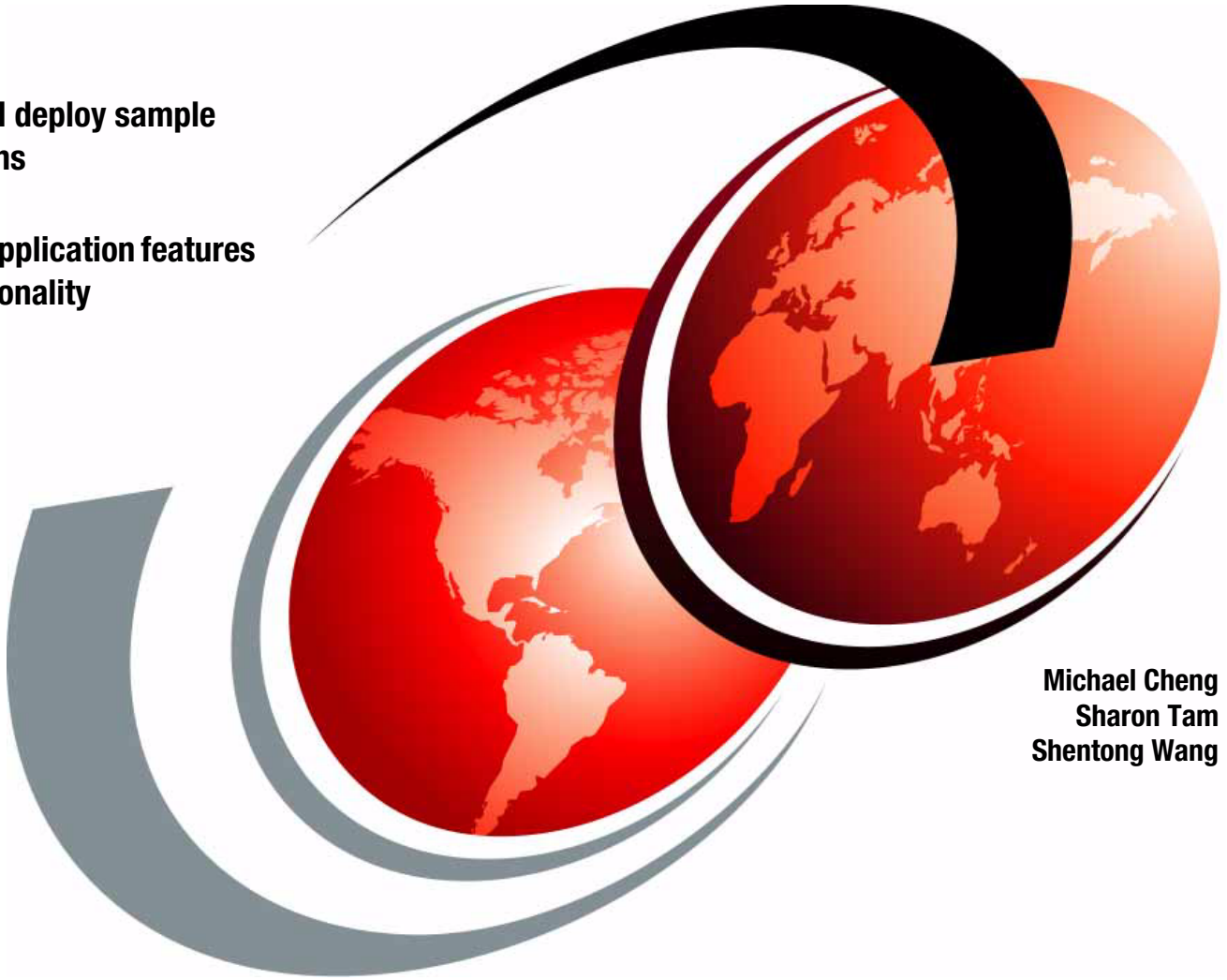
WebSphere® software

IBM

WebSphere Application Server for Developers V7

- Learn about WebSphere Application Server V7
- Create and deploy sample applications
- Enhance application features and functionality



Michael Cheng
Sharon Tam
Shentong Wang

ibm.com/redbooks

Redbooks

International Technical Support Organization

WebSphere Application Server for Developers V7

November 2010

SG24-7913-00

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

First Edition (November 2010)
This edition applies to Version 7, Release 0, of WebSphere Application Server.

© Copyright International Business Machines Corporation 2010. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices vii
Trademarks viii

Preface ix
The team who wrote this book ix
Now you can become a published author, too! x
Comments welcome. x
Stay connected to IBM Redbooks x

Chapter 1. Setting up WebSphere Application Server. 1
1.1 Downloading WebSphere Application Server for Developers V7 2
1.2 Installing WebSphere Application Server 3
1.3 Creating profiles 3
1.4 Using the command line 3
1.5 Using the GUI 4
1.6 Verifying the installations 18
1.7 Development environment 19

Chapter 2. WebSphere administration 21
2.1 Overview of WebSphere administration architecture 22
2.2 WebSphere Application Server Information Center 23
2.3 WebSphere configuration repository. 24
2.3.1 Scoping. 24
2.3.2 variables.xml. 25
2.3.3 The resources.xml file. 25
2.3.4 The serverindex.xml file, installed applications, and ports 25
2.3.5 The server.xml file. 26
2.4 Application-related files. 26
2.4.1 Configuration compared to runtime changes 27
2.5 Command-line tools 27
2.5.1 The startServer command. 27
2.5.2 The stopServer command. 28
2.5.3 serverStatus 28
2.5.4 The versionInfo command 29
2.6 Administrative console 29
2.6.1 Logging in to the administrative console. 29
2.6.2 Administrative console layout 30
2.6.3 Follow the breadcrumb 30
2.6.4 Configuration compared to run time 31
2.6.5 Saving or discarding configuration changes 31
2.6.6 Ports 32
2.6.7 Applications. 33
2.6.8 Scoping. 34
2.7 The wsadmin command and scripting. 35
2.7.1 Configuring wsadmin 35
2.7.2 Starting wsadmin 35
2.7.3 Exiting wsadmin 35
2.7.4 wsadmin objects 36
2.8 Log files. 57

© Copyright IBM Corp. 2010. All rights reserved.

2.8.1 Log header . . . 57
2.8.2 JVM Log Message Format . . . 59
2.8.3 Sample log messages . . . 59
2.8.4 FFDC logs . . . 60

Chapter 3. DictionaryApp example application . . . 63
3.1 What to know before proceeding . . . 64
3.2 DictionaryApp V1 overview . . . 64
3.3 DictionaryApp V1 source files . . . 65
3.3.1 showentry.jsp . . . 65
3.3.2 DictionaryServlet.java . . . 66
3.3.3 Dictionary.java . . . 67
3.3.4 DictionaryJDBCAdapter.java . . . 68
3.3.5 Entry.java . . . 69
3.3.6 Logger.java . . . 71
3.4 Creating a Derby database . . . 71
3.5 Creating a data source . . . 72
3.5.1 Administrative console . . . 72
3.5.2 wsadmin scripting . . . 78
3.5.3 Property file-based configuration . . . 80
3.6 Deploying, updating, and accessing DictionaryApp V1 . . . 81
3.6.1 Administrative console . . . 81
3.6.2 wsadmin scripting . . . 94
3.7 Accessing DictionaryApp V1 . . . 96

Chapter 4. Incorporating EJB3 and JPA into DictionaryApp V2 . . . 99
4.1 DictionaryApp V2 overview . . . 100
4.2 DictionaryApp source files . . . 100
4.2.1 DictionaryManagerLocal.java . . . 101
4.2.2 DictionaryManagerBean.java . . . 101
4.2.3 DictionaryJPAAdapter.java . . . 102
4.2.4 Entry.java . . . 103
4.2.5 DictionaryServlet.java . . . 103
4.2.6 showentry.jsp . . . 104
4.2.7 persistence.xml . . . 104
4.3 Redeploying DictionaryApp V2 . . . 104
4.3.1 Administrative console redeployment . . . 104
4.3.2 wsadmin scripting . . . 118

Chapter 5. Incorporating JMS and MDBs into DictionaryApp V3 . . . 121
5.1 DictionaryApp V3 overview . . . 122
5.2 DictionaryApp V3 source files . . . 122
5.2.1 DictionaryManagerLocal.java . . . 123
5.2.2 DictionaryManagerBean.java . . . 123
5.2.3 LookupLoggerBean.java . . . 125
5.2.4 DefineLoggerBean.java . . . 125
5.2.5 DictionaryServlet.java . . . 126
5.3 WebSphere Service Integration Bus . . . 127
5.4 Setting up the service integration bus for DictionaryApp V3 . . . 128
5.4.1 Administrative console . . . 128
5.4.2 wsadmin scripting . . . 141
5.5 Setting up the client (message sender) side of DictionaryApp V3 . . . 142
5.5.1 Administrative console . . . 142
5.5.2 wsadmin scripting . . . 154

5.6 Setting up the server (message receiver) side of DictionaryApp V3 155
5.6.1 Administrative console 156
5.6.2 wsadmin scripting 161
5.7 Redeploying DictionaryApp V3 162
5.7.1 Administrative console 162
5.7.2 wsadmin scripting 165
5.8 Accessing DictionaryApp V3 166

Chapter 6. Incorporating RESTful web services into DictionaryApp V4 167
6.1 Setting up Web 2.0 Feature Pack for WebSphere Application Server 7.0 168
6.2 Downloading Web 2.0 Feature Pack for WebSphere Application Server 7.0 168
6.2.1 Installing Web 2.0 Feature Pack for WebSphere Application Server 7.0 172
6.3 Updating DictionaryApp V4 source files 172
6.3.1 DictionaryApplication.java 173
6.3.2 DictionaryService.java 174
6.3.3 web.xml 174
6.4 Redeploying DictionaryApp V4 175
6.5 Accessing DictionaryApp V4 175
6.6 Creating a web service client for DictionaryApp V4 176

Chapter 7. Including WebSphere-specific bindings files for DictionaryApp V5 179
7.1 DictionaryApp V5 bindings files 180
7.2 ibm-web-bnd.xml for DictionaryWeb 180
7.3 ibm-web-bnd.xml for DictionaryEJB 181
7.4 Redeploying DictionaryApp V5 181
7.4.1 Administrative console 182
7.4.2 wsadmin scripting 183

Chapter 8. Debugging 185
8.1 Enabling WebSphere debug mode 186
8.2 Enabling verbose garbage collection 187
8.3 Generating a Java heap dump 191
8.3.1 wsadmin scripting 192
8.3.2 Process signaling 192
8.4 Generating Java core dump 195
8.4.1 wsadmin scripting 196
8.4.2 Process signaling 196
8.5 IBM Support Assistant 197
8.5.1 Downloading IBM Support Assistant 198
8.5.2 Installing IBM Support Assistant 200
8.5.3 Downloading the tools for data analysis 200
8.6 Using Garbage Collection and Memory Visualizer 203
8.7 Using Memory Analyzer 206
8.8 Using IBM Thread and Monitor Dump Analyzer for Java 211

Appendix A. Development tools reference 219
A.1 Downloading Eclipse 221
A.2 Installing Eclipse 222
A.3 Importing DictionaryApp into Eclipse 222
A.3.1 Working with separate versions of DictionaryApp 223
A.3.2 DictionaryApp V3 223
A.3.3 DictionaryApp V4 224
A.4 Exporting DictionaryApp EAR in Eclipse 224
A.5 Debugging DictionaryApp in Eclipse 224

A.6 Creating your own bindings file 225
A.6.1 Importing WebSphere XML schema into Eclipse 225
A.6.2 Creating an XML file with the XML schema 226

Appendix B. WebSphere configurations 227
B.1 WebSphere Application Server directory reference 228
B.2 Directory structure 228
B.2.1 Configuration directory 231
B.2.2 Configuration documents 232
B.3 Changing server ports 233
B.3.1 Administrative console 233
B.3.2 wsadmin 233

Appendix C. Additional material 235
Locating the web material 235
Using the web material 235

Related publications 237
IBM Redbooks publications 237
Online resources 237
How to get IBM Redbooks 238
Help from IBM 238

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 illustrate 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.

© Copyright IBM Corp. 2010. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

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

CICS®
ClearCase®
IBM®
IMS™
Rational®
Redbooks®
Redpaper™
Redbooks (logo) ®
RequisitePro®
WebSphere®

The following terms are trademarks of other companies:

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

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

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

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

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

Preface

This IBM® Redbooks® publication can help you install, tailor, and configure the WebSphere® Application Server for Developers V7 on the Microsoft® Windows® platforms. WebSphere Application Server for Developers is a no-charge version of the WebSphere Application Server for use in a development environment only. It allows application developers to develop and unit test against the same run time as the production version of the WebSphere Application Server.

This book tells you how to perform these tasks:

- Download and install WebSphere Application Server for Developers V7.
- Use the command-line tools, the web-based administrative console, and scripting tools.
- Deploy a web application with Java™ Database Connectivity (JDBC) to the application server with the first version of a sample application.
- Configure the sample application with Enterprise JavaBeans 3 (EJB3) and the Java Persistence API (JPA).
- Add Java Message Service (JMS) and message-driven beans (MDBs) to the sample application and configure the built-in system integration bus (SIBus) messaging infrastructure.
- Add REStful web service to the sample application.
- Incorporate WebSphere-specific application binding files with the application.
- Enable debugging, and produce and analyze Java virtual machine (JVM) outputs.
- Use Eclipse to view and debug the sample applications.

The team who wrote this book

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

Michael Cheng is the System Management Architect for WebSphere Application Server, currently working on the WebSphere Network Deployment and large topology system management architecture. He has been a contributor to IBM object-oriented and web technology since 1995, on areas including Object Request Broker, high availability architecture, and EJBs. He was also the technical lead for the IBM reference implementation and test compliance kit of Java Service Request 109: Implementing Enterprise Web Services. He holds a Doctorate in Computer Science from the University of Wisconsin-Madison.

Sharon Tam is a WebSphere Application Server Co-op at IBM Austin. She holds a Bachelor of Science degree in Computer Science and Engineering from Massachusetts Institute of Technology. Her interests include web applications, wireless communications, and affective computing, which is the development of technologies that relate to or influence emotion.

Shentong Wang is a WebSphere Application Server Co-op at IBM Austin. He is a junior at Cornell University studying computer science. His interests include OpenGL, shader development, and general-purpose graphics processing unit (GPU) computing.

© Copyright IBM Corp. 2010. All rights reserved.

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

- Stephen Smith
- Vishwanath Venkataramappa
- Bryant Luk
- Lohitashwa Thyagaraj
- Kevin Sutter
- Jim Knutson

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and client satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Setting up WebSphere Application Server

This chapter provides instructions for installing WebSphere Application Server for Developers on your system.

This chapter contains the following information:

- Downloading WebSphere Application Server for Developers V7
- Installing WebSphere Application Server
- Creating profiles
- Verifying the installation

© Copyright IBM Corp. 2010. All rights reserved.

1.1 Downloading WebSphere Application Server for Developers V7

WebSphere Application Server for Developers is offered at no charge. You can obtain it online by following these steps:

1. Go to the following URL, as shown in Figure 1-1:

   `http://www.ibm.com/developerworks/websphere/downloads/`

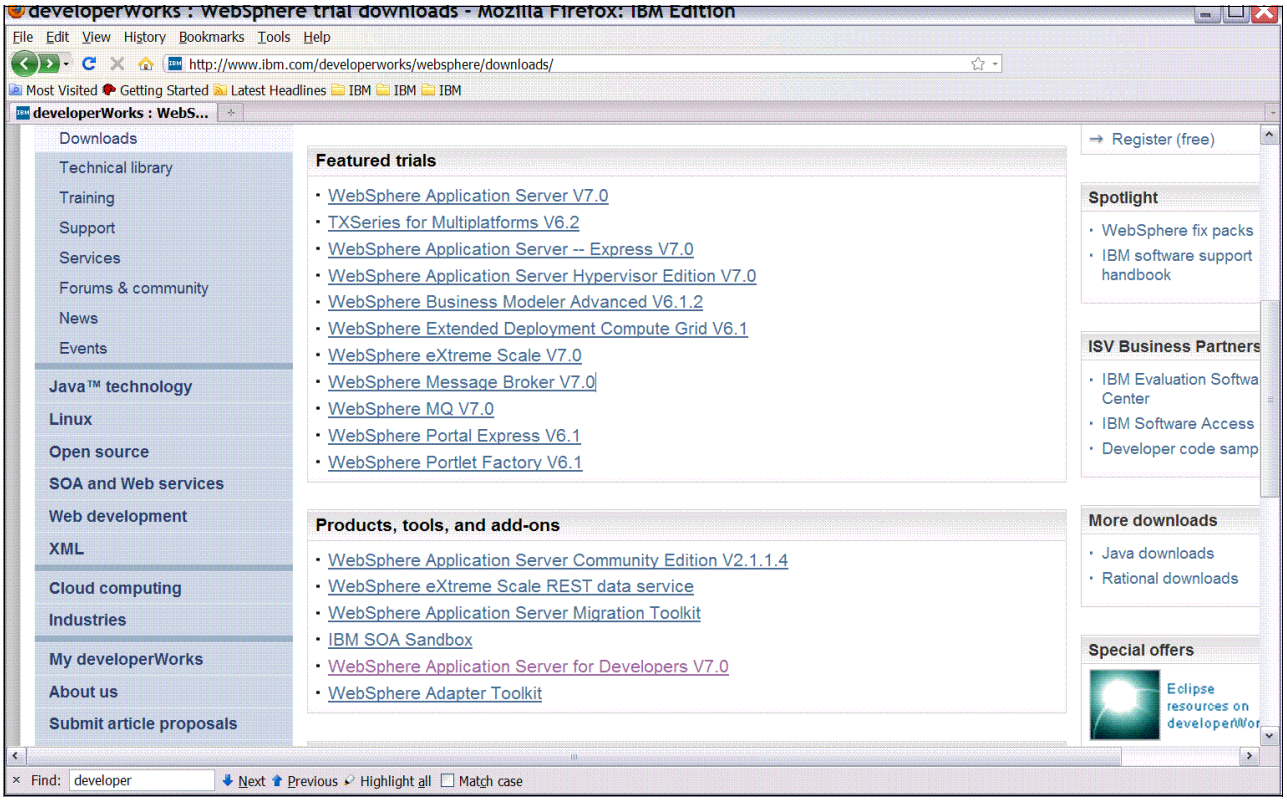


Figure 1-1 WebSphere downloads

2. Click **WebSphere Application Server for Developers V7.0**.
3. Click **Download now** in the Download section.
4. Log in with your IBM ID and password if you have an IBM ID, or click **Proceed without an IBM ID.**
5. Select **WebSphere Application Server for Developers V7.0**, and click **Continue**.
6. Complete all of the fields on the form on the next page, and click **I confirm**.
7. In Download using Download Director, select the `.zip` file, and click **Download now**. A Download Director window opens, where the File attribute is the file path to which your selected package is downloaded.

> **Note:** Download Director requires a plug-in that you might not have, so alternatively, you can choose the **Download using HTTP** tab, and click **Download now** next to the `.zip` file.

1.2 Installing WebSphere Application Server

The downloaded archive `.zip` file contains an executable file that launches the installer when run. Use the following steps to install WebSphere Application Server:

1. Extract the downloaded package.
2. In the extracted package folder, navigate to the `WAS` folder.
3. Double-click **Install.exe**.
4. Follow the steps provided by the installation wizard. Note the following considerations:
   - Although we are not using the sample applications that come with the product, installing the sample applications might be useful for you to test the deployment of various applications on your own later.
   - You can change the default installation directory so that it is easier to navigate to it.
   - You must create at least one profile to use WebSphere. You can create profiles through the installation wizard, or later. We show you how to create the profile after the installation, in case you want to create more than one profile later. So, choose **None** when you reach the WebSphere Application Server Environment window to not create the profile, and click **Yes** when prompted with a warning. We describe the various ways to create profiles next.

1.3 Creating profiles

A *profile* is the set of files that defines a single WebSphere Application Server runtime environment. You can configure multiple profiles on a single machine without having to perform a complete installation every time. You can create an initial profile during installation. To create a profile after installation or to create additional profiles, you can use either the command-line or GUI method.

Each profile has a profile name and a profile path. Profile names are unique within a WebSphere installation. The default path for a profile is in the *install_root*/profiles/*profile name* directory.

1.4 Using the command line

Example 1-1 shows you how to create a profile using the command-line tool.

Example 1-1 Create a profile using the command line

```
install root\bin\manageprofiles.bat -create -templatePath
install_root\profileTemplates\default -profileName profile_name -cellName
cell_name -nodeName node_name -hostName hostname -enableAdminSecurity true
-adminUserName user_name -adminPassword password -
samplesPassword samples_password -defaultPorts
```

We define the parameters that are used in **manageprofiles** command that is shown in Example 1-1:

create Creates the profile.

templatePath The path of the template that is used to create the new profile. For our environment, always use the default template.

profileName	The name of the profile.
cellName	The name of the cell. In WebSphere, a *cell* is a set of resources that are managed together, such as applications and application servers.
nodeName	The name of the node. In WebSphere, a node is a set of resources that are managed together on the same machine.
hostName	The name of the host machine.
enableAdminSecurity	Enables administrative security. Valid values include `true` or `false`. Always set this value to true to secure your application server.
adminUserName	The user ID that is used for administrative security.
adminPassword	The password for the administrative user ID.
samplesPassword	The password to use for samples. You use this password to restrict access to web application samples that are installed during the installation of the application server. The corresponding user name is `samples`.
defaultPorts	Use default port numbers for the ports on which the application server listens.

If you want to run the sample applications, you must specify a samples password when using the **`manageprofiles`** command to create a profile with administrative security enabled.

Try the following command to create a new profile, using the real path of *`install_root`* for your environment, and a user name and password of your choosing. We use this environment later to deploy the sample applications.

