
- ▶ RACF database
- ▶ WLM service policy
- ▶ RRS log stream

Additional notes

- ▶ The WLM application environments are unique since all of the servers have different names.
- ▶ The RACF definitions for each WAS node, all servers, userids, groups, etc., are different. Profile names start with CB or TB. There are no overlapping UID/GIDs, etc.

IBM support for multiple nodes in a sysplex

In January 2002 IBM delivered official support for multiple WebSphere Application Server nodes in a sysplex.

This support was introduced with APAR PQ55866 and PTF UQ99328. The update consists of code and documentation updates. A separate document, entitled *“WebSphere Application Server V4.0.1 for z/OS and OS/390: Multiple Nodes in a Sysplex”* was made available

To download this document, go to the following Web site:

http://www.ibm.com/software/webservers/appserv/zos_os390/support.html

At a later time, the information in this document will be integrated into the WebSphere for z/OS formal publications.

The code update introduces a mechanism to enforce unique server names. The recommended setup is to have production and test environments separated in independent sysplex setups. A mechanism to partially enforce unique server names is provided. Server names are synchronized during System Management (SM) server instance startup using ENQs.

For each server name that is not unique (i.e., used for test and production nodes) a warning message (activity log) is issued to indicate the potential problem.

Limitations of this approach:

1. If on one node (e.g., the production node) the SM server instance holding the ENQs for generic servers is stopped, then uniqueness of server names cannot be ensured until an SM server instance on that node is restarted to pick up the enqueues again.
2. For servers added after the WAS node startup, uniqueness of server names cannot be ensured.



ISPF wizard-driven runtime upgrade

If you use the WebSphere ISPF wizard to upgrade your runtime environment, it will generate the upgrade instructions and jobs to run for that purpose. For your reference and as an example we provide the ISPF wizard-generated instructions for the runtime upgrade. These instructions will be written to the BBOMISTR member in your WebSphere JCL data set.

Instructions for migrating from WebSphere for z/OS V4.0 to V4.0.1

The customization dialog has created migration jobs based on the information you provided. These instructions tell you how to modify the operating system and run the jobs for migrating to WebSphere for z/OS V4.0.1.

RULES:

1. If you created the target data sets

ANDRE.WAS.CNTL
ANDRE.WAS.DATA

on another (driving) system, you must copy them to the target system and give them the same data set names.

2. You must perform these instructions on your target system.
3. You must use the jobs newly-created by the customization dialog, not your old V4.0 jobs, some of which have the same data set member names.

There are two procedures you can follow. Base your choice of which procedure to use on the following table:

IF YOU ARE MIGRATING TO WEBSPHERE FOR Z/OS V4.0.1 AND ...	THEN FOLLOW . . .	NOTES
You can interrupt service to clients (either on a monoplex or a sysplex)	"Steps for performing a warm start from WebSphere for z/OS V4.0 to V4.0.1 with a system or sysplex-wide restart"	
You cannot interrupt service to clients	"Steps for performing a rolling warm start from WebSphere for z/OS V4.0 to V4.0.1"	A rolling warm start requires the dual HFS structure described in "Overview of creating the proper HFS structure for upgrades" in WebSphere for z/OS: Installation and Customization.

Steps for performing a warm start from WebSphere for z/OS V4.0 to V4.0.1 with a system or sysplex-wide restart

BEFORE YOU BEGIN: You must be prepared to stop WebSphere for z/OS. If you have WebSphere for z/OS running in a sysplex as a host cluster, this procedure has you shut down the entire host cluster.

If you have WebSphere for z/OS running in a sysplex as a host cluster and want to maintain service to your clients during the warm start, see "Steps for performing a rolling warm start from WebSphere for z/OS V4.0 to V4.0.1."

You must copy the target data sets

ANDRE.WAS.CNTL
ANDRE.WAS.DATA

to your target system and give them the same data set names.

You must be running on your target system.

Perform the following steps to do the warm start.

1. Back up your current system. This includes:

- o The system management database
- o The LDAP database tables containing the naming space and the interface repository
- o Files in the HFS containing WebSphere for z/OS run-time information (usually mounted at "/WebSphere390/CB390").
- o WebSphere for z/OS PROCLIBs
- o WebSphere for z/OS LOADLIBs

For more information, see "Guidelines for backup of the WebSphere for z/OS system" in WebSphere for z/OS: Installation and Customization.