Example 1-2 Command-line example to create a profile

```
manageprofiles.bat -create -templatePath C:\was\profileTemplates\default
-profileName dev -cellName devCell -nodeName devNode -hostName localhost
-enableAdminSecurity true -adminUserName <user> -adminPassword <password>
-samplesPassword <password> -defaultPorts


INSTCONFSUCCESS: Success: Profile dev now exists. Please consult C:\was7000\pro
files\dev\logs\AboutThisProfile.txt for more information about this profile.
```

1.5 Using the GUI

The GUI interface offers an approach to create a profile that prompts you for information at each step. Use the following steps to create a profile with the Profile Management Tool:

1. Access the Profile Management Tool by going to **Start** → **All Programs** → **IBM WebSphere Application Server V7.0** → **Profile Management** → **Tool**, as shown in Figure 1-2 on page 5.

> **Note:** Alternatively, you can access the WebSphere Application Server install folder and run the following `.bat` file:
>
> *`install root`*`/bin/profileManagement/pmt.bat`

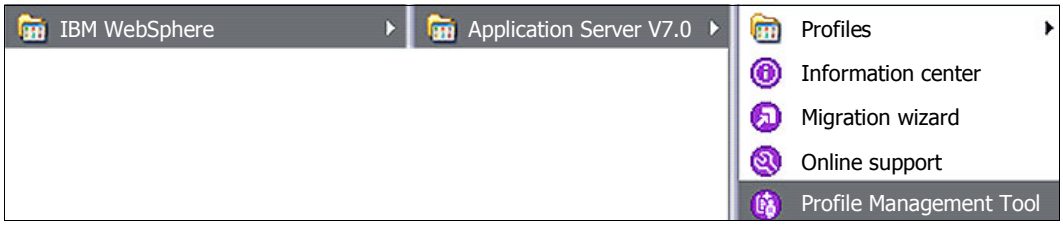


Figure 1-2 Accessing Profile Management Tool

2. Click **Launch Profile Management Tool**, or click **Profile Management Tool**, as shown in Figure 1-3.

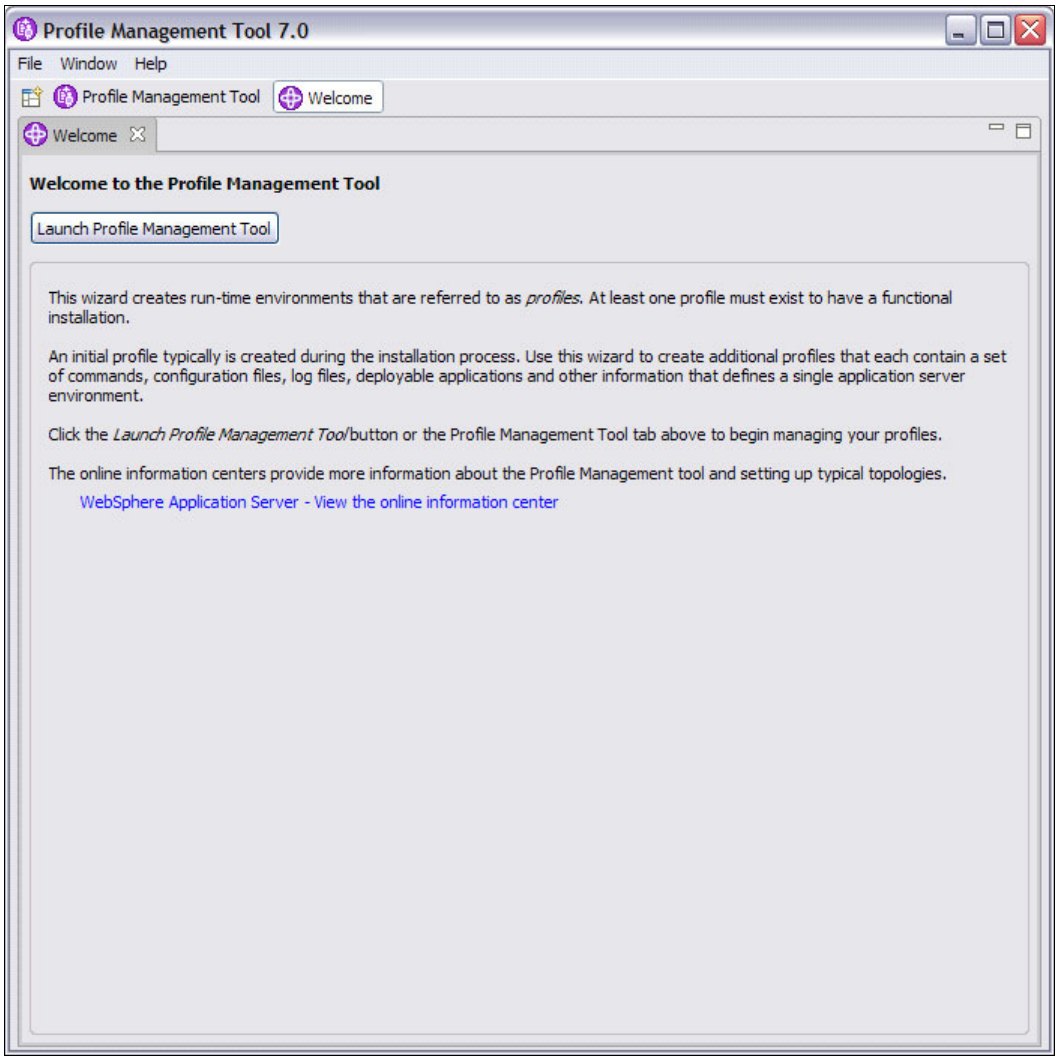


Figure 1-3 Profile Management Tool start menu

3. You see a listing of the profiles that are already created, as shown in Figure 1-4. Because you did not create a profile during installation, you do not see any profiles listed. Click **Create**.

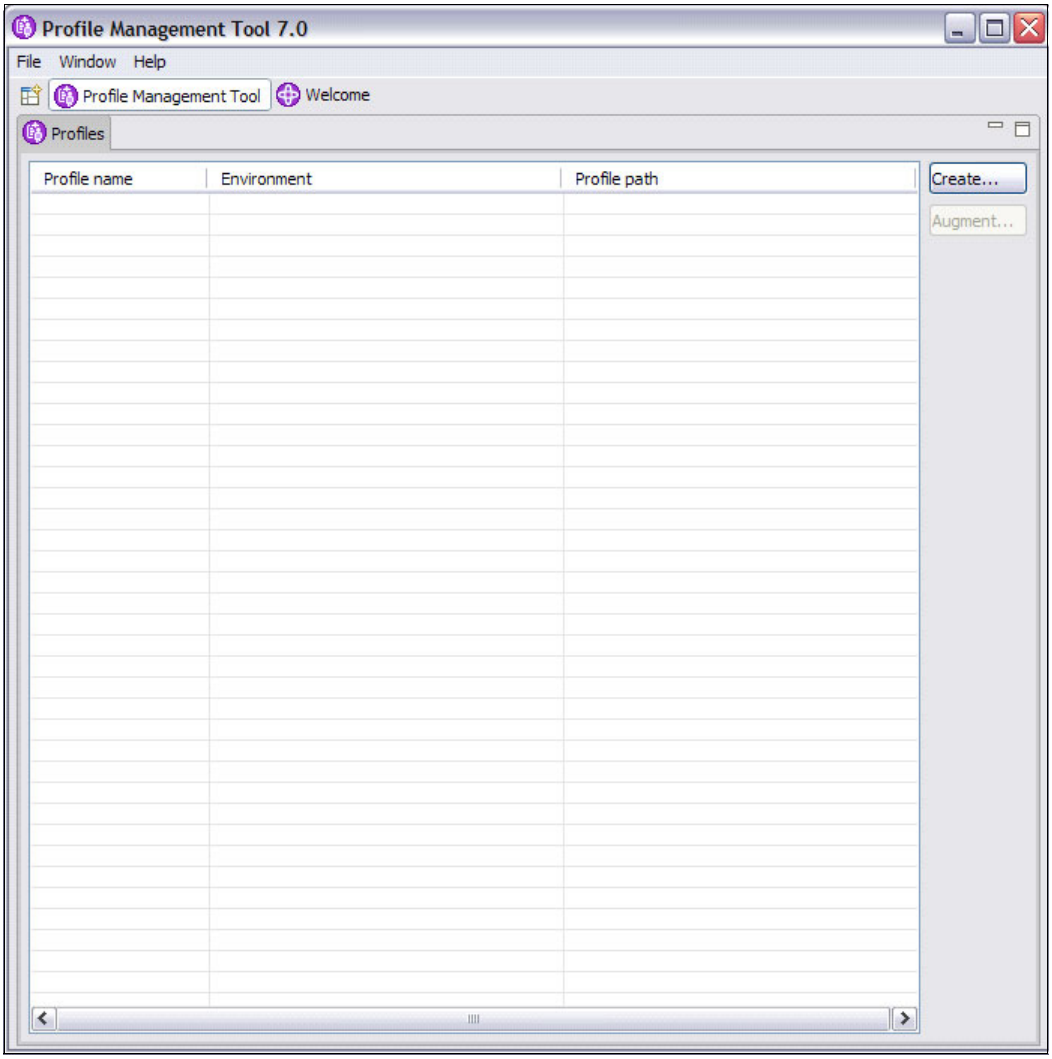


Figure 1-4 Profile Management Tool

4. Select the environment on which you want to create your profile. In this example, the environment is `Application server`, as shown in Figure 1-5. An Application server environment is an independent server that is able to run your application. The Management environment allows you to manage multiple application servers through an administrative server and other services. Click **Next**.

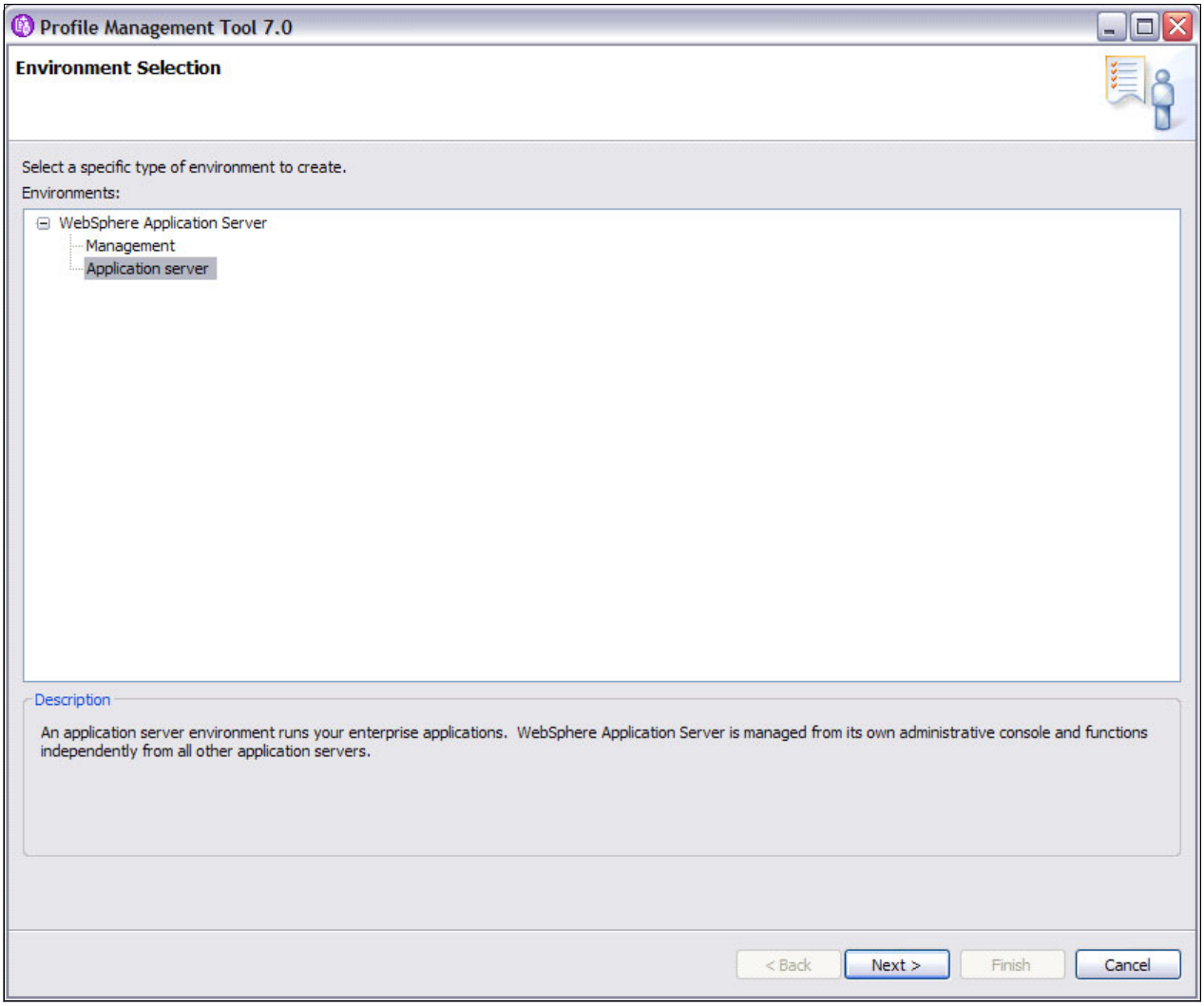


Figure 1-5 Profile creation wizard

5. Select the profile create option that you want. If you want to install the sample applications or you choose to not run your application server as a Microsoft Windows service, choose Advanced Profile Creation. Otherwise, choose Typical Profile Creation. In our example, we choose **Advanced Profile Creation**, as shown in Figure 6 on page 9. Click **Next**.

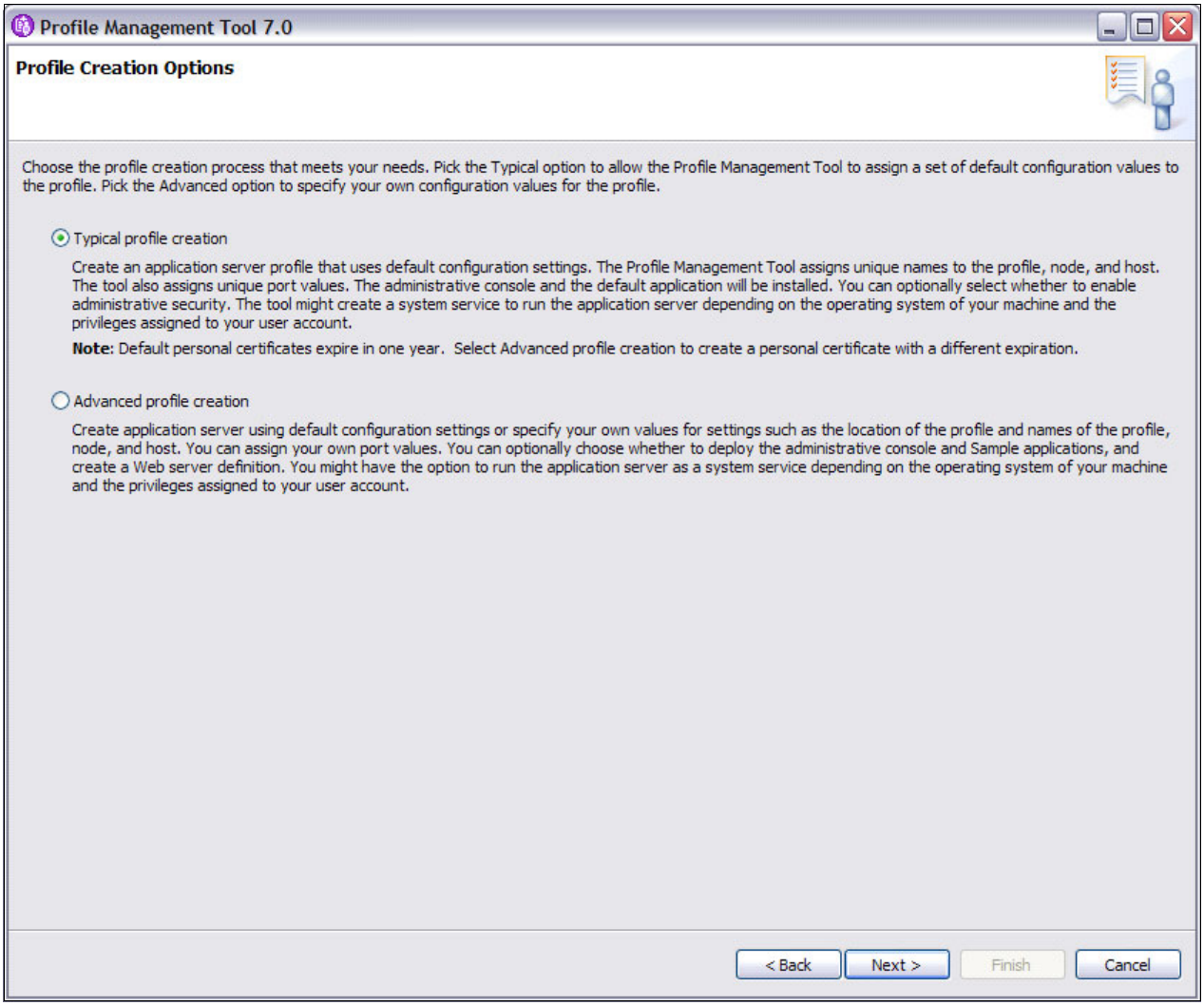


Figure 1-6 Profile Creation Options

6. Ensure that **Deploy the administrative console (recommended)** is selected, as shown in Figure 7 on page 10. Click **Next**.

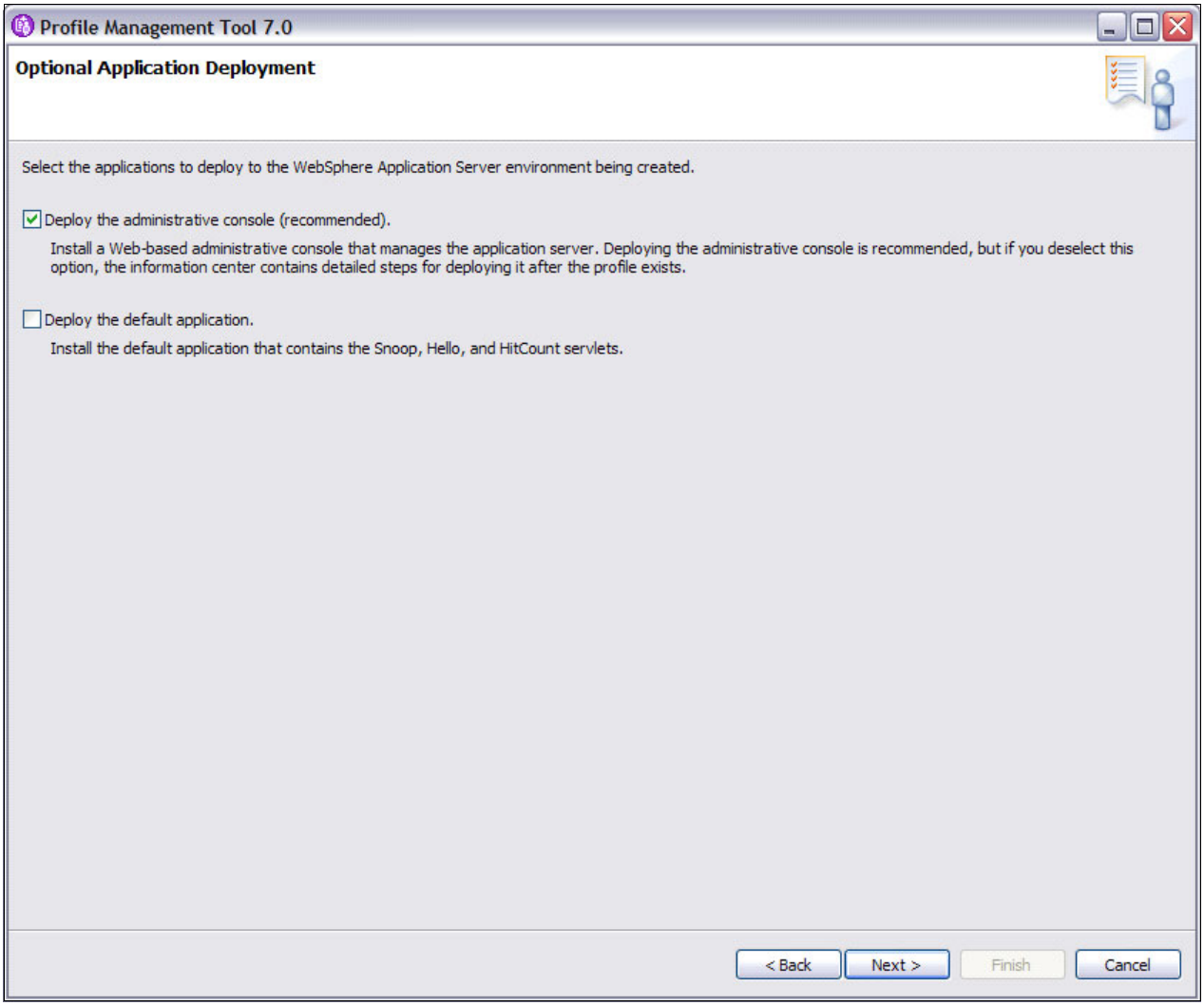


Figure 1-7 Profile creation Optional Application Deployment

7. Under Profile name, enter `dev`. For the Profile directory, enter *`install_root`*`\profiles\dev`, as shown in Figure 8 on page 11. Click **Next**.