2. Stop all application servers and WebSphere for z/OS (on a sysplex, stop all clustered host instances).

3. Unmount the WebSphere for z/OS V4.0 HFSeS and mount the WebSphere for z/OS V4.0.1 HFSeS:

- o /usr/lpp/WebSphere
- o /usr/lpp/java/IBM/J1.3

4. If not already authorized, APF-authorize the following data sets:

BB0.V401.PID.SBBOLPA
BB0.V401.PID.SBBLOAD
BB0.V401.PID.SBBOLD2

5. On either the monoplex or one system in the sysplex, do the following:

- a. Run the following jobs. The jobs are in ANDRE.WAS.CNTL.

BBOMCFG	User ID requirement: UID=0.
Done:	This job updates the HFS previously-created for V4.0.
By:	Upon completion, examine the job output and use the z/OS UNIX shell to examine the directory structure. For more information about BBOMCFG and the HFS structure it creates, see WebSphere for z/OS: Installation and Customization.
BBOMPAT2	User ID requirement: DB2 SYSADM authority.
Done:	This job is a patch utility for the system management database.
By:	
BBOBIND	User ID requirement: DB2 SYSADM authority.
Done:	This job rebinds the system management database.
By:	
BBOIVPP	User ID requirement: DB2 SYSADM authority.
Done:	This job is a patch job for the database used by the installation verification programs (IVPs).
By:	
BBOIBN	User ID requirement: DB2 SYSADM authority.
Done:	This job re-binds the database used by the installation verification programs (IVPs).
By:	
BBOWMCP1	Optional job.
Done:	User ID requirement: Update authority for SYS1.PROCLIB.

By:	<p>ATTENTION: This job copies start procedures to SYS1.PROCLIB. You may have a private master subsystem PROCLIB to which you want to copy the start procedures.</p> <p>This job copies the tailored WebSphere for z/OS start procedures for the Daemon, System Management Server, Naming Server, Interface Repository Server, and IVP servers. If you made additional changes to the V4.0 start procedures, skip this job, do your changes, and copy the jobs by hand.</p>
BBOWMCP2	User ID requirement: UID=0
Done:	This job copies
By:	<ul style="list-style-type: none"> o The J2EE IVP shell script to /tmp o The CORBA IVP shell script to /tmp

- b. Either re-catalog your system PROCLIB or modify your server start procedures, PROGxx, and link list to point to the data sets with the new code.
- c. Load new run-time modules into LPA and update the link list. You can do this dynamically, but IBM recommends you re-IPL the system.
- d. Start the Daemon and application servers. When completed, you receive messages that the servers are ready for a warm start.

BBOU0579I CB SERIES SERVER <server> IS READY FOR WARMSTART.

where <server> is the name of the server.

TIPS:

- o There is no time limit set when you must warm start WebSphere for z/OS.
- o The Operations application flags servers ready for warm start with a green bullet.

6. If you are running on a sysplex, for each system, one at a time,

do the following:

- a. Either re-catalog your system PROCLIB or modify your server start procedures, PROGxx, and link list to point to the data sets with the new code.
- b. Load new run-time modules into LPA and update the link list. You can do this dynamically, but IBM recommends you re-IPL the system.
- c. Start the Daemon and application servers. When completed, you receive messages that the servers are ready for a warm start.

BBOU0579I CB SERIES SERVER <server> IS READY FOR WARMSTART.

where <server> is the name of the server.

TIPS:

- 1) There is no time limit set when you must warm start WebSphere for z/OS.
- 2) The Operations application flags servers ready for warm start with a green bullet.