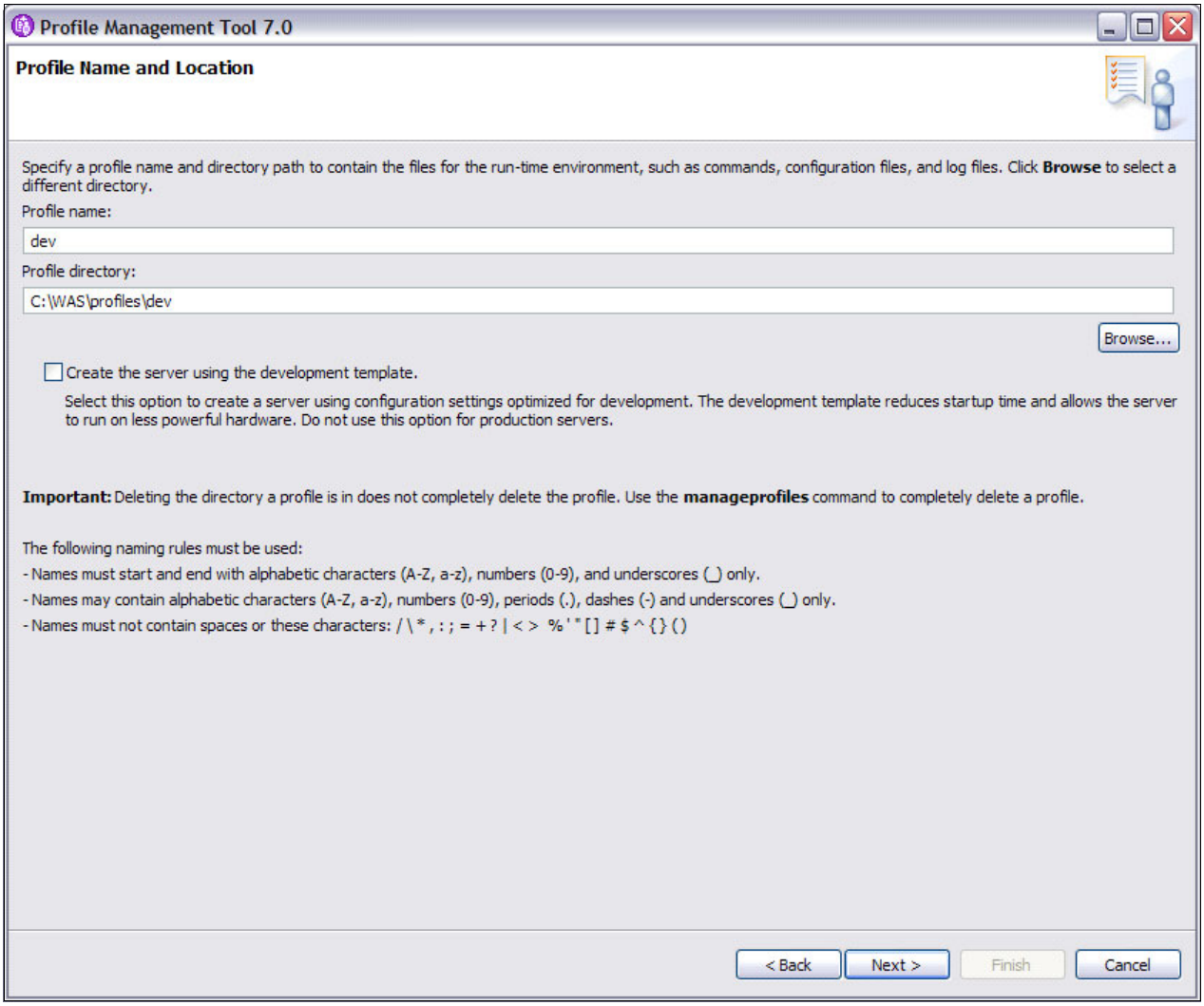


Figure 1-8 Profile creation Name and Location

8. Fill out the fields, as shown in Figure 1-9. Click **Next**.

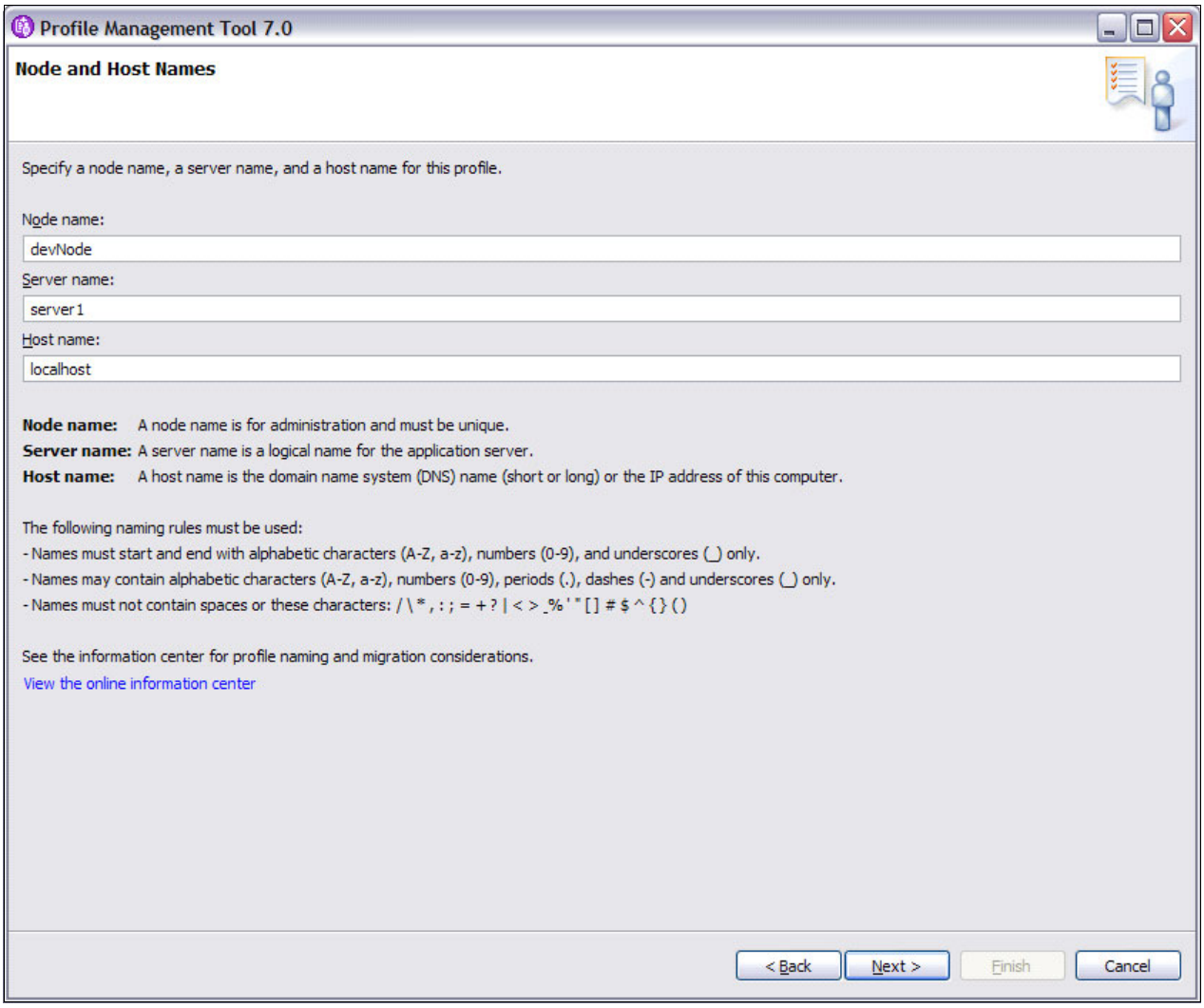


Figure 1-9 Profile creation Node and Host Names

9. Manage your security settings. To enable administrative security, select **Enable Administrative Security** and add a user name and password of your choice, as shown in Figure 1-10. Click **Next**.

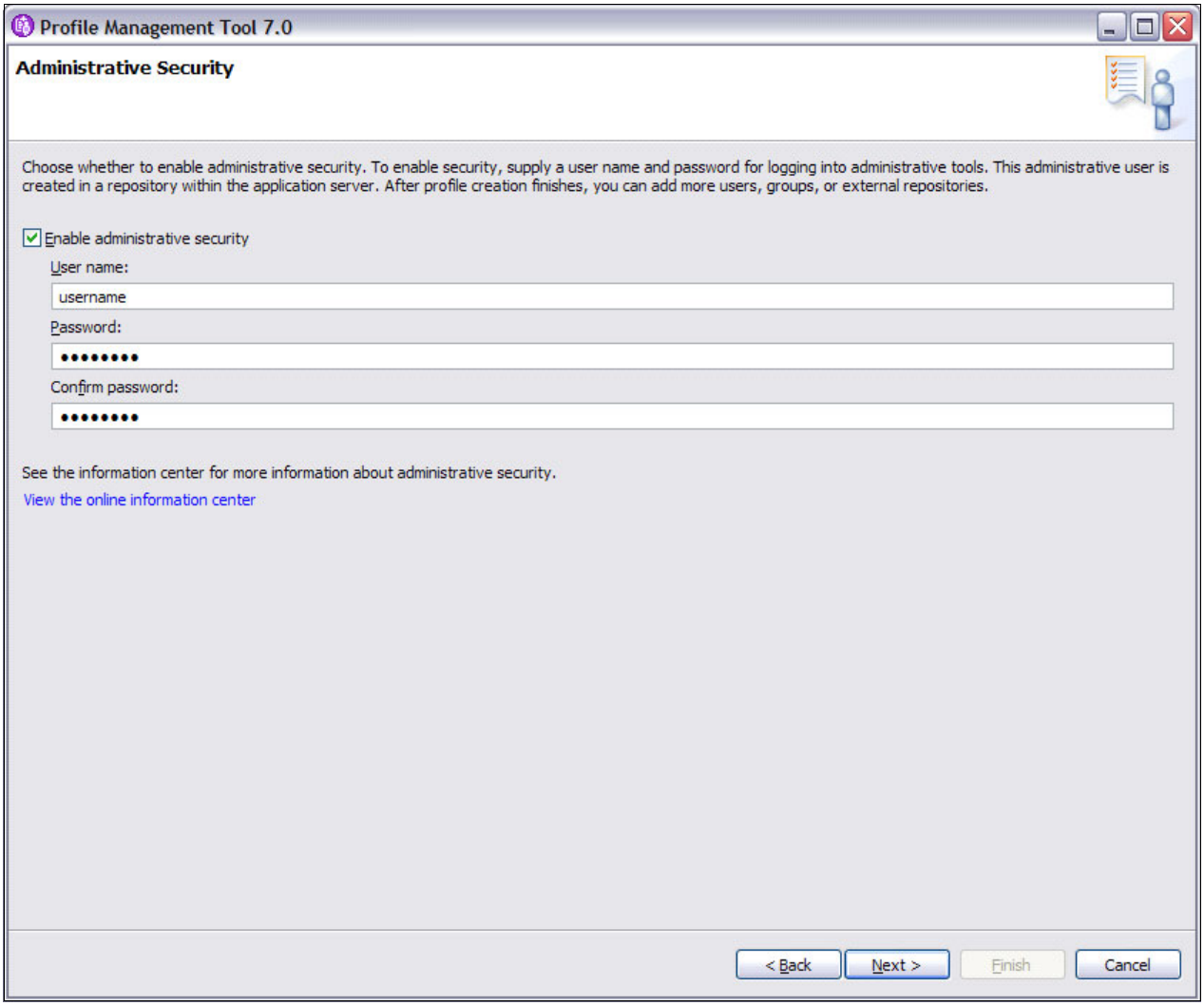


Figure 1-10 Profile creation Administrative Security

10. You can choose whether to create a new security certificate or import it from existing keystores. For the `dev` profile, accept the default values, as shown in Figure 1-11. Click **Next**.

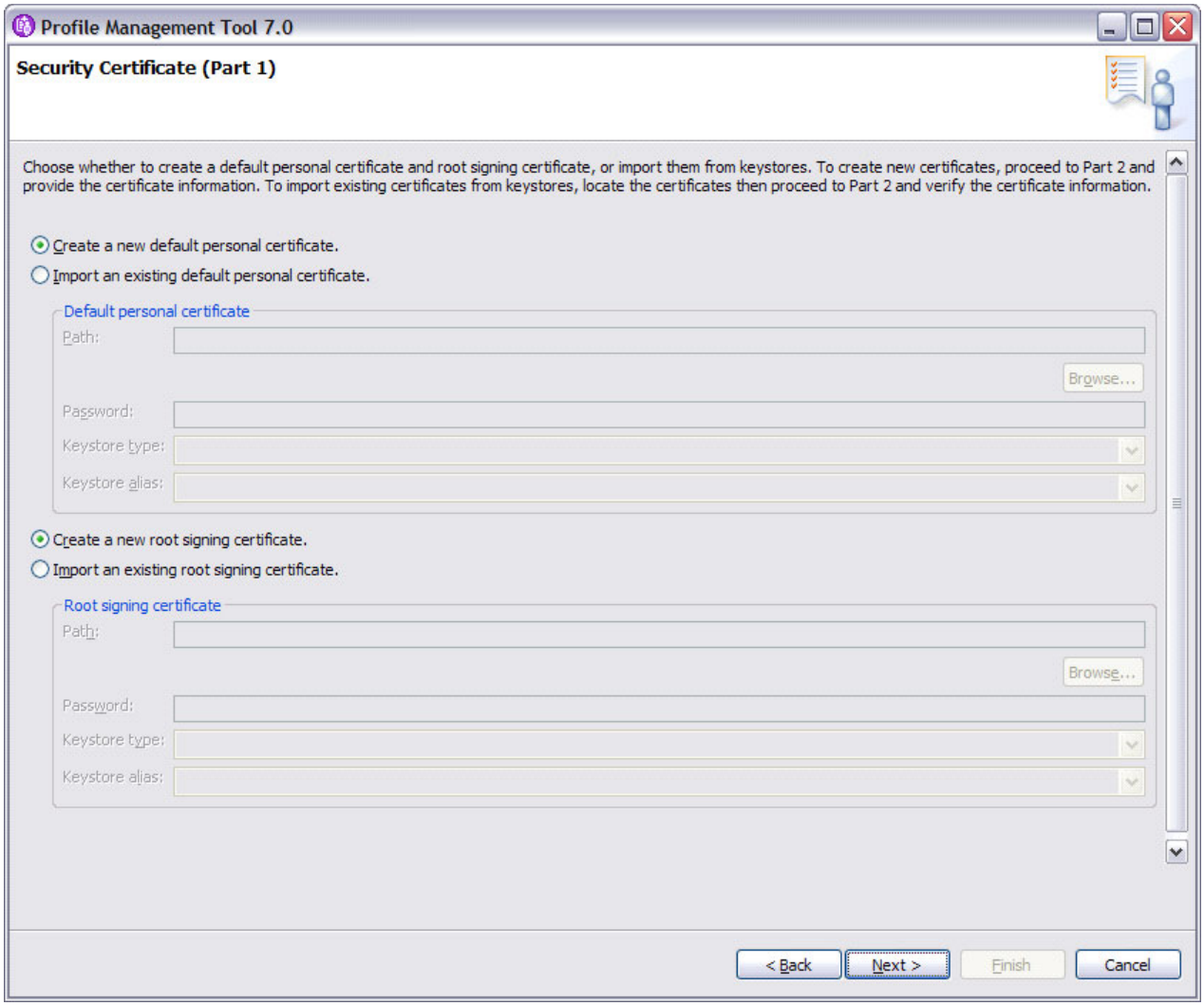


Figure 1-11 Profile creation Security Certificate (Part 1)

11. Configure the security certificate further, as shown in Figure 1-12. For the `dev` profile, accept the defaults. Click **Next**.

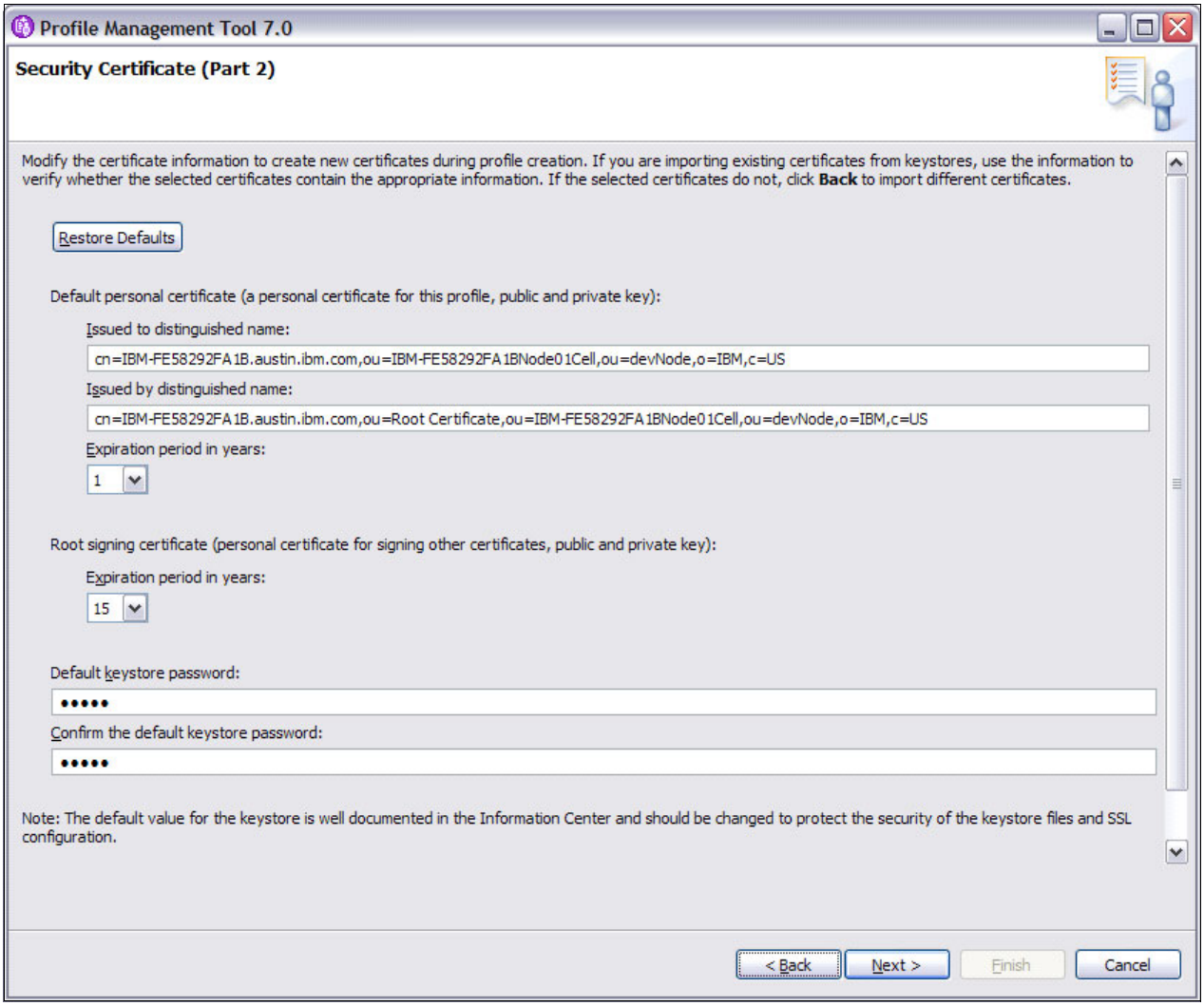


Figure 1-12 Profile creation Security Certificate (Part 2)

12. Click **Default Port Values** to use the default ports, as shown in Figure 13 on page 16. Click **Next**.

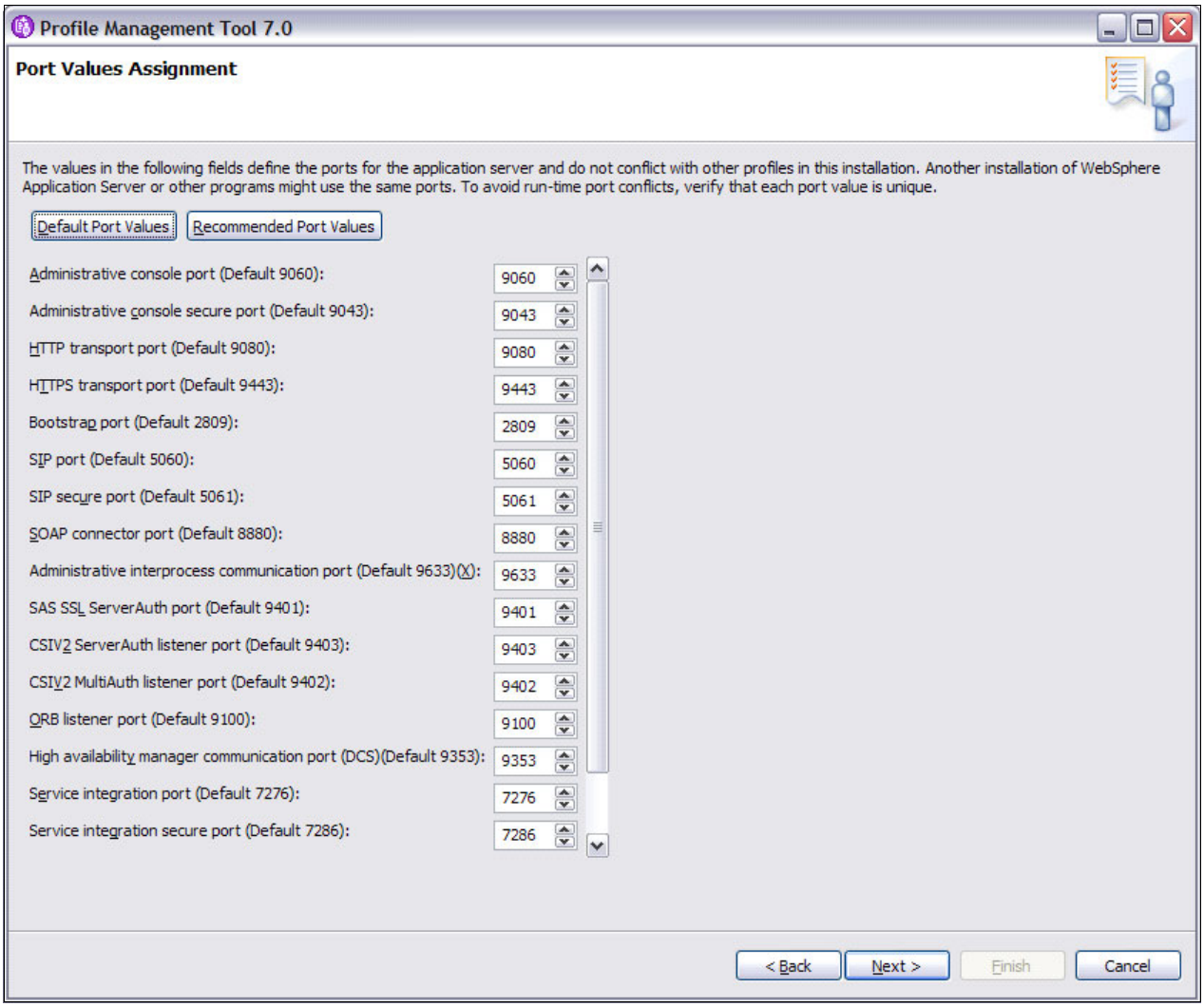


Figure 1-13 Profile creation Port Values Assignment

13. You can choose whether to run your application server process as a Windows service. You can clear the “Run the application server process as a Windows service” check box if you do not want Windows Service to monitor your application server, as shown in Figure 1-14. Click **Next**.