7. When all servers are ready for warm start, on either the monoplex or one system in the sysplex, do the following:

- a. Stop the application servers and the Daemon.
- b. Start the Daemon with the warm start option:

s bbodmn,srvname='...',parms='-ORBCBI WARM'

- c. Run the following job. The job is in ANDRE.WAS.CNTL.

```
+-----+
| BBOWCMIG | User ID requirement: a user ID that has Systems
+-----+ Management administrative authority, such as CBADMIN.
| Done:    |
|          | This job migrates J2EE servers with a V4.0 level of the
|          | RemoteWebContainer object installed to a V4.0.1
|          | RemoteWebContainer object. Servlets and JSPs require
| By:      | this object in the server.
|          |
|          | If you receive a return code other than 0, see WebSphere
|          | for z/OS: System Management Scripting API for problem
+-----+
```

```
|          | determination information.          |
+-----+-----+-----+-----+-----+-----+
```

- d. Start your application servers with the warm start option:

```
s <server_proc>,srvname='...',parms='-ORBCBI WARM'
```

where <server_proc> is the application server start procedure.

You can also do the warm start for the application servers through the Operations application.

8. If you are running on a sysplex, for each of the other systems, one at a time, do the following:

- a. Stop the application servers and the Daemon.

- b. Start the Daemon with the warm start option:

```
s bbodmn,srvname='...',parms='-ORBCBI WARM'
```

- c. Start your application servers with the warm start option:

```
s <server_proc>,srvname='...',parms='-ORBCBI WARM'
```

where <server_proc> is the application server start procedure.

You can also do the warm start for the application servers through the Operations application.

9. Download and install the new level of the Administration application (SM EUI).

10. Re-run the installation verification programs (IVPs). For more information, see "Running the WebSphere for z/OS installation verification programs (IVPs)" in Chapter 3 of WebSphere for z/OS: Installation and Customization.

You are done when WebSphere for z/OS and all your application servers are running and the IVPs run successfully.

Steps for performing a rolling warm start from WebSphere for z/OS V4.0
to V4.0.1

BEFORE YOU BEGIN: You must have an HFS structure as described in
"Recommendations for the HFS structure" in WebSphere for z/OS:
Installation and Customization.

You must copy the target data sets

ANDRE.WAS.CNTL
ANDRE.WAS.DATA

to your target system and give them the same data set names.

You must be running on your target system.

Perform the following steps to do the warm start.

1. Back up your current system. This includes:
 - o The system management database
 - o The LDAP database tables containing the naming space and the
interface repository
 - o Files in the HFS containing WebSphere for z/OS run-time
information (usually mounted at "/WebSphere390/CB390").
 - o WebSphere for z/OS PROCLIBs
 - o WebSphere for z/OS LOADLIBs

For more information, see "Guidelines for backup of the WebSphere
for z/OS system" in WebSphere for z/OS: Installation and
Customization.

-
2. Set up your version-specific HFS for the new level of code.
-

3. If not already authorized, APF-authorize the following data sets:

BB0.V401.PID.SBBOLPA
BB0.V401.PID.SBBLOAD
BB0.V401.PID.SBBOLD2

4. Select a clustered host instance to begin the warm start process.
On that clustered host instance:
 - a. Stop the application servers and the WebSphere for z/OS Daemon.
 - b. Run the following jobs. The jobs are in ANDRE.WAS.CNTL.

BBOMCFG	User ID requirement: UID=0.
Done:	This job updates the HFS previously-created for V4.0.
By:	Upon completion, examine the job output and use the z/OS UNIX shell to examine the directory structure. For more information about BBOMCFG and the HFS structure it creates, see WebSphere for z/OS: Installation and Customization.
BBOMPAT2	User ID requirement: DB2 SYSADM authority.
Done:	This job is a patch utility for the system management database.
By:	
BBOBIND	User ID requirement: DB2 SYSADM authority.
Done:	This job rebinds the system management database.
By:	
BBOIVPP	User ID requirement: DB2 SYSADM authority.
Done:	This job is a patch job for the database used by the installation verification programs (IVPs).
By:	
BBOIBN	User ID requirement: DB2 SYSADM authority.
Done:	This job re-binds the database used by the installation verification programs (IVPs).

By:	
BBOWMCP1	Optional job.
Done:	User ID requirement: Update authority for SYS1.PROCLIB.
By:	<p>ATTENTION: This job copies start procedures to SYS1.PROCLIB. You may have a private master subsystem PROCLIB to which you want to copy the start procedures.</p> <p>This job copies the tailored WebSphere for z/OS start procedures for the Daemon, System Management Server, Naming Server, Interface Repository Server, and IVP servers. If you made additional changes to the V4.0 start procedures, skip this job, do your changes, and copy the jobs by hand.</p>
BBOWMCP2	User ID requirement: UID=0
Done:	This job copies
By:	<ul style="list-style-type: none"> o The J2EE IVP shell script to /tmp o The CORBA IVP shell script to /tmp

- c. Either re-catalog your system PROCLIB or modify your server start procedures, PROGxx, and link list to point to the data sets with the new code.
- d. Load new run-time modules into LPA and update the link list. You can do this dynamically, but IBM recommends you re-IPL the system.
- e. Switch the HFS that your system references with the SETOMVS command. Use the command to change the \$VERSION symbolic.

EXAMPLE: Previously, the \$VERSION symbolic was VersionA for all systems in the sysplex. Through the use of a built-in symbolic link, references to /usr resolved to VersionA/usr. To switch the HFS that this system references to, say, VersionB, issue:

```
setomvs version=VersionB
```

- f. Start the Daemon and application servers. When completed, you receive messages that the servers are ready for a warm start.

BBOU0579I CB SERIES SERVER <server> IS READY FOR WARMSTART.

where <server> is the name of the server.

TIPS:

- o There is no time limit set when you must warm start WebSphere for z/OS.
- o The Operations application flags servers ready for warm start with a green bullet.

-
- 5. For each of the remaining clustered host instances, one at a time, do the following:

- a. Stop the application servers and the WebSphere for z/OS Daemon.
- b. Either re-catalog your system PROCLIB or modify your server start procedures, PROGxx, and link list to point to the data sets with the new code.
- c. Load new run-time modules into LPA and update the link list. You can do this dynamically, but IBM recommends you re-IPL the system.
- d. Switch the HFS that your system references with the SETOMVS command. Use the command to change the \$VERSION symbolic.

EXAMPLE: Previously, the \$VERSION symbolic was VersionA for all systems in the sysplex. Through the use of a built-in symbolic link, references to /usr resolved to VersionA/usr. To switch the HFS that this system references to, say, VersionB, issue:

setomvs version=VersionB

- e. Start the Daemon and application servers. When completed, you receive messages that the servers are ready for a warm start.
-

- 6. Select one system and do the following:

- a. Stop the application servers and the Daemon.

- b. Start the Daemon with the warm start option:

```
s bbodmn,srvname='...',parms='-ORBCBI WARM'
```

- c. Run the following job. The job is in:
ANDRE.WAS.CNTL

BBOWCMIG	User ID requirement: a user ID that has Systems Management administrative authority, such as CBADMIN.
Done:	This job migrates J2EE servers with a V4.0 level of the RemoteWebContainer object installed to a V4.0.1 RemoteWebContainer object. Servlets and JSPs require this object in the server.
By:	If you receive a return code other than 0, see WebSphere for z/OS: System Management Scripting API for problem determination information.

- d. Start your application servers with the warm start option:

```
s <server_proc>,srvname='...',parms='-ORBCBI WARM'
```

where <server_proc> is the application server start procedure.

You can also do the warm start for the application servers through the Operations application.

7. On the other systems in the sysplex, do the following, one at a time:

- a. Stop the application servers and the Daemon.

- b. Start the Daemon with the warm start option:

```
s bbodmn,srvname='...',parms='-ORBCBI WARM'
```

- c. Start your application servers with the warm start option:

```
s <server_proc>,srvname='...',parms='-ORBCBI WARM'
```

where <server_proc> is the application server start procedure.

You can also do the warm start for the application servers through the Operations application.

8. Download and install the new level of the Administration application (SM EUI).

9. Re-run the installation verification programs (IVPs). For more information, see "Running the WebSphere for z/OS installation verification programs (IVPs)" in Chapter 3 of WebSphere for z/OS: Installation and Customization.

You are done when WebSphere for z/OS and all your application servers are running and the IVPs run successfully.



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/SG246521>

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, SG246521.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
SG246521.zip	Zipped Code Samples

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 5 MB minimum

Operating System: Windows and z/OS

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

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 237.