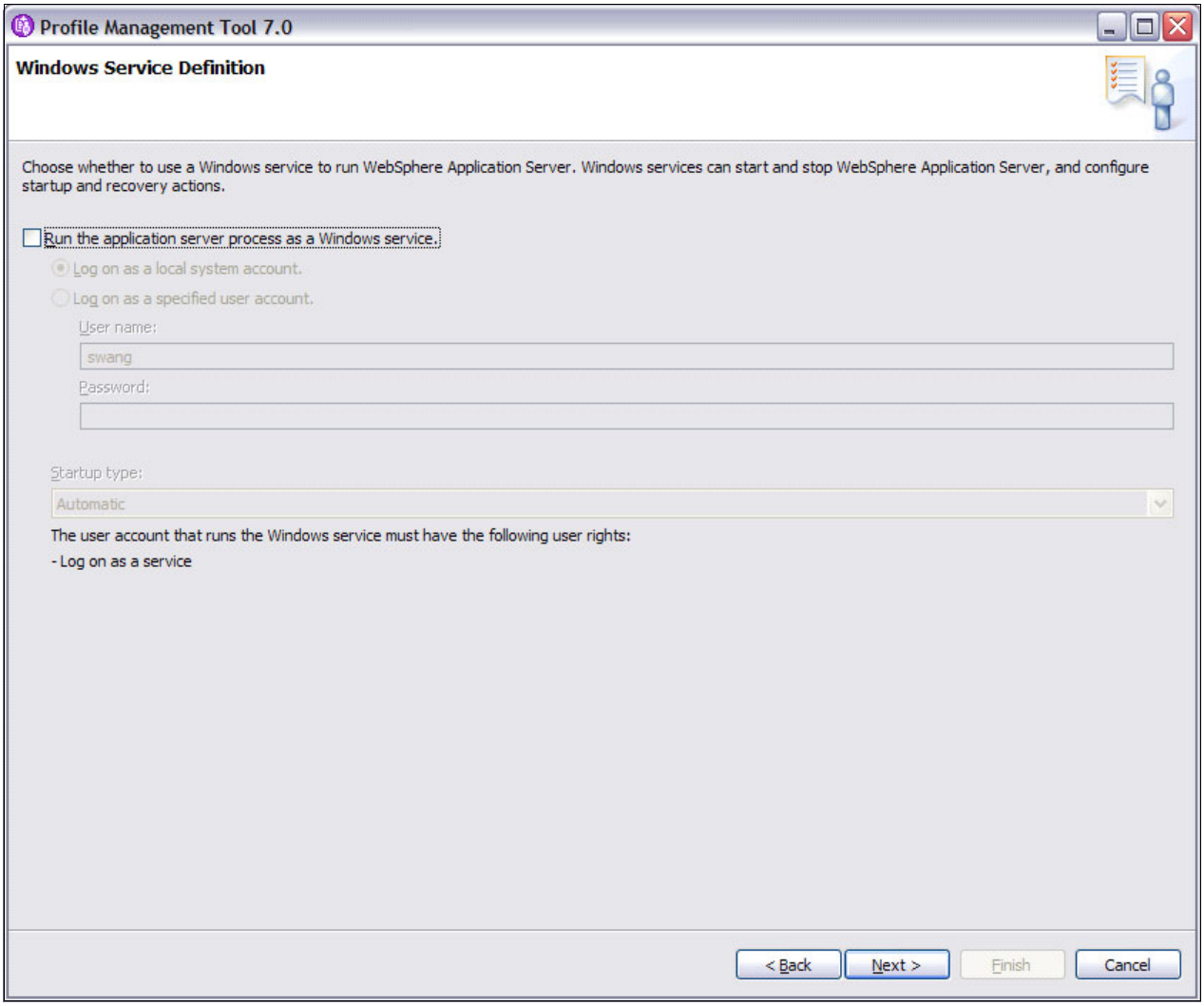


Figure 1-14 Profile creation Windows Service Definition

14. You can choose to create a web server definition. For the `dev` profile, clear the “Create a Web Server definition” check box, as shown in Figure 15 on page 18. Click **Next**.

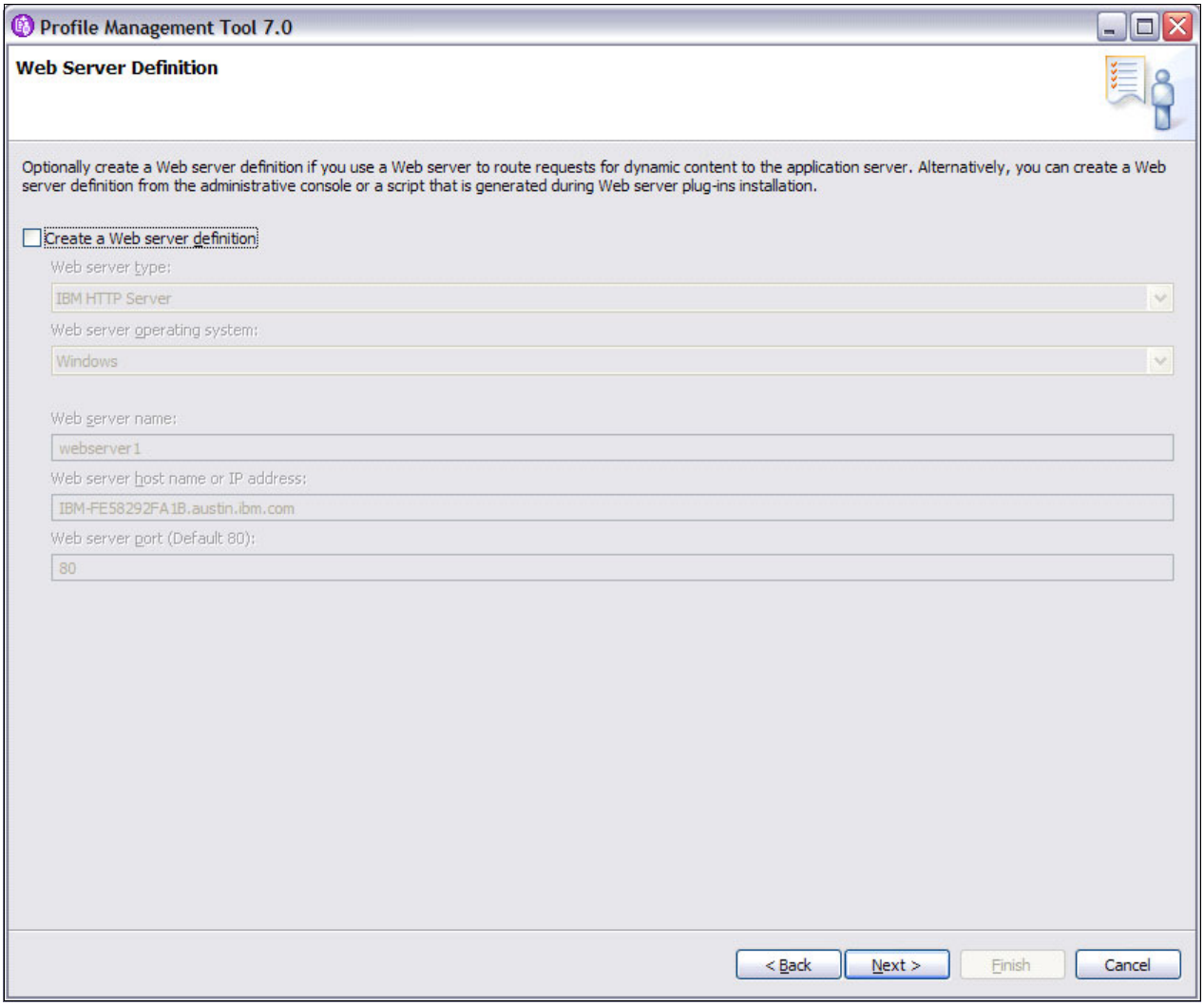


Figure 1-15 Profile creation Web Server Definition

15. In the Summary page, as shown in Figure 1-16, review your selections and click **Create** to create the profile.

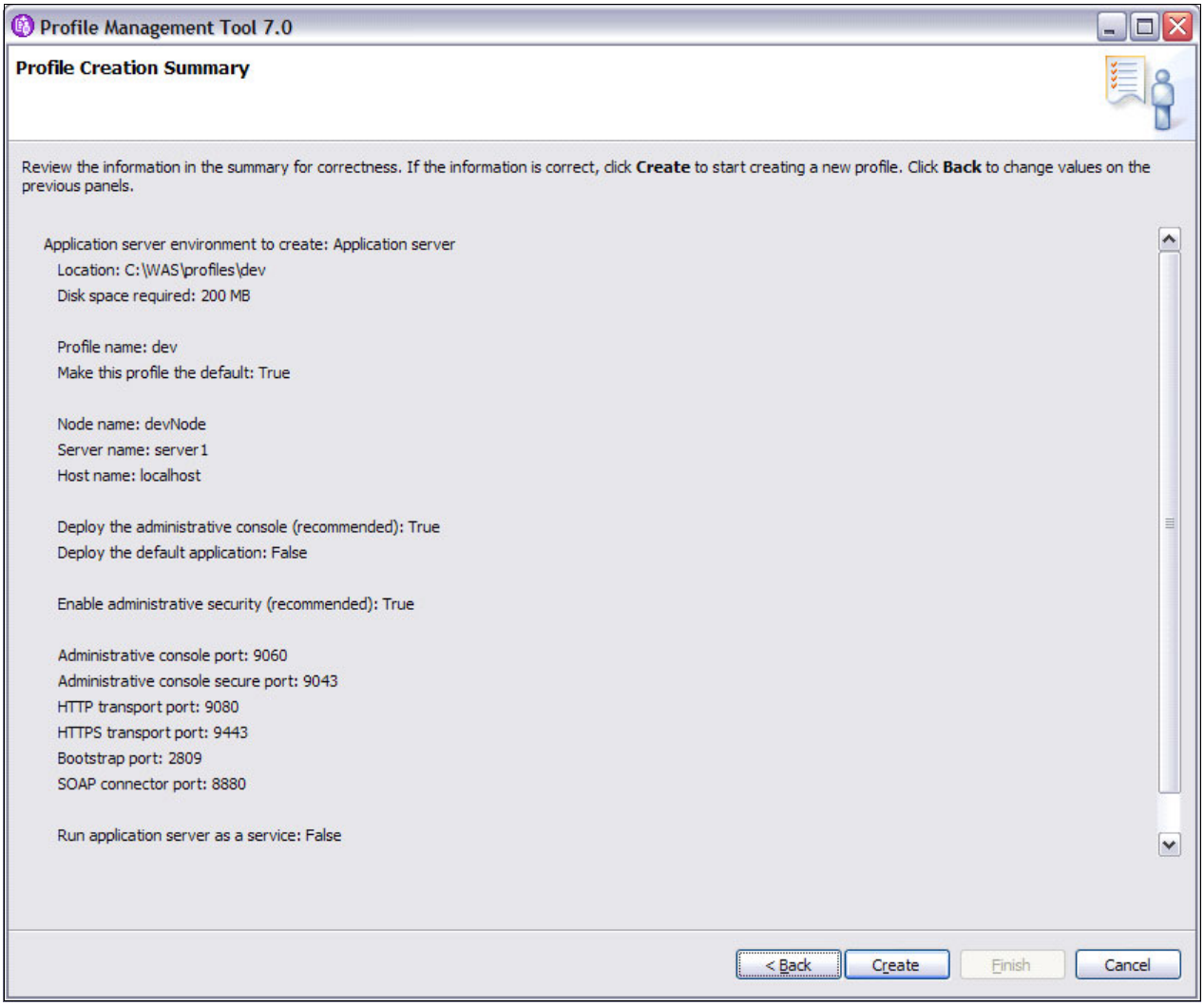


Figure 1-16 Profile Creation Summary

1.6 Verifying the installations

Follow these steps to verify the installations:

1. In Microsoft Windows, go to **Start Menu** → **All Programs** → **IBM WebSphere** → **Application Server V7.0** → **profiles** → **dev** → **First Steps**.
2. Click **Installation verification**.
3. Monitor the progress in the **Installation Verification** window and ensure that it reports success at the end.

> **Application server:** The install verification leaves the application server running after it completes. See 2.5, “Command-line tools” on page 27 for information about using command-line tools to stop or start the application server.

1.7 Development environment

We set up our development environment in the following manner:

install root	The root of the product install directory
profile root	The root of the profile's directory, which is <*install_root*>/profiles/dev
profile name	dev
cell name	devCell
node name	devNode
server name	server1
host name	localhost

We created this profile by using the **manageprofiles** command-line tool. If you used only the profile creation GUI tool to create the profile, you cannot set the cell name to devCell. Instead, the cell name is similar to *hostName*Cell01, for example, localhostCell01. Be aware of this difference when you see devCell used elsewhere in this document.

2

WebSphere administration

This chapter provides information about the WebSphere administration concepts and tools that you need to help you manage your development environment. If you are a first-time WebSphere user, you need to read all of the sections in this chapter up to and including the administrative console section before proceeding to subsequent chapters.

We start this chapter with an overview of the administration architecture, including its building blocks based on Java Management Extensions (JMX) infrastructure. We follow the overview with an introduction to the WebSphere Information Center, which is the location for WebSphere reference materials. We then take you through the structure of the application server configuration files. We follow the structure of the application server configuration files with a tutorial for the tools that you need to manage the development environment, including command-line tools, the administrative console, and the **wsadmin** scripting tool. Finally, we describe the various log files that are generated by the application server, because you need them for debugging.

© Copyright IBM Corp. 2010. All rights reserved.

2.1 Overview of WebSphere administration architecture

Figure 2-1 provides an illustration of the basic architecture of WebSphere administration.

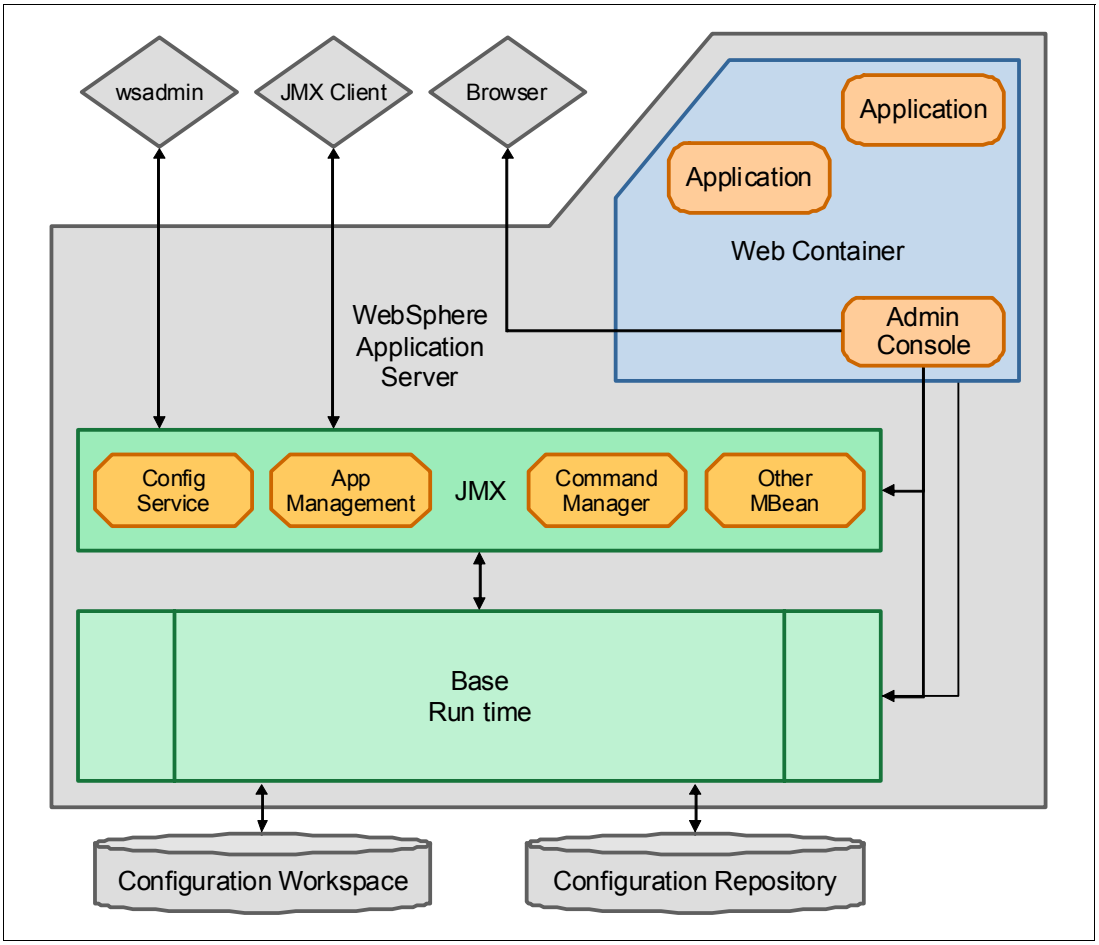


Figure 2-1 WebSphere administration architecture

The WebSphere administration architecture is built on the standard JMX infrastructure. Command-line tools, such as **`stopServer`**, **`startServer`**, and **`serverStatus`**, use the Java JMX API to communicate with the application server. The **`wsadmin`** scripting tool allows you to use scripting languages to manage WebSphere through JMX. The built-in administrative console offers a browser-based user interface for management. It also uses JMX transparently. You can also write your own Java program that uses JMX to manage WebSphere.

The JMX architecture defines management beans, which are also known as *MBeans*, as manageable resources. WebSphere provides a built-in set of MBeans, each of which is for a separate management operation. The ConfigService MBean manages WebSphere configurations. The AppManagement MBean performs application installation-related tasks. The CommandManager MBean calls built-in task-oriented commands that are designed to simplify administration by linking multiple configuration steps together. You can use other MBeans for these functions:

- Start or stop applications
- Query the state of the Java virtual machine (JVM)
- Generate java core or heap dumps
- Manage thread and database connection pools

- Change trace settings
- Gather statistics

MBeans interact with the base administration run time. The services that are offered by the base run time include the configuration repository service to access the configuration files, the security service to authenticate and authorize users, and the workspace service that offers a temporary scratch pad to store unsaved configuration changes.

2.2 WebSphere Application Server Information Center

You can access the WebSphere Application Server Version 7.0 Information Center at the following website:

`http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp`

On the left side of the page is the navigation tree that allows you to access information about various editions of WebSphere Application Server. The edition that best matches the developer edition is **WebSphere Application Server (Distributed platforms and Windows), Version 7.0**, as shown in the list on Figure 2-2.

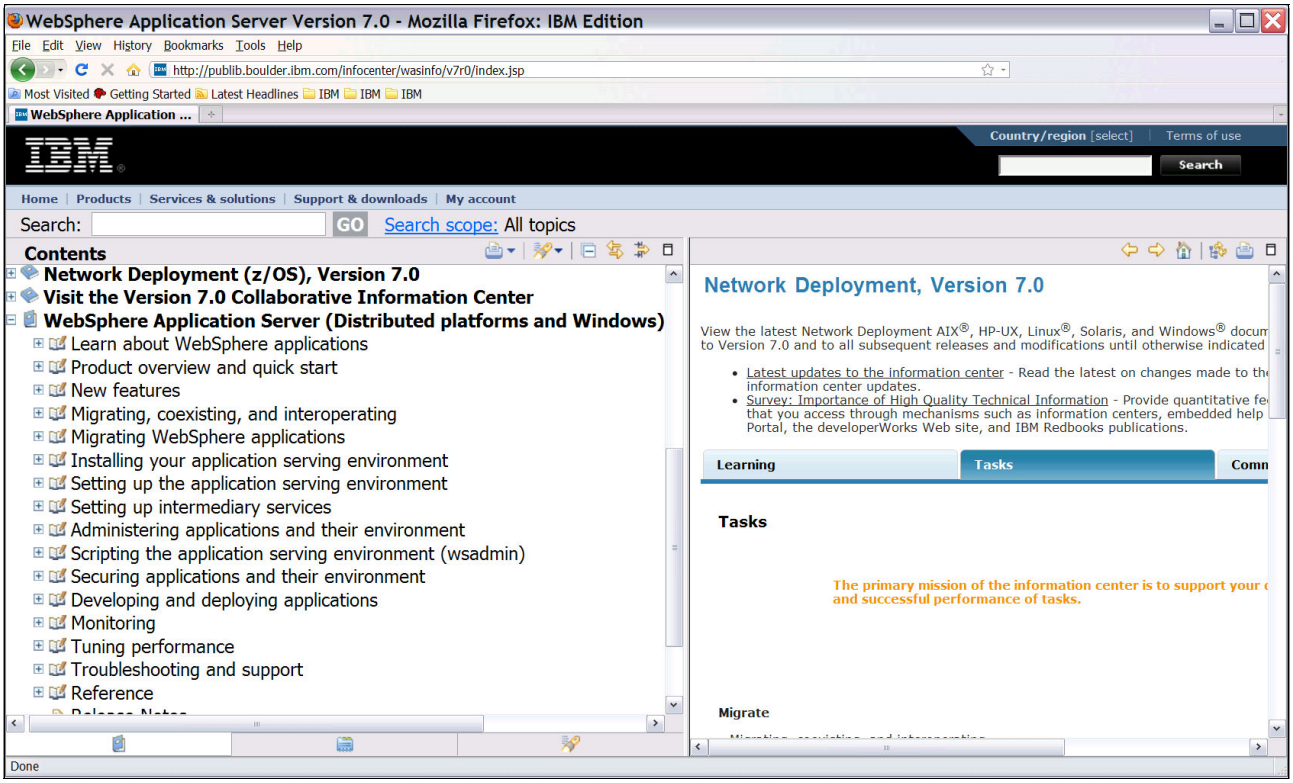


Figure 2-2 WebSphere Application Server Version 7.0 Information Center

We provide pointers to various sections of the information center in the rest of this chapter where relevant. For now, you can navigate to the sections in the information center to familiarize yourself with the layout. As you become more familiar with WebSphere, you can use the Search box at the top to search for specific information.