- ▶ *Migrating WebLogic Applications to WebSphere Advanced Edition*, SG24-5956
- ▶ *Programming J2EE APIs with WebSphere Advanced*, SG24-6124
- ▶ *WebSphere Version 4 Application Development Handbook*, SG24-6134
- ▶ *S/390 File and Print Serving*, SG24-5330
- ▶ *OS/390 e-business Infrastructure: IBM HTTP Server V5.1 for OS/390*, SG24-5603
- ▶ *Java Programming Guide for OS/390*, SG24-5619
- ▶ *EJB Development with VisualAge for Java for WebSphere Application Server*, SG24-6144
- ▶ *TCP/IP in a Sysplex*, SG24-5235
- ▶ *WebSphere V4.0 Advanced Edition Handbook*, SG24-6176
- ▶ *e-business Enablement Cookbook for z/OS Volume I: Technology Introduction*, SG24-5664 (Tamas residency)
- ▶ *e-business Enablement Cookbook for z/OS Volume II: Infrastructure for Java-based Solutions*, SG24-5981 (Tamas residency)
- ▶ *e-business Enablement Cookbook for z/OS Volume III: Java Development*, SG24-5980 (tamas residency)
- ▶ *Web Services Wizardry with WebSphere Studio Application Developer*, SG24-6292

Other resources

These publications are also relevant as further information sources:

- ▶ *WebSphere Application Server V4.0.1 for z/OS and OS/390: Installation and Customization*, GA22-7834
- ▶ *WebSphere Application Server V4.0.1 for z/OS: Assembling J2EE Applications*, SA22-7836
- ▶ *WebSphere Application Server V4.0.1 for z/OS and OS/390: Migration*, GA22-7860
- ▶ *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management Scripting API*, SA22-7839
- ▶ *WebSphere Application Server V4.0.1 for z/OS and OS/390: Messages and Diagnosis*, GA22-7837
- ▶ *WebSphere Application Server V4.0.1 for z/OS and OS/390: Operations and Administration*, SA22-7835
- ▶ *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management User Interface*, SA22-7838
- ▶ *WebSphere Application Server V4.0.1 for z/OS and OS/390 Program Directory*, G110-0680
- ▶ *HTTP Server Planning, Installing, and Using*, SC34-4826
- ▶ *IBM DB2 Universal Database SQL Reference Version 7*, SC09-2974
- ▶ *IBM DB2 Universal Database SQL Reference Version 7*, SC09-2975
- ▶ *DB2 UDB for OS/390 and z/OS V7 Application Programming Guide and Reference for Java*, SC26-9932
- ▶ *DB2 UDB for OS/390 and z/OS V7 SQL Reference*, SC26-9944

Referenced Web sites

These Web sites are also relevant as further information sources:

- ▶ IBM WebSphere Web site
<http://www.ibm.com/software/webservers/appserv/>
- ▶ IBM WebSphere Library
<http://www.ibm.com/software/webservers/appserv/library.html>
- ▶ IBM Techdocs, we especially worked with
Configuring Web Applications
www.ibm.com/support/techdocs

- ▶ IBM Java for z/OS Web site
<http://www.s390.ibm.com/java>.
- ▶ Java2 Platform Enterprise Edition specification v1.2
<http://java.sun.com/j2ee/download.html>

How to get IBM Redbooks

Search for additional Redbooks or Redpieces, view, download, or order hardcopy from the Redbooks Web site:

ibm.com/redbooks

Also download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become Redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

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.

Glossary

Address Space A z/OS address space is the virtual logical memory space hosting an application. A z/OS UNIX process runs as an address space.

Application Assembly Tool (AAT) A free, Windows-based tool which can be used to create an archive which bundles all files (java class files, HTML files, graphics, etc.) used in an enterprise (J2EE) application together with an XML deployment descriptor. This archive is named with the suffix .ear, and can be deployed directly to the application server.

An archive can contain a bundle of other archives which are referred to as modules in this case. In 4.2, “Common migration steps” on page 51 we make use of the AAT to generate an .ear file of our sample application.

Cluster In the z/OS world, a cluster typically refers to a number of nodes which are organized in a way so that they act as one machine, from a client’s point of view. The machines are clustered in order to bundle performance.