2.3 WebSphere configuration repository

For a developer familiar with XML and object relationship models, the easiest way to become familiar with WebSphere configuration is by browsing its configuration repository. The path to the WebSphere configuration repository is <*profile_root*>/config. Use your favorite file browser to explore the contents of this directory and its sub-directories. These configuration files determine how the application server behaves when it is started. They include security settings, JVM settings, initial thread pool and connection pool settings, and which applications to start.

Note that even though we are in a single server environment, the directory structure is designed for a multi-server, multi-node environment. This environment allows our single server environment to share the same directory structure as the Network Deployment version of WebSphere that supports multiple nodes and servers.

Focus on these subdirectories:

- cells/devCell

  The root of the cell configuration. A *cell* encompasses the set of resources, such as nodes, applications, and servers, that can be managed together.

- cells/devCell/applications

  The root of where the installed applications are stored.

- cells/devCell/nodes

  The root for node-level configuration files. For our environment, there is only one node.

- cells/devCell/nodes/devNode

  The root directory of our node is devNode.

- cells/devCell/nodes/devNode/servers

  The root directory for all servers under devNode. For our environment, there is only one server.

- cells/devCell/nodes/devNode/servers/server1

  The root directory for the configuration files for server1.

When you browse the directories, you see various types of files: XML configuration files, application binaries, and Secure Sockets Layer (SSL) key and truststore files that end in .p12. For the XML files, use your favorite XML viewer to view them. For those XML files that contain xmi:id attributes, which are the majority of the files, note the hierarchy of the elements within the files. This hierarchy follows the WebSphere configuration model.

Note: WebSphere does not support the direct edit of its configuration XML files, because direct editing might collide with changes being made concurrently through its administrative console or scripting interfaces.

2.3.1 Scoping

When you browse the directories, you notice that the same named files appear under multiple directories, for example:

- *profile_root*/config/cells/devCell/variables.xml
- *profile_root*/config/cells/devCell/nodes/devNode/variables.xml
- *profile_root*/config/cells/devCell/nodes/devNode/servers/server1/variables.xml

The directories under which a file can appear are defined by its allowed scope. For `variables.xml`, the allowed scopes are cell, node, or server. In general, configurations defined at a lower scope override configurations defined at a higher scope, for example, server scope overrides node scope, which overrides cell scope.

2.3.2 variables.xml

The `variables.xml` file defines a set of name to value variable substitutions. By defining environment-specific values in `variables.xml`, you can restrict environment-specific values in the file. You can then use the variable names elsewhere in the configuration so that they are environment neutral. Note the following variables in *<profile_root>*`/config/cells/devCell/nodes/devNode/variables.xml`:

WAS_INSTALL_ROOT	Where WebSphere is installed.
USER_INSTALL_ROOT	Where the profile is stored.
DERBY_JDBC_DRIVER_PATH	Where the Derby Java Database Connectivity (JDBC) driver is stored.
APP_INSTALL_ROOT	Where the deployed applications are expanded and read by the run time.

2.3.3 The resources.xml file

Information about resources, such as JDBC providers and data sources, is stored in `resources.xml`. You can find the resources under these directories:

- *profile_root*`/config/cells/devCell/resources.xml`
- *profile_root*`/config/cells/devCell/nodes/devNode/resources.xml`
- *profile_root*`/config/cells/devCell/nodes/devNode/servers/server1/resources.xml`

Each `resources.xml` file is defined under cell, node, or server scope. Resources defined at a lower scope override resources defined at a higher scope. For our single server environment, this resource information does not really matter. For a multiple-server, multiple-node environment, a resource defined at the cell scope can be shared by all servers. A resource defined at the node scope can be shared by all servers under that node.

You can define a resource at the application scope to override all other scopes. However, application-scoped resource is currently only available through Rational® Application Developer.

Next, open the file named *profile_root*`/config/cells/devCell/nodes/devNode/servers/server1/resources.xml` and familiarize yourself with the attributes and nested elements for the Derby JDBC provider. Note how datasources are defined. Look at DefaultDataSource, which is one of the definitions.

2.3.4 The serverindex.xml file, installed applications, and ports

The `serverindex.xml` file contains information about the ports and the applications for the servers that are defined under a node. Open the file named *profile_root*`/config/cells/devCell/nodes/devNode/serverindex.xml` and familiarize yourself with its contents. Look at the deployedApplications elements to determine which applications have been installed. Look at the specialEndPoints elements for the port numbers assigned to various ports.

Note the following ports:

SOAP_CONNECTOR_ADDRESS The port you use when using **wsadmin** through the SOAP connector, the default

WC_adminhost The port you use to access the administrative console

WC_defaulthost The port you use to access your applications

2.3.5 The server.xml file

The `server.xml` file contains the settings for the JVM, JMX connectors, thread pools, web container, and Enterprise JavaBeans (EJB) container settings. Open *profile_root*`/config/cells/devCell/nodes/devNode/servers/server1/server.xml`and familiarize yourself with its contents. Look for the element jvmEntries, which corresponds to the settings for the JVM. Note the settings for the attributes verboseGarbageCollection and debugMode. Also, note the potential absence of the initialHeapSize and maximumHeapSize attributes. If these attributes are not set in the configuration file, the server uses built-in default values, which are 50 Mb and 256 Mb.

2.4 Application-related files

When you deploy a Java Enterprise Edition (JEE) application to WebSphere, the application is stored in the `config/cells/devCell/applications` directory. To see an example, go to the *profile_root*`/config/cells/devCells/applications/DefaultApplication.ear` directory. The `DefaultApplication.ear` file that was input to application deployment is stored here. You need additional files to enable the application to run in the WebSphere environment. Open *profile_root*`/config/cells/devCell/applications/DefaultApplication.ear/deployments/DefaultApplication/deployment.xml`and look for these names:

binariesURL Where the application is extracted so that the application server run time can read the contents to run the application. Note the use of variables $(*APP_INSTALL_ROOT*) and $(*CELL*) in the path. You can find the values in one of the `variables.xml` files.

classloader How WebSphere loads classes. You can obtain more information about class loaders in the WebSphere Application Server V7.0 Information Center.

targetMappings The application server that a module is to run. For our environment, there is only one application server.

During application deployment, WebSphere also generates a set of binding files that map the resource references in the deployment descriptor or annotations in the application modules to corresponding WebSphere artifacts. Open the file named *profile_root*`/config/cells/devCell/applications/DefaultApplication.ear/deployments/DefaultApplication/DefaultWebApplication.war/WEB-INF/ibm-web-bnd.xml`. Note how it specifies that the ejb reference `EjbRef_1` in the web module maps to an EJB whose Java Naming and Directory Interface (JNDI) name is `Increment`.

Open the file that is named *profile_root*`/config/cells/devCell/applications/DefaultApplication.ear/deployments/DefaultApplication/Increment.jar/META-INF/ibm-ejb-jar-bnd.xmi`. Note the ejbBindings element that specifies the JNDI name for the EJB as `Increment`. Also, note the defaultCMPConnectionfactory that uses the DefaultDataSource data source.

2.4.1 Configuration compared to runtime changes

Several of the MBean operations change the files in the configuration repository. These changes are categorized as configuration changes. Other MBean operations only affect the in-memory state of the application server. These changes are categorized as runtime changes. Configuration changes are made within the scope of a session. Whenever you log in to the administrative console, a session is used to track your changes. The intermediate changes are stored in a temporary workspace. When you use the **wsadmin** scripting command to connect to the application server, a new session and its associated temporary workspace are created for the duration of **wsadmin**.

Configuration changes are not saved to the master configuration repository until you explicitly save your changes, either through the administrative console or through **wsadmin**. The temporary workspace files are stored in the *profile_root*/wstemp directory, with a separate subdirectory created for each session. The **wsadmin** session names start with the prefix Script. The administrative console sessions are digits, for example, 111578566.

In general, the operations of ConfigService, AppManagement, and CommandManager MBeans are used to make configuration changes. After saving the configuration, you need to restart the application server for the run time to pick up the changes. One exception is for applications. If you only deploy or update an application, the run time is able to load it without having to restart the application server.

2.5 Command-line tools

Command-line tools reside in the *profile_root*/bin directory. The tools that you use frequently are **startServer**, **stopServer**, **serverStatus**, and **versionInfo**.

2.5.1 The startServer command

Use the **startServer** command to start an application server. Use this syntax:

```
startServer server [options]
```

Our server name is server1. The following options are the most useful options:

-help Print available options.

-trace Enables tracing of the **startServer** command itself. The trace can be found at *profile_root*/logs/server1/startServer.log.

-script Generates the start_server1 batch file that you can call directly to start the server. It is useful for debugging initial JVM start issues before it has a chance to write to log files.

Example 2-1 shows sample output from the **startServer** command.

Example 2-1 The startServer command output

```
startServer server1
ADMU0116I: Tool information is being logged in file
           C:\was\profiles\dev\logs\server1\startServer.log
ADMU0128I: Starting tool with the dev profile
ADMU3100I: Reading configuration for server: server1
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server server1 open for e-business; process id is 2512
```

You can find the process ID of the server in the following places:

- The output of the **startServer** command
- *profile_root*/logs/server1/startServer.log
- *profile_root*/logs/server1/SystemOut.log

2.5.2 The stopServer command

Use the **stopServer** command to stop an application server. Use this syntax:

stopServer server [options]

Our server name is server1. The following options are the most useful options:

-help	Print usage information.
-trace	Print a trace of the command.
-username	The user name to access the application server.
-password	The password needed to access the application server.

Note: Sometimes, the server can stop to the extent that you are unable to connect to it through JMX. This situation happens rarely, but when it happens, the **stopServer** command reports that it is unable to contact the server. You need to use an operating system tool, such as Microsoft Windows task manager, to terminate the server.

Example 2-2 shows sample output from the **stopServer** command.

Example 2-2 The stopServer command output

```
stopServer server1 -user <user> -password <password>
ADMU0116I: Tool information is being logged in file
           C:\was\profiles\dev\logs\server1\stopServer.log
ADMU0128I: Starting tool with the dev profile
ADMU3100I: Reading configuration for server: server1
ADMU3201I: Server stop request issued. Waiting for stop status.
ADMU4000I: Server server1 stop completed.
```

2.5.3 serverStatus

Use the **serverStatus** command to check the status of an application server. Use this syntax:

serverStatus *server* [options]

Our server name is `server1`. The following options are the most useful options:

-help Print usage information.

-username The user name to access the application server.

-password The password needed to access the application server.

Note: Sometimes the server might stop to the extent that you are unable to connect to it through JMX. This situation happens rarely, but when it does happen, the **`serverStatus`** command reports that it is unable to contact the server.

Example 2-3 shows sample output from the **`serverStatus`** command.

Example 2-3 The serverStatus command output

```
serverStatus.bat server1 -user <user> -password <password>
ADMU0116I: Tool information is being logged in file
           C:\was\profiles\dev\logs\server1\serverStatus.log
ADMU0128I: Starting tool with the dev profile
ADMU0500I: Retrieving server status for server1
ADMU0508I: The Application Server "server1" is STARTED
```

2.5.4 The versionInfo command

Use the **`versionInfo`** command to display your version of WebSphere. Example 2-4 shows sample output from the **`versionInfo`** command.

Example 2-4 The versionInfo command output

```
Installed Product
------------------------------------------------------------------------
Name                        IBM WebSphere Application Server
Version                     7.0.0.0
ID                          BASE
Build Level                 r0835.03
Build Date                  8/31/08
```

The version number is encoded as *v.w.x.y,* where *v* is the major version, *w* is the minor version, and *x.y* are the patch levels. For example, `7.0.0.0` is the first version of WebSphere V7, and `7.0.0.13` is 13th patch level for V7.

2.6 Administrative console

The WebSphere administrative console offers a graphical web-based administrative interface for you to use to configure and manage the resources within the scope of the console. The administrative console is the most user friendly administrative tool for getting started with WebSphere.

2.6.1 Logging in to the administrative console

After starting the application server, open a browser and type the following URL:

`http://localhost:9060/ibm/console`

Type the user ID and password to log in to the administrative console.

You can also access the administrative console by using the Microsoft Windows Start menu: **Start** → **All Programs** → **IBM WebSphere** → **Application Server V7.0** → **Profiles** → **dev**.

2.6.2 Administrative console layout

Figure 2-3 shows the administrative console after you log in.

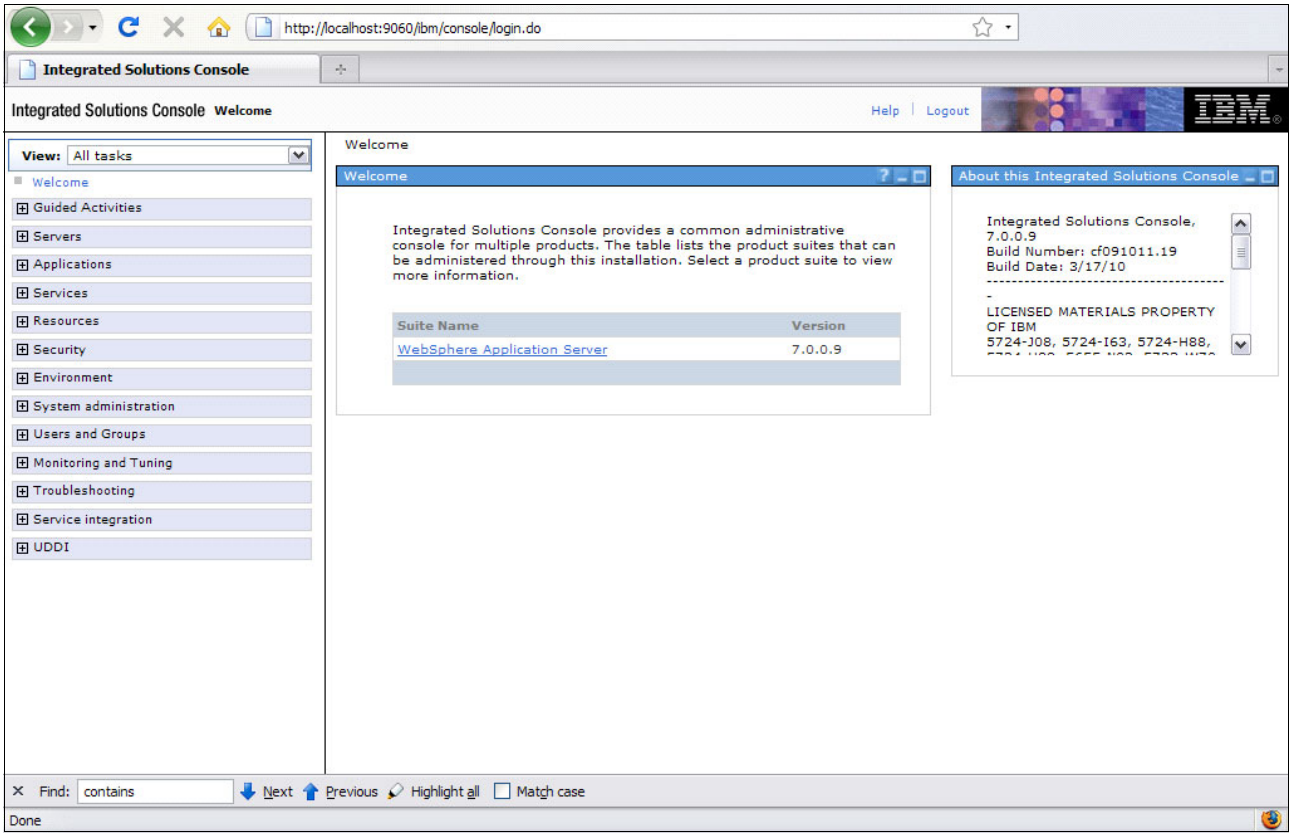


Figure 2-3 WebSphere administrative console

On the left side of the console is the navigation tree where you can navigate to other areas of interest, for example, Applications or Servers. In the middle of the console is an area for messages. After you log in, the console displays welcome messages. As you continue to use the console, it also displays error messages or a link allowing you to save your configuration changes. Underneath the messages is a workspace area for additional panels as you navigate through the navigation tree. In the current view, the workspace is empty. On the right is the area with additional panels to provide more information about the current page. This area is where the Help panel displays when you navigate to other pages.

2.6.3 Follow the breadcrumb

Start your interaction with the console from the navigation tree:

1. Select **Servers** → **Server Types** → **WebSphere application servers**.
2. In the workspace area, click **server1**. This action displays the Detail panel for `server1`, as shown in Figure 2-4 on page 31.

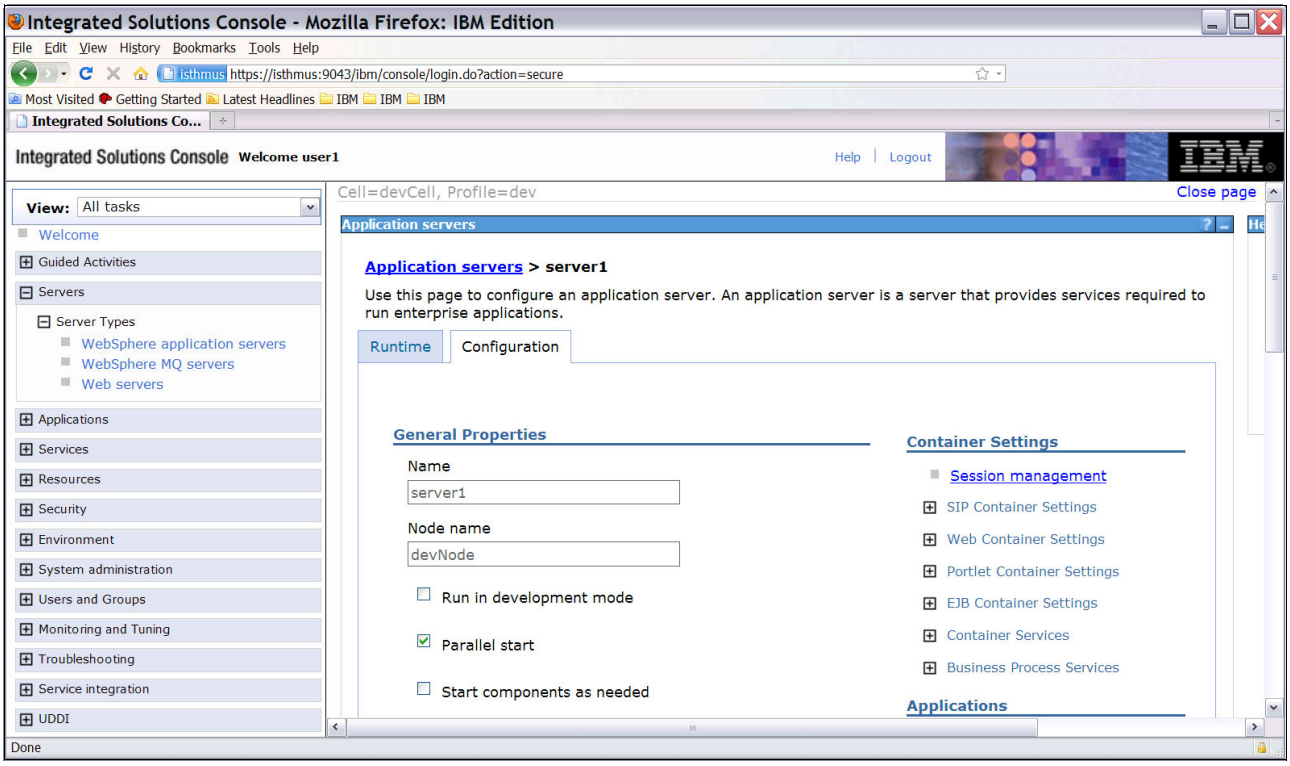


Figure 2-4 Administrative console detail pane

At the top of the workspace area, you can see the breadcrumb `Application servers > server1`. This breadcrumb shows the path that you have followed to get to this page. Part of the breadcrumb is a link that allows you to go back to a previous path in the navigation. For example, you can click Application servers to go back to the starting panel for application servers.

2.6.4 Configuration compared to run time

The console has **Configuration** and **Runtime** tabs.

- Configuration changes persist to the configuration repository and require a server restart for the changes to take place.
- Runtime changes apply immediately to the running state of the application server.

Browse attributes and the links under `server1`. Compare what you see to the elements and attributes in `server.xml`. Note that the administrative console Server panel is a front end to viewing and editing the contents of `server.xml`. Where there is a **Runtime** tab, click it to see what runtime operations are available.

2.6.5 Saving or discarding configuration changes

Follow these steps to save configuration changes:

1. Select **Application server** → **server1** → **Process Definitions** → **Java Virtual Machine**.
2. Change the maximum heap size to 512.
3. Click **OK**.
4. Under the Messages area, click **Save**.

5. Check **server.xml**, and ensure that the attribute has been set to `maximumHeapSize="512"`.

You do not have to save every change. The recommendation is for you to make all the related changes first, and then save all changes at one time. This approach allows you to exit if for any reason you make a mistake in one of the intermediate steps. Use the following steps as an example:

1. Select **Application server** → **server1** → **Process Definitions** → **Java Virtual Machine**.
2. Change the maximum heap size to 1024.
3. Click **OK**.
4. Click **Review**, and then click **Discard**.
5. Navigate back and ensure that the changes have been discarded.

2.6.6 Ports

You can find the ports on which the server is listening by using the administrative console by going to **Application servers** → **server1** → **Ports**.

Table 2-1 shows the ports and their default values.

Table 2-1 Port default values

Port name	Default value
SOAP_CONNECTOR_ADDRESS (SOAP Connector Port)	8880
SIP_DEFAULTHOST_SECURE (SIP Container Secure Port)	5061
SIP_DEFAULTHOST (SIP Container Port)	5060
SIB_ENDPOINT_ADDRESS (Service Integration Port)	7276
WC_defaulthost_secure (HTTPS Transport Port)	9443
DCS_UNICAST_ADDRESS (High Availability Manager Communication Port)	9353
SIB_MQ_ENDPOINT_SECURE_ADDRESS (MQ Transport Secure Port)	5578
WC_adminhost_secure (Administrative Console Secure Port)	9043
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS (CSIV2 Client Authentication Listener Port)	9402
ORB_LISTENER_ADDRESS (ORB Listener Port)	9100
BOOTSTRAP_ADDRESS (Bootstrap Port)	2809
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS (CSIV2 Server Authentication Listener Port)	9403
IPC_CONNECTOR_ADDRESS (IPC Connector Port)	9633
SIB_ENDPOINT_SECURE_ADDRESS (Service Integration Secure Port)	7289
WC_defaulthost (HTTP Transport Port)	9080
SIB_MQ_ENDPOINT_ADDRESS (MQ Transport Port)	5558
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	9401
WC_adminhost (Administrative Console Port)	9060

2.6.7 Applications

To access enterprise applications, as shown in Figure 2-5, select **Applications** → **Application Types** → **WebSphere enterprise applications**.

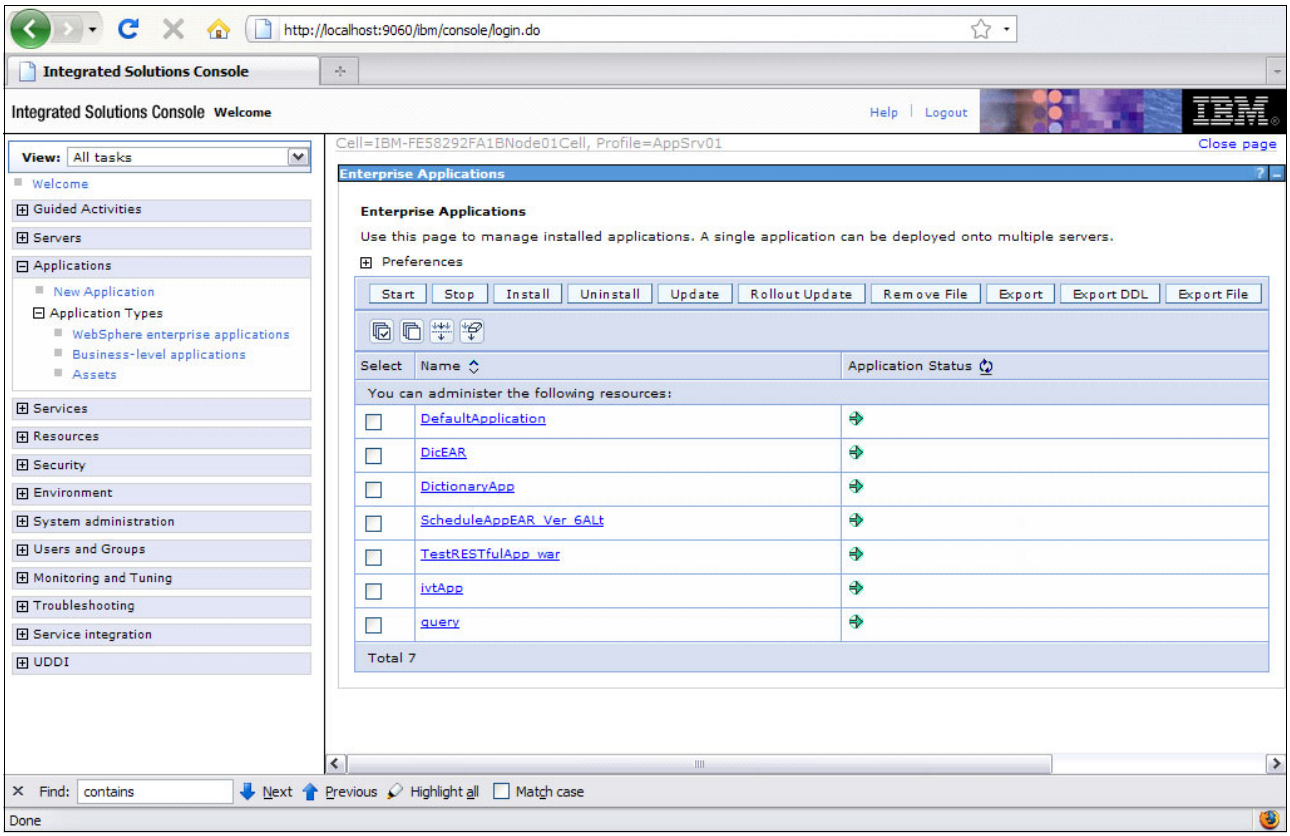


Figure 2-5 WebSphere Enterprise Applications

Starting and stopping an application

To start an application, such as `DefaultApplication`, select the application and click **Start**. To stop an application, select it and click **Stop**. You can also use the Install and Uninstall buttons to install or uninstall applications. Go to **Enterprise Applications** → **DefaultApplication** to learn more about the attributes and the links on this page.

Uninstalling an application

Use the following steps to uninstall the `ivtApp` application:

1. Select the `ivtApp` application from the list, and click **Uninstall**.
2. Follow the steps in the wizard to uninstall the application.
3. Click **Save** to save the changes.

Installing an application

Use the following steps to install the `ivtApp` application:

1. Uninstall the `ivtApp` application, if it is not already uninstalled.
2. Click **Install** to start the application deployment wizard.
3. Browse to *`INSTALL`*`/installableApps/ivtApp.ear`. Follow the wizard, using all of the default settings.

4. Save the configuration.
5. Select the new application (`IVT Application`), and click **Start**.
6. Go to the following URL:

   `http://localhost:9080/ivt/ivtservlet`

Updating an application

To update an application, you can uninstall and then install the application. Or, you can click **Update** on the Enterprise application panel. Choosing the default option allows you to update an entire application. Application update also allows you to update specific modules or files if your development environment has the capability to generate just the changed files since the last deployment.

2.6.8 Scoping

You can select the scope for variables and resources through the administrative console. For example, select **Environment** → **WebSphere variables**. Select the scope for which you want to work with the variables, as shown in Figure 2-6.

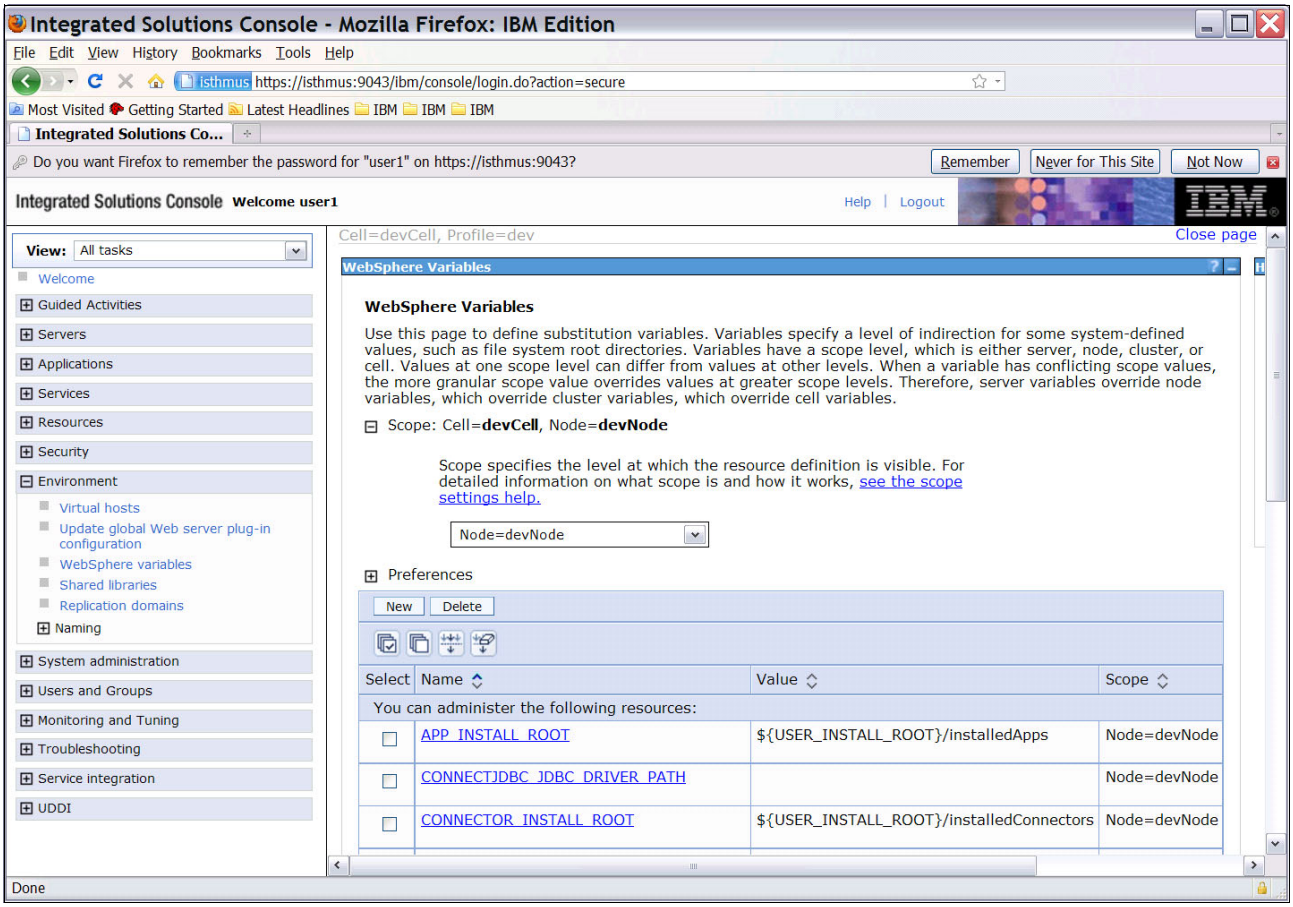


Figure 2-6 Scope for WebSphere variables

2.7 The wsadmin command and scripting

Although the administrative console gives you the capabilities that you need to manage the WebSphere environment, you will soon find that you need a programmatic application programming interface (API) to increase your productivity with repetitive and commonly used functions. The **wsadmin** command is useful in this situation. The **wsadmin** command supports two scripting languages: Java TCL (Jacl) and Jython. The support for Jacl, based on TCL, is currently stabilized, meaning no new function is expected to be added in future versions of the product. Jython, based on Python, is the current recommended scripting language and is the scripting language that we use in this book.

2.7.1 Configuring wsadmin

The configuration file for **wsadmin** is in *profile_root*/properties/wsadmin.properties. For backward compatibility, the default language for **wsadmin** is Jacl. You can change the default language to the recommended Jython language by changing the value in the following line in wsadmin.properties from Jacl to Jython:

```
com.ibm.ws.scripting.defaultLang=jython
```

If you do not change this line, you must specify the `-lang jython` option every time that you start **wsadmin**.

2.7.2 Starting wsadmin

The **wsadmin** command is located in *profile_root*/bin/wsadmin. The following options are the most useful:

-user *user*	The user ID to access the application server.
-password *password*	The password to access the application server.
-c *command*	Executes a single command. Configuration changes are automatically saved.
-f *file*	Executes the contents of a script file.
-host *host*	The host name of the application server.
-port *port*	The connector port of the application server.
-lang jacl | jython	The scripting language.
-conntype *conntype*	Specifies which connector to use, including SOAP, interprocess communication (IPC), Remote Method Invocation (RMI), and Java Specification Request (JSR) 160RMI (JSR160RMI).
-help	Prints available options.

2.7.3 Exiting wsadmin

If you specify the **-c** option, the command is run before **wsadmin** exits. If you specify the **-f** option, the file containing the script is run before **wsadmin** exits. If you do not specify one of these options, **wsadmin** starts in interactive mode. To exit interactive mode, type `exit`.

If you have made configuration changes, you need to type `exit` twice before the system allows you to exit with any changes discarded.

Example 2-5 shows a sample session that starts **wsadmin** in interactive mode and exits immediately.

Example 2-5 Starting wsadmin and exiting

```
C:\g\70x\Developer70\profiles\dev\bin>.\wsadmin.bat -user <user> -password
<password>
WASX7209I: Connected to process "server1" on node devNode using SOAP connector;
 The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>exit
```

2.7.4 wsadmin objects

The **wsadmin** command has five administrative objects that provide server configuration and management capabilities:

- "Help" on page 36
- "AdminControl" on page 37
- "AdminConfig" on page 40
- "AdminTask" on page 46
- "AdminApp" on page 48

Use the Help object to provide help in interactive mode. You can use the other objects to perform various administrative tasks.

Help

The Help object contains methods to print information about MBeans and other **wsadmin** objects. Start **wsadmin** in interactive mode, and type this command:

```
wsadmin>print Help.help()
```

A list of commands supported by the Help object displays, as shown in Example 2-6.

Example 2-6 wsadmin help object supported commands

```
wsadmin>print Help.help()
WASX7028I: The Help object has two purposes:

First, provide general help information for the the objects
        supplied by wsadmin for scripting: Help, AdminApp, AdminConfig,
        and AdminControl.

        Second, provide a means to obtain interface information about
        MBeans running in the system.  For this purpose, a variety of
        commands are available to get information about the operations,
        attributes, and other interface information about particular
        MBeans.

        The following commands are supported by Help; more detailed
        information about each of these commands is available by using the
        "help" command of Help and supplying the name of the command
        as an argument.

attributes              given an MBean, returns help for attributes
operations              given an MBean, returns help for operations
```

```
constructors            given an MBean, returns help for constructors
description             given an MBean, returns help for description
notifications           given an MBean, returns help for notifications
classname               given an MBean, returns help for classname
all                     given an MBean, returns help for all the above
help                    returns this help text
AdminControl            returns general help text for the AdminControl object
AdminConfig             returns general help text for the AdminConfig object
AdminApp                returns general help text for the AdminApp object
AdminTask               returns general help text for the AdminTask object
wsadmin                 returns general help text for the wsadmin script
                        launcher
message                 given a message id, returns explanation and
                        user action message
```

AdminControl

You use AdminControl to access MBeans that are not ConfigService, CommandManager, or AppManagement. These MBeans affect the runtime behavior in the application server.

Getting help with AdminControl()

Use the following command to get help with the syntax of the AdminControl object:

```
wsadmin>print AdminControl.help()
```

This command displays a complete description of all of the commands that are supported by AdminControl. If you want to get a detailed description of a specific command, you can narrow your search, as shown in Example 2-7.

Example 2-7 wsadmin AdminControl help query

```
wsadmin>print AdminControl.help ("queryMBeans")
WASX7456I: Method: queryMBeans

        Arguments: object name

        Description: Returns a Set containing ObjectInstances object
        that match the input object name.

        Method: queryMBeans

        Arguments: object name (type ObjectName), query (type QueryExp)

        Description: Returns a Set containing ObjectInstances object
        that match the input object name and query.
```

Getting MBean ObjectName

Each MBean has an object reference that is stored as an ObjectName. ObjectName has the following format:

"*domain*:*key1*=*val1*,*key2*=*val2*,..."

WebSphere MBeans all use the domain `WebSphere`. For our single server environment, the most important keys are `type` and `name`.

The following types of MBeans are useful for the development environment:

JVM	To get or set information about the JVM
Server	Information about the application server
ApplicationManager	For starting and stopping applications
Application	One MBean for each application currently deployed and started in the environment

Use the following command to query all available MBeans:

```
wsadmin>print AdminControl.queryNames("*:*")
```

To query all MBeans for a particular type, use the following `*:type=`*`type`*`,*` pattern, as shown in Example 2-8.

Example 2-8 wsadmin AdminControl MBean type query

```
wsadmin>print AdminControl.queryNames("*:type=JVM,*")
WebSphere:name=JVM,process=server1,platform=proxy,node=devNode,j2eeType=JVM,J2EESe
rver=server1,version=7.0.0.0,type=JVM,mbeanIdentifier=JVM,cell=devCell,spec=1.0
```

If you are interested in only one of the potentially many matching MBeans, use the example that is shown in Example 2-9.

Example 2-9 wsadmin AdminControl MBean application query

```
wsadmin>AdminControl.completeObjectName("*:type=Application,*")
WASX7026W: String "*:type=Application,*" corresponds to 10 different MBeans;
returning first one.
'WebSphere:name=ibmasyncrsp,process=server1,platform=dynamicproxy,node=devNode,J2E
EName=ibmasyncrsp,Server=server1,version=7.0.0.0,type=Application,mbeanIdentifier=
cells/devCell/applications/ibmasyncrsp.ear/deployments/ibmasyncrsp/deployment.xml#
ApplicationDeployment_1185820123453,cell=devCell,spec=1.0'
```

Listing MBean attributes and operations

Use the **`Help`** command to list the attributes and operations of an MBean, as shown in Example 2-10.

Example 2-10 wsadmin AdminControl MBean attributes and operations query

```
wsadmin>server=AdminControl.queryNames("*:type=Server,*")
wsadmin>print Help.attributes(server)
Attribute                       Type                            Access
name                            java.lang.String                RO
shortName                       java.lang.String                RO
threadMonitorInterval           int                             RW
threadMonitorThreshold          int                             RW
threadMonitorAdjustmentThreshold  int                             RW
pid                             java.lang.String                RO
cellName                        java.lang.String                RO
...

wsadmin>print Help.operations(server)
Operation
java.lang.String getName()
java.lang.String getShortName()
int getThreadMonitorInterval()
void setThreadMonitorInterval(int)
```

```
int getThreadMonitorThreshold()
void setThreadMonitorThreshold(int)
int getThreadMonitorAdjustmentThreshold()
void setThreadMonitorAdjustmentThreshold(int)
...
```

JavaDoc for MBean attributes and operations

To get a list of all available MBeans and their attributes and operations, go to the WebSphere Application Server Version 7.0 Information Center at this website:

`http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp`

From the navigation tree, click **WebSphere Application Server (Distributed and Windows)** → **Reference** → **Programing interfaces** → **Mbean interfaces**. A list of MBeans displays. Find the Server MBean, and click it. The JavaDoc for the Server MBean displays, as shown in Figure 2-7. You can also browse other beans using the same procedure.

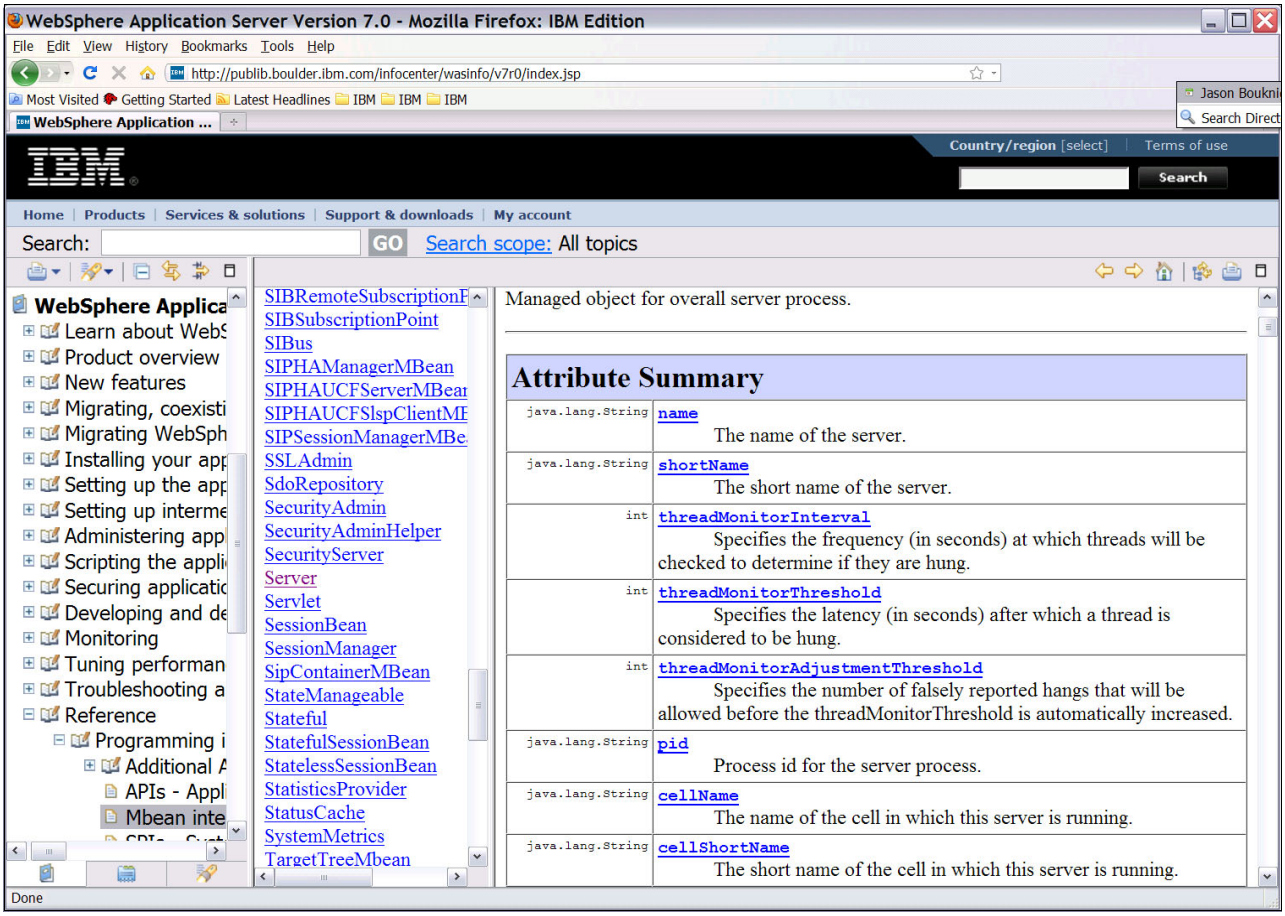


Figure 2-7 MBean JavaDoc in the WebSphere Information Center

Setting and getting attributes

Use **AdminControl.setAttribute()** to set an attribute on a MBean and **AdminControl.getAttribute()** to get an attribute. Example 2-11 shows how to set and query the attribute that the thread monitor uses to determine how long to wait before issuing a warning message that a thread has stopped.