A special mainframe form of a cluster is a system complex (sysplex). Machines in a sysplex are connected via a Coupling Facility which enables very efficient communication and sharing between the machines.

Console A z/OS console is the logical screen where all system events are written.

Direct access storage device (DASD)

Refers to a mass storage medium on which a computer stores data. This is realized by a number of hard drives or disks. It does not refer to tape.

Hierarchical File System (HFS) In z/OS the file system HFS is used. It is an implementation of the UNIX/POSIX file structure. HFS also refers to a grouping of directories and files that are mounted as a group.

Installation verification procedure

(IVP) This procedure provides an easy way for an administrator to test if a software product is successfully installed and configured, and that the correct authorization is set up.

The administrator runs an IVP batch job and receives a log file with information on eventual problems. IVP comes with most IBM products like WebSphere for z/OS and DB2 Data Joiner for z/OS. See 5.3.5, “Automatic application verification” on page 113 for a more detailed description on how we make use of IVP to test a successful migration from a test system to a production system.

IShell Interactive Shell; a set of ISPF panels to manage UNIX System Services.

Node A WebSphere Application Server node is a related set of servers that share the same system management configuration database.

OEdit The 3270-based editor to edit files hosted in the UNIX file tree.

OMVS A shell session manager available from a 3270 terminal.

Parallel Sysplex A Parallel Sysplex consists of one (monoplex) or more instances of the z/OS or OS/390 operating system which are configured to share resources and workload.

Server or J2EE Server Within a system, or more commonly within the several systems of a sysplex, there may be several server instances, all of the same type. A server is the unit of configuration that defines the common characteristics for these several server instances and into which applications are installed.

Server instance This represents the instantiation of a functional unit that provides services to one or more clients over a network. This instantiation might be realized by a process, or by a part of another process, or by multiple processes. For example, two instances of an application server running on the same machine can share the workload that is generated by HTTP requests to a specific URL. On z/OS, a mechanism called Workload Manager can even start an additional instance of this server, if necessary, to handle all requests in a desired timeframe.

Shell The user interface of UNIX system softwares. In z/OS, an xpg4.2-compliant shell is used. Very often OMVS is used as an interface for z/OS shells.

Sysout The regular output for a program on z/OS is SYSOUT. It is the functional equivalent of stdout on UNIX. In batch, there can be multiple SYSOUT.

System A single instance of the z/OS or OS/390 operating system in a sysplex.

System complex (sysplex) A cluster of z/OS nodes which are connected through a Coupling Facility to bundle performance. From a client's point of view, the whole sysplex might look like one machine.

System Management Enhanced User Interface (SMEUI) A Windows-based tool to perform administrative tasks for an IBM software product like WebSphere from a Windows workstation. In our porting example (see 4.1, "Minimum migration steps for Trade2" on page 46), we use the SMEUI to deploy a new application to WebSphere on z/OS.

TCB Task Control Block; manages dispatchable tasks. Each UNIX thread is assigned to a TCB.

Vi A popular UNIX editor. It can only be used from an ASCII Telnet connection.

Workload Manager (WLM) In z/OS, the WLM dispatches workloads to the available system resources. The system administrator defines performance goals as well as business importances for every goal.

The WLM constantly monitors the whole system and adopts processing to meet these goals. Therefore, it assigns resources like CPU and storage to the current processes. Refer to the section on balancing a diverse workload in *IBM Component Broker on System/390*, SG24-5127, for a detailed description of how the WLM works.

Note:

The WLM in z/OS has nothing to do with the WLM of WebSphere Application Server, which is available on distributed platforms only and makes use of a completely different paradigm.