Example 2-11 wsadmin AdminControl set and get attributes

```
wsadmin>server=AdminControl.queryNames("*:type=Server,*")
wsadmin>AdminControl.setAttribute(server, 'threadMonitorInterval', '240')
''
wsadmin>AdminControl.getAttribute(server, 'threadMonitorInterval')'240'
```

Invoking an operation on an MBean

Use the invoke method to invoke an operation on an MBean. Example 2-12 shows how to get the value of a system property in the application server JVM.

Example 2-12 wsadmin invoke system properties operation

```
wsadmin>jvm=AdminControl.queryNames("*:type=JVM,*")
wsadmin>AdminControl.invoke(jvm, "getProperty", "path.separator")
';'
```

You can obtain the value of an attribute by invoking the get <*attributeName*> method, and you can set the value of an attribute by using the set <*attributeName*> method, as shown in Example 2-13.

Example 2-13 wsadmin AdminControl invoke server intervals set and get operations

```
wsadmin>server=AdminControl.queryNames("*:type=Server,*")
wsadmin>AdminControl.invoke(server, "setThreadMonitorInterval", "360")
''
wsadmin>AdminControl.invoke(server, "getThreadMonitorInterval")
'360'
```

AdminConfig

The AdminConfig object allows you to access objects in the WebSphere configuration model and to persist them in the configuration repository.

Getting help

Use this command to get a list of operations on the AdminConfig object:

```
wsadmin>print AdminConfig.help()
```

Listing object types

To list object types in the configuration model, see Example 2-14.

Example 2-14 wsadmin AdminConfig list objects

```
wsadmin>print AdminConfig.types()
AccessPointGroup
ActivationSpec
ActivationSpecTemplateProps
ActiveAffinityType
ActivitySessionService
AdminAgentRegistration
...
```

Getting atttributes

Use `AdminConfig.attributes(“type”)` to list the attributes for a type, as shown in Example 2-15. The attributes of a type are persisted in the `.xml` configuration file as attributes and elements.

Example 2-15 wsadmin AdminConfig list attributes

```
wsadmin>print AdminConfig.attributes("DataSource")
authDataAlias String
authMechanismPreference ENUM(BASIC_PASSWORD, KERBEROS)
category String
connectionPool ConnectionPool
datasourceHelperClassname String
description String
diagnoseConnectionUsage boolean
jndiName String
logMissingTransactionContext boolean
manageCachedHandles boolean
mapping MappingModule
name String
preTestConfig ConnectionTest
properties Property(TypedProperty, DescriptiveProperty)*
propertySet J2EEResourcePropertySet
provider J2EEResourceProvider@
providerType String
relationalResourceAdapter J2CResourceAdapter@
statementCacheSize int
xaRecoveryAuthAlias String
```

Required attributes of a type

You must specify the required attributes of a type when creating a new object of that type, as shown in Example 2-16.

Example 2-16 wsadmin AdminConfig required attributes

```
wsadmin>print AdminConfig.required("DataSource")
Attribute                       Type
name                            String
```

Default values for a type

The default values for a type are not persisted in the configuration files, as shown in Example 2-17.

Example 2-17 wsadmin AdminConfig default values

```
wsadmin>print AdminConfig.defaults("DataSource")
Attribute                       Type                            Default
name                            String
jndiName                        String
description                     String
category                        String
providerType                    String
authMechanismPreference         ENUM                            BASIC_PASSWORD
authDataAlias                   String
manageCachedHandles             boolean                         false
logMissingTransactionContext    boolean                         true
xaRecoveryAuthAlias             String
diagnoseConnectionUsage         boolean                         false
statementCacheSize              int                             10
datasourceHelperClassname       String
provider                        J2EEResourceProvider
propertySet                     J2EEResourcePropertySet
connectionPool                  ConnectionPool
preTestConfig                   ConnectionTest
mapping                         MappingModule
properties                      Property
relationalResourceAdapter       J2CResourceAdapter
```

JavaDoc for configuration types

The JavaDoc for the types in the configuration model ship with the product and are at this location:

INSTALL/web/apiDocs/index.html

You need to know the package name for the type for which you are looking so that you can navigate to it from the navigation tree. Look at the XML namespace in the `.xml` file in the configuration repository, for example, to find the JavaDoc for a DataSource, and look at `resources.xml` to find the XML namespace:

```
xmlns:resources.jdbc="http://www.ibm.com/websphere/appserver/schemas/5.0/resources
.jdbc.xmi"
```

By convention, the starting package is `resource`, and the subpackage is `jdbc`, as shown in Figure 2-8 on page 43.

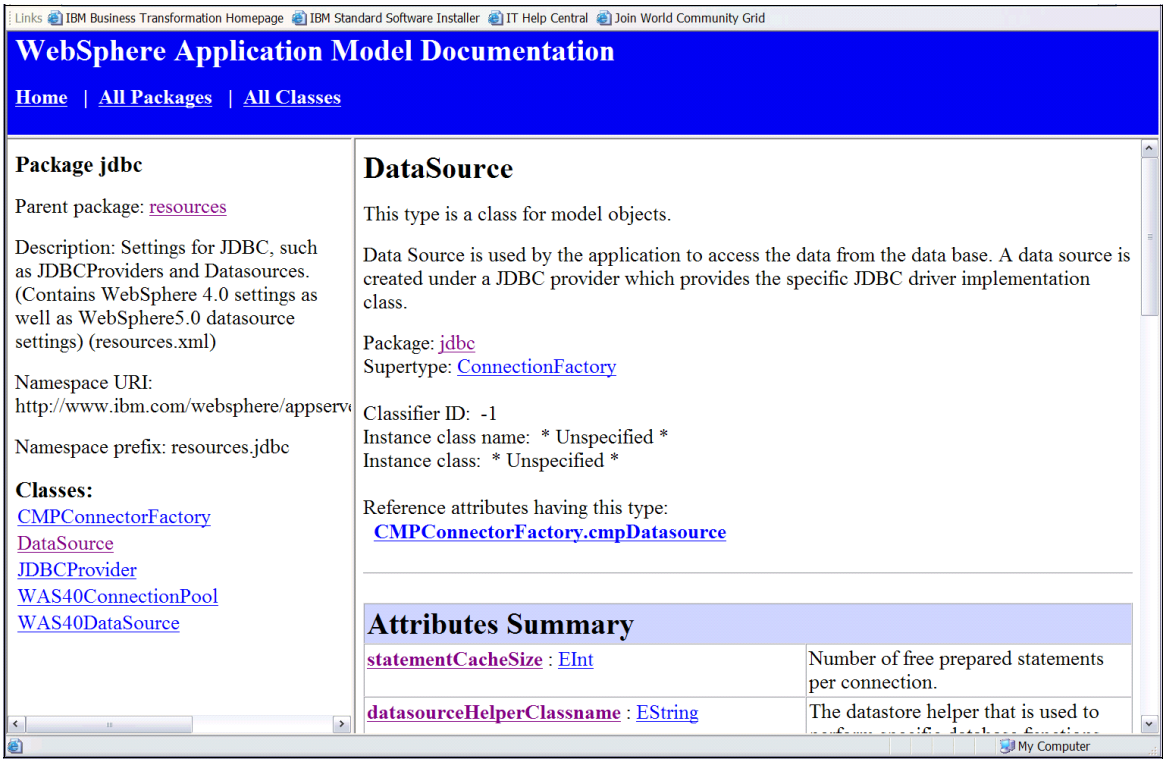


Figure 2-8 JavaDoc for DataSource configuration

Configuration ID

You use the configuration ID to uniquely identify an instance of a type in the configuration. Many operations in AdminConfig, including showing attributes, modifying attributes, and creating children instances, require a configuration ID. For example, the following configuration ID is for `server1`:

```
server1(cells/devCell/nodes/devNode/servers/server1|server.xml#Server_1183122130078)
```

You can use `AdminConfig.list()` to list all of the configuration IDs of a particular type, as shown in Example 2-18.

Example 2-18 wsadmin AdminConfig.list

```
wsadmin>AdminConfig.list("Cell")
'devCell(cells/devCell|cell.xml#Cell_1)'

wsadmin>AdminConfig.list("Node")
'devNode(cells/devCell/nodes/devNode|node.xml#Node_1)'

wsadmin>AdminConfig.list("Server")
'server1(cells/devCell/nodes/devNode/servers/server1|server.xml#Server_11831221300
78)'

wsadmin>AdminConfig.list("JavaVirtualMachine")
'(cells/devCell/nodes/devNode/servers/server1|server.xml#JavaVirtualMachine_118312
2130078)'

wsadmin>print AdminConfig.list("DataSource")
```

```
"Default
Datasource(cells/devCell/nodes/devNode/servers/server1|resources.xml#DataSource_11
83122153625)"
DefaultEJBTimerDataSource(cells/devCell/nodes/devNode/servers/server1|resources.xm
l#DataSource_1000001)
PLANTSDB(cells/devCell/nodes/devNode/servers/server1|resources.xml#DataSource_1183
122165968)
PLANTSDBNonJTA(cells/devCell/nodes/devNode/servers/server1|resources.xml#DataSourc
e_1204733259593)
```

Use `AdminConfig.getid()` to search for the configuration ID of a configuration object in a path, as shown in Example 2-19.

Example 2-19 wsadmin AdminConfig.getid configuration IDs

```
wsadmin>AdminConfig.getid("/Node:devNode/Server:server1/")
'server1(cells/devCell/nodes/devNode/servers/server1|server.xml#Server_11831221300
78)'

wsadmin>AdminConfig.getid("/Node:devNode/Server:server1/JDBCProvider:Derby JDBC
Provider/DataSource:Default Datasource")
'"Default
Datasource(cells/devCell/nodes/devNode/servers/server1|resources.xml#DataSource_11
83122153625)"'
```

You can limit your search in `AdminConfig.list()` by including a scope. In Example 2-20, the search looks for all `JDBCProviders` under `server1`.

Example 2-20 wsadmin AdminConfig.getid limited list search

```
wsadmin>server=AdminConfig.getid("/Node:devNode/Server:server1/")
wsadmin>print AdminConfig.list("JDBCProvider", server)
"Derby JDBC Provider
(XA)(cells/devCell/nodes/devNode/servers/server1|resources.xml#builtin_jdbcprovide
r)"
"Derby JDBC
Provider(cells/devCell/nodes/devNode/servers/server1|resources.xml#JDBCProvider_11
83122153343)"
"Samples Derby JDBC Provider
(XA)(cells/devCell/nodes/devNode/servers/server1|resources.xml#JDBCProvider_118312
2165828)"
```

Displaying and setting attributes

Use `AdminConfig.show()` to show the attributes of a configuration object, as shown in Example 2-21 on page 45.

Example 2-21 wsadmin AdminConfig.getid attributes search

```
wsadmin>ds=AdminConfig.getid("/Node:devNode/Server:server1/JDBCProvider:Derby JDBC
Provider/DataSource:Default Datasource")
wsadmin>print AdminConfig.show(ds)
[authMechanismPreference BASIC_PASSWORD]
[connectionPool
(cells/devCell/nodes/devNode/servers/server1|resources.xml#ConnectionPool_11831221
53625)]
[datasourceHelperClassname com.ibm.websphere.rsadapter.DerbyDataStoreHelper]
...
```

`AdminConfig.show()` does not follow attributes that are configuration IDs that point to a separate type. To show all attributes, use `Adminconfig.showall()`, as shown in Example 2-22.

Example 2-22 wsadmin AdminConfig.showall

```
wsadmin>print AdminConfig.showall(ds)
[authMechanismPreference BASIC_PASSWORD]
[connectionPool [[agedTimeout 0]
```

Use `AdminConfig.showAttribute()` to show a specific attribute, as shown in Example 2-23.

Example 2-23 wsadmin AdminConfig.showAttribute

```
wsadmin>AdminConfig.showAttribute(ds, "connectionPool")
'(cells/devCell/nodes/devNode/servers/server1|resources.xml#ConnectionPool_1183122
153625)'
```

Because the attribute returned is a configuration ID, you can use it for other configuration operations, as shown in Example 2-24.

Example 2-24 wsadmin AdminConfig.showAttribute

```
wsadmin>ds=AdminConfig.getid("/Node:devNode/Server:server1/JDBCProvider:Derby JDBC
Provider/DataSource:Default Datasource")
wsadmin>cp=AdminConfig.showAttribute(ds, "connectionPool")
wsadmin>print AdminConfig.show(cp)
[agedTimeout 0]
[connectionTimeout 180]
...
```

Use `AdminConfig.modify()` to modify an attribute of a configuration object. Example 2-25 shows how to change the trace specification for the application server to indicate what packages to trace.

Example 2-25 wsadmin AdminConfig.modify

```
wsadmin>server=AdminConfig.getid("/Node:devNode/Server:server1/")
wsadmin>ts=AdminConfig.list("TraceService", server)
wsadmin>AdminConfig.showAttribute(ts, "startupTraceSpecification")
'*=info'
wsadmin>newAttr=[ ["startupTraceSpecification", "com.ibm.ws.management.*=all"] ]
wsadmin>AdminConfig.modify(ts, newAttr)
''
wsadmin>AdminConfig.showAttribute(ts, "startupTraceSpecification")
```

```
'com.ibm.ws.management.*=all'
```

Saving or discarding changes

The changes that you make through AdminConfig are stored in the configuration workspace until you save your changes. Make all of your configuration changes and save them one time at the end so that you do not get partial changes saved to the master repository. This approach is also better for performance. Use the following **wsadmin** command to save your changes:

```
wsadmin>AdminConfig.save()
```

Use this command to discard all modifications that have been made since the last change:

```
wsadmin>AdminConfig.reset()
```

Use this command to see which files have been modified in the temporary workspace since your last save:

```
wsadmin>AdminConfig.queryChanges()
```

AdminTask

AdminTask offers logical operations that might involve multiple steps to AdminConfig, and potentially to AdminControl. It also works with other configurations in the WebSphere configuration tree that are not part of the configuration model that is accessible through AdminConfig, such as user registries.

Getting help with AdminTask

AdminTask commands are grouped into command groups.

The top-level help from AdminTask lists all of the options available for the AdminTask command, as shown in Example 2-26.

Example 2-26 wsadmin AdminTask help

```
wsadmin>print AdminTask.help()
```

The commands in a command group are related, as shown in Example 2-27.

Example 2-27 wsadmin AdminTask help

```
wsadmin>print AdminTask.help("-commandGroups")
WASX8005I: Available admin command groups:

AdminAgentNode - Admin Agent Managed Node related tasks
AdminAgentSecurityCommands - Commands used to configure security-related items
during Admin Agent registration.
AdminReports - Admin configuration reports
AdministrativeJobs - This command group contains all the job management commands.
…
```

You can also use wildcards to search for command groups, as shown in Example 2-28.

Example 2-28 wsadmin AdminTask help using wildcards

```
wsadmin>print AdminTask.help("-commandGroups", "*SIB*")
WASX8005I: Available admin command groups:
```

```
SIBAdminBusSecurityCommands - A group of commands that help configure SIB
security.
SIBAdminCommands - A group of commands that help configure SIB queues and
messaging engines.
SIBJMSAdminCommands - A group of commands that help configure SIB JMS connection
factories, queues and topics.
SIBWebServices - A group of commands to configure service integration bus Web
services.
```

You can list commands under a command group, as shown in Example 2-29.

Example 2-29 wsadmin AdminTask help listing commands under a command group

```
wsadmin>print AdminTask.help("SIBAdminCommands")
WASX8007I: Detailed help for command group: SIBAdminCommands

Description: A group of commands that help configure SIB queues and messaging
engines.

Commands:
addSIBBootstrapMember - Nominates a server or cluster for use as a bootstrap
server.
addSIBPermittedChain - Adds the specified chain to the list of permitted chains
for the specified bus.
addSIBusMember - Add a member to a bus.
```

You can get help for a specific command, as shown in Example 2-30.

Example 2-30 wsadmin AdminTask help

```
wsadmin>print AdminTask.help("createSIBus")
WASX8006I: Detailed help for command: createSIBus

Description: Create a bus.

Target object:   None

Arguments:
  *bus - Name of bus to create, which must be unique in the cell.
  description - Descriptive information about the bus.
```

Interactive mode and ports

AdminTask commands support an interactive mode that prompts you for input if you use the **'-interactive'** option. For example, **reportConfiguredPorts** prompts you for the node to report port usage, as shown in Example 2-31.

Example 2-31 wsadmin AdminTask reporting port usage

```
wsadmin>print AdminTask.reportConfiguredPorts('-interactive')
Ports configured in cell {0}

Generates a report of the ports configured in the cell

Node name (node): devNode

Ports configured in cell {0}
```

```
F (Finish)
C (Cancel)

Select [F, C]: [F]
WASX7278I: Generated command line: AdminTask.reportConfiguredPorts('[-node
devNode]')
Ports configured in cell devCell

Node devNode / Server server1
    localhost:2809 BOOTSTRAP_ADDRESS
    localhost:8880 SOAP_CONNECTOR_ADDRESS
    localhost:9100 ORB_LISTENER_ADDRESS
    localhost:9401 SAS_SSL_SERVERAUTH_LISTENER_ADDRESS
    localhost:9403 CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS
    localhost:9402 CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS
    *:9060 WC_adminhost
    *:9080 WC_defaulthost
    *:9353 DCS_UNICAST_ADDRESS
    *:9043 WC_adminhost_secure
    *:9443 WC_defaulthost_secure
    *:5060 SIP_DEFAULTHOST
    *:5061 SIP_DEFAULTHOST_SECURE
    *:7276 SIB_ENDPOINT_ADDRESS
    *:7286 SIB_ENDPOINT_SECURE_ADDRESS
    *:5558 SIB_MQ_ENDPOINT_ADDRESS
    *:5578 SIB_MQ_ENDPOINT_SECURE_ADDRESS
    ${LOCALHOST_NAME}:9633 IPC_CONNECTOR_ADDRESS
```

You can also use the listServerPorts command:

```
wsadmin>print AdminTask.listServerPorts("-interactive")...
```

Saving or discarding changes

You can use AdminConfig commands to save or discard changes that you made through AdminTask. See “Saving or discarding changes” on page 46 for more information.

AdminApp

The AdminApp object allows you to manage Java Enterprise Edition (JEE) applications. You can list, install, uninstall, update, and edit an application.

Help for AdminApp

Use the following command to display help for AdminApp:

```
wsadmin>print AdminApp.help()
```

Listing applications

You can list AdminApp applications, as shown in Example 2-32.

Example 2-32 wsadmin AdminApp list applications

```
wsadmin>print AdminApp.list()
DefaultApplication
PlantsByWebSphere
SamplesGallery
ivtApp
```

```
query
```

Uninstalling applications

You can use **AdminApp.uninstall*()*** to uninstall an application, as shown in Example 2-33. Remember to save the changes.

Example 2-33 wsadmin AdminApp uninstall applications

```
wsadmin>AdminApp.uninstall('ivtApp')
ADMA5017I: Uninstallation of ivtApp started.
ADMA5104I: The server index entry for WebSphere:cell=devCell,node=devNode is
updated successfully.
ADMA5102I: The configuration data for ivtApp from the configuration repository is
deleted successfully.
ADMA5011I: The cleanup of the temp directory for application ivtApp is complete.
ADMA5106I: Application ivtApp uninstalled successfully.
''
wsadmin>AdminConfig.save()
```

Installing applications

Use **AdminApp.installInteractive(*filename*)** to let **wsadmin** prompt you for options to install the application, as shown in Example 2-34. You can substitute <*INSTALL*> with the actual installation directory in your environment.

Example 2-34 wsadmin AdminApp install applications

```
wsadmin>AdminApp.installInteractive('<INSTALL>/installableApps/ivtApp.ear')
…
Application name:  [IVT Application]: ivtApp
…
wsadmin>AdminConfig.save()

The actual command used to deploy the application is found in
<profile_root>/logs/wsadmin.traceout.
[8/19/10 14:04:04:812 CDT] 00000000 AdminAppClien A   WASX7278I: Generated command
line: AdminApp.install('c:/g/70x/Developer70/installableApps/ivtApp.ear',
'[-nopreCompileJSPs -distributeApp -nouseMetaDataFromBinary -nodeployejb -appname
ivtApp -createMBeansForResources -noreloadEnabled -nodeployws -validateinstall off
-noprocessEmbeddedConfig -filepermission
.*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755 -buildVersion WAS70.SERV1 [r0834.28]
-noallowDispatchRemoteInclude -noallowServiceRemoteInclude
-asyncRequestDispatchType DISABLED -nouseAutoLink]')
```

You can simplify the command to incorporate it into your own script:

```
wsadmin>AdminApp.install('c:/g/70x/Developer70/installableApps/ivtApp.ear',
'[-appname ivtApp]')
```

Updating an application

If you want to update your application, you can either uninstall it followed by a reinstall, or you can use **AdminApp.update*()*** or **AdminApp.updateInteractive*()***. You have the option to update the entire application or update only part of the application. You can use partial update if your development tool has the capability to generate only the files that have changed since the last deployment.