Index

A

- Additional material 233
- APAR II12832 66
- APAR PQ55866 218
- Application analysis 32
- Application Assembling Tool (AAT)
 - distributed 35
 - z/OS 36
- Application Assembly Tool (AAT) 38
 - Installation 47
- Application Lifecycle 81
- Application migration 31
- ASCII editing 73
- ASCII-to-EBCDIC conversion 72
- Automated application verification 113
- Automated installation 105

B

- BBOMISTR 219
- Business or EJB tier 16

C

- Change and Problem Management 120
 - concepts 120
 - controlled deployment 123
 - deployment and activation 121
 - deployment integration 125
 - Deployment Principles 121
 - managing conversation 125
 - Software configuration management 121
 - Software Configuration Management (SCM) 121
- Change and problem management 120
- Change Management
 - minor changes 75
- Cisco MNLN 85
- ClassLoader 68
 - ClassLoader problems 68
- Client tier 15
- Code optimization 70
- Code pages 33
 - ASCII/EBCDIC issues 72

- EBCDIC 33

- Column names 34
- Compliance to J2EE specifications 33
- Connection Pooling 71
- Connectors 21
 - connectors 33
- Content management 120
- conversion from ASCII to EBCDIC 72
- current.env 50

D

- Data or EIS tier 16
- Datasources 50
- DB2
 - Changing SQL commands 64
 - Column names 34
 - CTHREAD 112
 - DB2 UDB issues 74
 - IDBACK 112
 - install application database tables 49
- Deploy- and Runtime-Environment 38
 - distributed 38
- Development system 20
- Digest authentication 19
- Disruptive runtime upgrade 119
- Distributed Debugger 147
- DNS/WLM 85

E

- Eclipse 36
- EJB Deploy Tool 38
- EJB Deploy Tool (ejbdeploy) 35
 - ejbdeploy 35
- ENCLAVE directive 100
- Enterprise Application Archive (EAR) 35
- Enterprise Java Bean (EJB)
 - EJB access beans 28
 - EJB links 48
 - Runtime restrictions 66
 - Session EJBs transaction attributes 72

F

Form-based login 19

H

hints and tips 72

Housekeeping 50

HTTP Transport Handler 98

HTTP transport handler

WLM classification 103

I

IDE 36

Indirect Object Reference (IOR) 56

Initial Context Factory 62

Integrated Development Environment (IDE) 36

Inter-ORB Protocol (IIOP) 15

Introscope 151

Enterprise Manager 164

ISPF Wizard 219

J

J2EE APIs 5

J2EE Blueprint 15

J2EE resource 91

J2EE server

customization 193

J2EE test server 97

manual installation steps 94

naming convention 193

production server 98

z/OS setup 85

J2EE Standard 5

Java 2 Enterprise Edition (J2EE)

Compliance 33

Introduction 4

Java Naming & Directory Interface (JNDI)

Concept 53

EJB lookup 58

finding EJBs 55

Initial Context Factory name 62

IOR 56

Java Logical Reference 55

JNDI problems 66

LDAP 56

LDAP Entry 56

Locating Datasources 59

Namespace 53

Overview 53

z/OS implementation 53

Java Native Directory Interface (JNDI) 53

JCA connectors 64

JCA connectors issues 64

JCL 177

JDBC

javax.sql.DataSource 71

jdwp port 148

Jinsight

135

Jinsight 2.1

137

deployment 138

Download 137

Installation 137

visualizer 141

JMS Listener 66

JVM_DEBUG_PORT 148

L

Lightweight Directory Access Protocol (LDAP)

LDAP authentication 66

Locating Datasources 59

M

Management summary 129

Operational view 130

Technical view 130

Total Cost of Ownership 131

Migration check list 51

Migration infrastructure 34

setup 34

Migration outline 30

Model, View, Controller Architecture (MVC) 25

Multiple Nodes in a Sysplex 218

APAR PQ55866 218

PTF UQ99328 218

Multiple WSAS nodes in Sysplex 215

N

Namespaces

global 61

local 61

Naming Convention 193

Native methods 33

Nondisruptive application upgrade 117

Nondisruptive runtime upgrade 120
non-J2EE connector 63

O

Object Level Trace (OLT)
 143
 Distributed Debugger 147
 execution modes 145
 Implementation 145
 OLT server. 145
 OLT viewer 145
 trace mode 144
Operational view 130

P

Persistence Builder 64
Porting check list 51
PQ46174 100
PQ53684 68
PQ56008 100
PQ59911 104
Presentation or Web tier 15
Production system 22
PTF UQ99328. 218

Q

quality assurance system 22

R

Redbooks 235
Redbooks Web site 237
 Contact us xv
Redeployment of an .ear file 75
Remapping CMP Beans 63
Remote connectors 33
Remote Debugger 148
Remote Method Invocation (RMI) 15
restrictions, runtime 66
REXX-driven configuration 106
Roles
 Development job roles 9
 Development roles 8
 EJB roles 8
 Implementation job roles 10
 DB2 administrator 12
 Network administrator 12
 Security administrator 11

 System administrator 11
 System operator 12
 System programmer 11
 Implementation roles 9
 Infrastructure roles 9
 infrastructure roles 8
 Job roles 8
 overview 8
 Skills 12
run-as 19

S

Sample code 177
Scalable Mode Classification 100
Scripted installation 105
Security
 Authentication 19
 Certificates 19
 Digest authentication 19
 Form based login 19
 HTTPS 20
 WebSphere security model 33
Security model 19
 run-as 19
Setting up J2EE servers 85
Skills 12
SMP/E 119
Source code 32
SQL commands 64
Standalone Mode Classification 100
System landscape 20
System Management End User Interface (SMUI)
36
System Management Enhanced User Interface
(SMUI)
 Installation 47
System Management Scripting API (SMAPI) 106
 REXX support 106
 samples 201
System Measurement Facility (SMF)
 enable data recording 97
System Modification Product/Extended (SMP/E)
119

T

Technical view 130
Test client 115
test client

- batch mode 116
- command line mode 116
- Test system 21
- Tiers
 - 3-tier applications 17
 - Multi-tier applications 16
 - multi-tiered Architecture 14
 - overview 14
 - The individual tiers 15
 - topologies 14
 - Two-tier applications 16
- Total Cost of Ownership 131
- Trade2 40
 - migration 46
 - porting 46
- Topology and Architecture 25

U

- UQ57041 100
- UQ61610 163
- UQ62753 100

V

- verification system 22
- VisualAge for Java (VAJ) 21
- VisualAge for Java Version 4.0 35
- Vocabulary 7

W

- Web sites 236
- Web tier 15
- WebSphere administrator 10
- WebSphere Application Server
 - characteristics 4
 - J2EE support 4
 - Job Roles 10
 - Security model 19
 - Skills 10
 - Vocabulary 7
- WebSphere Library 236
- WebSphere sample performance benchmark application 23, 39
 - Trade2 25
 - Web Primitives 25
- WebSphere Studio Application Developer 20, 21, 35, 36
 - deployment 40

- Remote Debugger 148
- supported standards 37
- Wily Technology 151
- WLMSN 100
- Workload Distribution 85
 - Cisco MNLN 85
 - DNS/WLM 85
 - WebSphere Edge Server 85
- Workload Manager (WLM) 82
 - Address space management 99
 - APPLENV directive 101
 - Application Environment 82
 - CB default classification 100
 - Classification Overrides 83
 - ENCLAVE directive 100
 - enclaves 99
 - HTTP transport handler 103
 - IBM HTTP Server plugin 100
 - scalable mode classification 100
 - service class 99
 - standalone mode classification 100
 - Sysplex routing 99
 - WLM classification 98
 - WLMSN directive 100
- ws.ext.dirs 72

X

- XML over HTTP 15
- XML parsing 15



Migrating WebSphere Applications to z/OS

(0.2" spine)
0.17" <-> 0.473"
90 <-> 249 pages



Migrating WebSphere Applications to z/OS

Technical and management views

Tooling and scripting for optimization

Operational Integration and Change Management

This IBM Redbook covers the technical and organizational issues involved in migrating a WebSphere EJB application from a distributed environment to the z/OS platform. It provides guidance to application developers and WebSphere on z/OS administrators.

Part 1 covers the migration aspects and details the following:

- Hints and tips on how to optimize application code to fit seamlessly into its final deployment environment
- J2EE naming concepts that you need to perform the migration job
- A complete walkthrough of a migration project
- The necessary setup of deployment, test and production systems within the Parallel Sysplex

Part 2 describes how to manage WebSphere applications and details the following:

- How to integrate your migrated application into the managed z/OS infrastructure, for example Change Management and System Automation
- Centralized system administration, change management, and highly automated processes

Part 3 describes the following tools for WebSphere Application Server on z/OS:

- Jinsight
- Object Level Trace
- Introscope

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