Editing an application

Use **AdminApp.edit*()*** and **AdminApp.editInteractive*()*** to modify the information that you specified during the original application installation or update.

console command assistance

Operations carried out through the administrative console have equivalent commands in **wsadmin**. The administrative console offers command assistance by showing the most recent commands that it uses through the Help panel. Not all panels in the console offer command assistance, but the frequently used commands offer command assistance.

Go to the administrative console and navigate to Enterprise Applications, and stop the `DefaultApplication` application. In the Help panel, click **View administrative Scripting command for last action**. A new window opens with the equivalent scripting function, as shown in Example 2-35.

Example 2-35 Console command for wsadmin AdminControl

```
AdminControl.invoke('WebSphere:name=ApplicationManager,process=server1,platform=dy
namicproxy,node=devNode,version=7.0.0.0,type=ApplicationManager,mbeanIdentifier=Ap
plicationManager,cell=devCell,spec=1.0', 'stopApplication',
'[DefaultApplication]', '[java.lang.String]')
```

Start the application, and look at the command assistance window, as shown in Example 2-36.

Example 2-36 Console command assistance window

```
AdminControl.invoke('WebSphere:name=ApplicationManager,process=server1,platform=dy
namicproxy,node=devNode,version=7.0.0.0,type=ApplicationManager,mbeanIdentifier=Ap
plicationManager,cell=devCell,spec=1.0', 'startApplication',
'[DefaultApplication]', '[java.lang.String]')
AdminApp.list()
```

Note that the output of command assistance gives you the exact parameter values that it uses. You can easily adopt this output to a script if you want to automate your administration through scripting, as shown in Example 2-37.

Example 2-37 Console command assistance scripting example

```
wsadmin>appmgr=AdminControl.queryNames("*:type=ApplicationManager,*")
wsadmin>AdminControl.invoke(appmgr,'stopApplication', '[DefaultApplication]',
'[java.lang.String]')
''
wsadmin>AdminControl.invoke(appmgr,'startApplication', '[DefaultApplication]',
'[java.lang.String]')
''
```

You can further simplify the script by using another syntax for **AdminControl.invoke*()***, as shown in Example 2-38.

Example 2-38 Console command assistance scripting example

```
wsadmin>appmgr=AdminControl.queryNames("*:type=ApplicationManager,*")
wsadmin>AdminControl.invoke(appmgr,'stopApplication', "DefaultApplication")
''
wsadmin>AdminControl.invoke(appmgr,'startApplication', "DefaultApplication")
```

''

You can also configure command assistance to log output to a log file. Use Preferences in the command assistance window, as shown in Figure 2-9. The output goes to this log:

`<profile_root>/logs/server1/commandAssistanceJythonCommands_<username>.log`

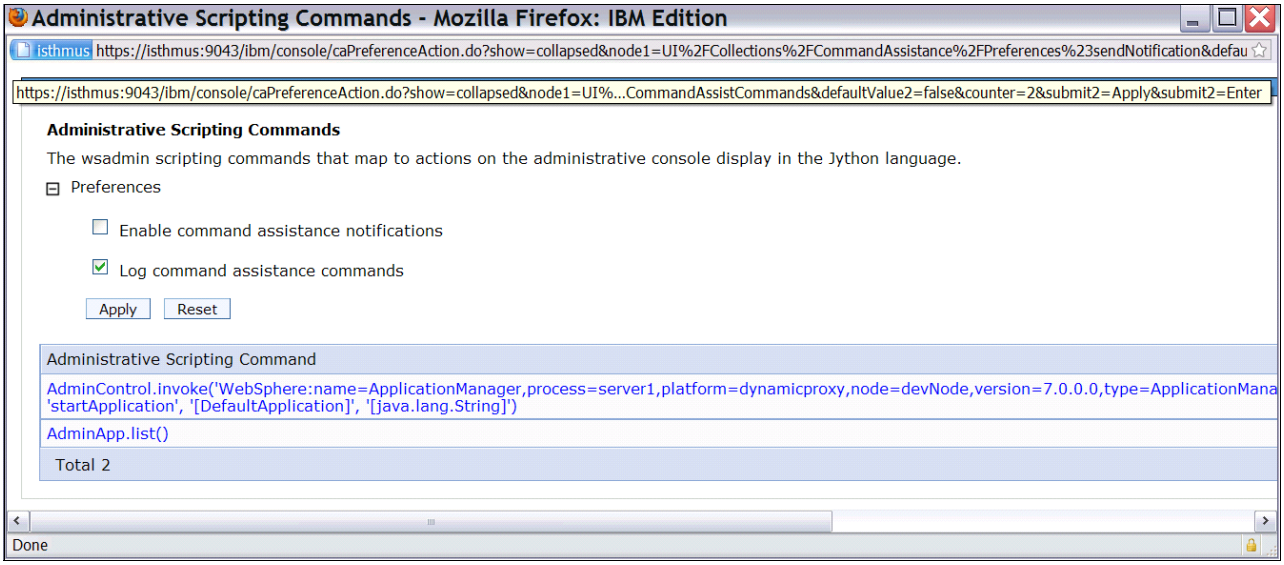


Figure 2-9 Configuring command assistance logging

Script library

If you need examples of ready to run scripts, go to the `<INSTALL>/scriptLibraries` directory. This directory contains sample scripts to handle application deployment, resource configuration, security configuration, and server configuration. Note that several of the scripts, such as managing clusters, creating application servers, and starting or stopping servers, only apply to the Network Deployment environment, not to our single server environment.

You can find documentation for the script library at the following website:

`http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp`

In the navigation tree, go to **WebSphere Application Server (IBM i), Version 7.0** → **Reference** → **Jython script library**, as shown in Figure 2-10 on page 52.

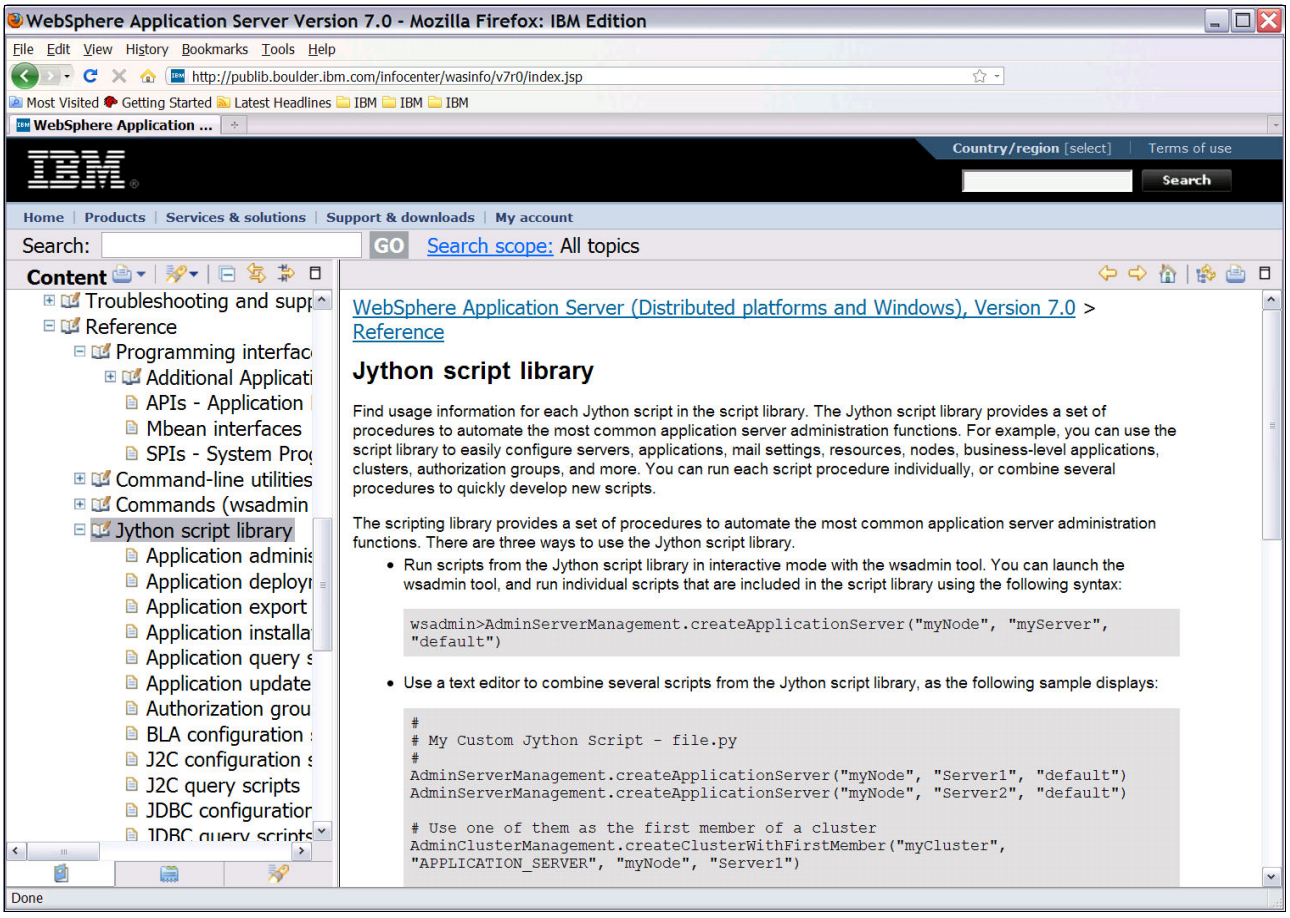


Figure 2-10 WebSphere Information Center Jython script library

Properties file-based configurations

Properties file-based configuration is a WebSphere administration tool that gives you the ability to manage WebSphere configuration through human-readable text files. It is designed primarily as an alternative to the AdminConfig object in **`wsadmin`**. Rather than writing scripts to navigate the configuration tree to query for configuration IDs and calling methods to get and set attributes on the configuration instances, you can declare the path to the objects and their associated attribute names and values. In addition, you can use properties file-based configuration as an alternative syntax to AdminApp and certain AdminTask operations.

Note: Before attempting to use the examples in this section, ensure that your WebSphere Application Server is at Version 7.0.0.9 or later.

Customizing your configuration using properties file-based configuration

Figure 2-11 shows the WebSphere Information Center section for Properties File-Based Configuration. In the navigation tree, click **Scripting the application server environment (wsadmin) → Using properties files to manage system configuration → Managing specific configuration objects using properties files.**

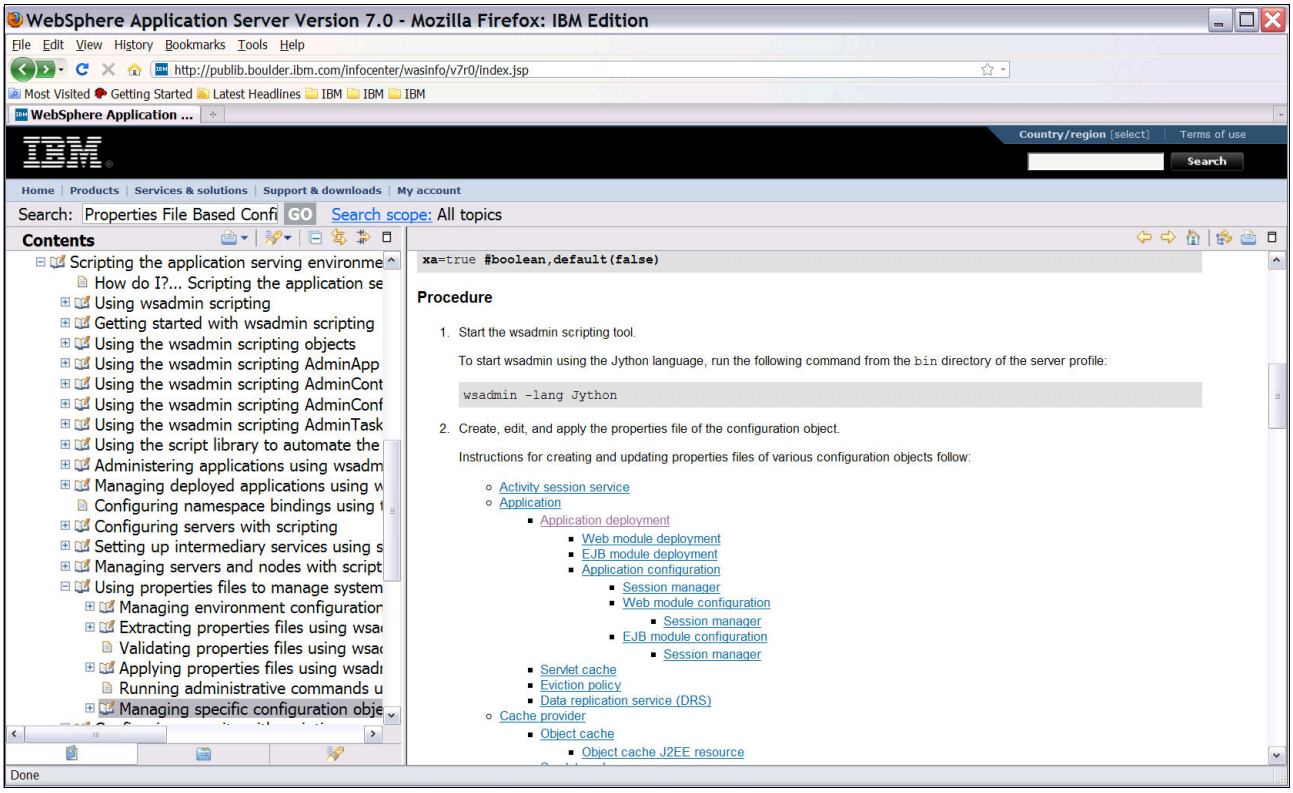


Figure 2-11 WebSphere Information Center properties file-based configuration

On this page are links to specific configuration objects in the WebSphere configuration that you want to modify. For example, if you want to modify the JavaVirtualMachine settings to the application server, click **Java Virtual Machine(JVM)**. Read the document about what actions are supported for this configuration object. For Java Virtual Machine, the Modify action is supported. If we want to modify the initial heap size to 256 Mb and the maximum heap size to 512 Mb, we use the following steps to produce the properties file-based configuration file:

1. Copy the sample properties file-based configuration file into an editor.
2. Edit the sample to only the types and attributes that you want to modify. For example, to change the heap sizes, the type is `JavaVirtualMachine`, and the attributes are `initialHeapSize` and `maximumHeapSize`. Do not include any other attribute, because these attributes are used to override the current values in the application server.
3. Apply the properties file.

Example 2-39 shows the resultant properties file.

Example 2-39 Properties file-based configuration file example

```
#
# Header
#
ResourceType=JavaVirtualMachine
ImplementingResourceType=GenericType
```

```
ResourceId=Cell=!{cellName}:Node=!{nodeName}:Server=!{serverName}:JavaProcessDef=:JavaVirtualMac
hine=
AttributeInfo=jvmEntries

#Properties
#
initialHeapSize=256 #integer,default(0)
maximumHeapSize=512 #integer,default(0)

EnvironmentVariablesSection
#
#Environment Variables
cellName=devCell
serverName=server1
nodeName=devNode
```

Save the file as `jvm.props`, and apply **`wsadmin -c "command"`** (Example 2-40).

Example 2-40 wsadmin .bat file

```
wsadmin.bat -c "AdminTask.applyConfigProperties('[-propertiesFileName jvm.props
-reportFileName jvm.report]')" -user user -password password
WASX7209I: Connected to process "server1" on node devNode using SOAP connector;
The type of process is: UnManagedProcess
''
```

You can verify that the configuration has been modified by examining `server.xml`.

You can make your `jvm.props` file environment neutral by placing the values of the variables into a separate environment variable file. We can create a separate environment variable file named `env.props`, as shown in Example 2-41.

Example 2-41 jvm.props file with a neutral environment

```
cellName=devCell
serverName=server1
nodeName=devNode
```

We create a `jvm1.props` file, which is a copy of `jvm.props` but without the environment variable entries. We also change the heap sizes to 128 and 256. Example 2-42 shows the new `jvm1.props` file.

Example 2-42 Copy of the jvm.props file

```
#
# Header : Make sure JavaHome is not in the property list or it is unchanged as it is readonly
#
ResourceType=JavaVirtualMachine
ImplementingResourceType=GenericType
ResourceId=Cell=!{cellName}:Node=!{nodeName}:Server=!{serverName}:JavaProcessDef=:JavaVirtualMac
hine=
AttributeInfo=jvmEntries

#Properties
#
initialHeapSize=128 #integer,default(0)
```

```
maximumHeapSize=256 #integer,default(0)
```

We apply the new properties, as shown in Example 2-43.

Example 2-43 env.props file with variables

```
wsadmin -c "AdminTask.applyConfigProperties('[-propertiesFileName jvm1.props
-variablesMapFileName env.props -reportFileName jvm1.report ]')" -user user
-password password
WASX7209I: Connected to process "server1" on node devNode using SOAP connector;
The type of process is: UnManagedProcess
''
```

You can verify that the configuration has been modified by examining `server.xml`.

Adding JVM system properties

To add additional system properties, edit the systemProperties section of the JavaVirtualMachine, for example, to add `myvar=myval` to the system properties of the application server, as shown in Example 2-44.

Example 2-44 systemProperties example

```
#
# Header JVM System properties
#
ResourceType=JavaVirtualMachine
ImplementingResourceType=Server
ResourceId=Cell=!{cellName}:Node=!{nodeName}:Server=!{serverName}:JavaProcessDef=:JavaVirtualMac
hine=
AttributeInfo=systemProperties(name,value)
#

#
#Properties
#
myvar=myval
```

We save the changes in a file named `jvmsystemprops.props` and apply the new properties, as shown in Example 2-45.

Example 2-45 Adding JVM system properties

```
wsadmin -c "AdminTask.applyConfigProperties('[-propertiesFileName
jvmsystemprops.props -variablesMapFileName env.props -reportFileName
jvmsystemprops.report ]')" -user user -password password
WASX7209I: Connected to process "server1" on node devNode using SOAP connector;
 The type of process is: UnManagedProcess
''
```

You can verify that the system property has been added by examining `server.xml`.

Deleting system properties

We delete the system properties in `jvmsystemprops.props`, but we use `AdminTask.deleteConfigProperties`, as shown in Example 2-46 on page 56.

Example 2-46 Deleting JVM system properties

```
wsadmin -c "AdminTask.deleteConfigProperties('[-propertiesFileName
jvmsystemprops.props -variablesMapFileName env.props -reportFileName
jvmsystemprops.report ]')" -user user -password password
WASX7209I: Connected to process "server1" on node devNode using SOAP connector;
 The type of process is: UnManagedProcess
```

You can verify that the system property has been deleted by examining `server.xml`.

Extracting the existing configuration through the properties file-based configuration

You can also start with your current configuration by first extracting the configuration into a text file, editing it, and then applying it back. To extract the configuration, use the following command in **wsadmin**:

```
AdminTask.extractConfigProperties (‘config id’ , ‘[options]’)
```

The optional *config id* is the configuration ID of the configuration object whose properties you want to extract. If *config id* is not specified, **wsadmin** extracts all the properties of all of the configuration objects specified by the **options** parameter:

-interactive Interactive mode.

-propertiesFileName *name* Where the extracted file is stored. This option is required.

-filtermechanism Specifies the method to filter the configuration objects. The options that can be specified after this option parameter are `ALL`, `NO_SUBTYPES`, `NO_SUBTYPES_AND_EXTENSIONS`, and `SELECTED_SUBTYPES`. `ALL` specifies that all configuration objects will be included in the properties file, including children. `NO_SUBTYPES` specifies that no children objects will be extracted. `NO_SUBTYPES_AND_EXTENSIONS` affects the behavior when extracting all of the objects under a scope, such as a server or node. The extensions are additional files outside of the configuration model that can also be extracted through properties file-based configuration, for example, node-metadata.properties. `SELECTED_SUBTYPES` specifies that only the children whose types are listed under the option `-selectedSubTypes` will be extracted.

-options Additional options. You should always use the [`PortablePropertiesFile true`] option to create an environment-neutral properties file.

Example 2-47 shows how to extract the JavaVirtual machine entries from our environment.

Example 2-47 Extracting JavaVirtual machine configuration

```
AdminTask.extractConfigProperties('[-propertiesFileName jvm2.props -configData
Server=server1 -options [[PortablePropertiesFile true]] -filterMechanism
SELECTED_SUBTYPES -selectedSubTypes [JavaVirtualMachine ] ]')
```

After the extraction, edit `jvm2.props`, and then apply it back. Because the file contains the most recent snapshot of the server, you do not need to remove any attribute that you do not want changed, because there is no danger of it differing when you apply it. However, you might still want to remove them so that you are sure to apply only those changes that you want to make, and make it portable enough that it can be applied against other profiles.

Creating and deleting configurations

As you browse each of the examples in the properties file-based configuration section of the Information Center, notice that certain types also support “create” or “delete” actions. Therefore, you can use properties file-based configuration to either create a new instance of the type or delete an instance. Follow the instructions to construct the properties file that is needed to create or delete the instance. We provide a specific example of how to create a JDBC data source in the next chapter.

2.8 Log files

While the application server is running, it logs information about its tasks and processes. These log files are useful for problem determination within the application server, such as debugging applications, troubleshooting server crashes, and diagnosing server startup errors. These log files are located in this directory:

profile_root\logs\server1

Types of logs

The following list describes the types of log files:

native_stderr.log Contains text written to the `stderr` streams by native modules (.dlls, .exes, UNIX® libraries, and other modules), including the JVM, or other parts of the security implementation

native_stdout.log Contains text written to the `stdout` streams by native modules (.dlls, .exes, UNIX libraries, and other modules), including the JVM, or other parts of the security implementation

serverStatus.log Contains the trace produced when the **`serverStatus`** is called

startServer.log Contains the trace produced when **`startServer`** is called

stopServer.log Contains the trace produced when **`stopServer`** is called

SystemErr.log The JVM log that contains messages written to the `System.err` stream and information concerning exceptions, the stack trace, and the server run time

SystemOut.log The JVM log that contains messages written to the `System.out` stream and information concerning exceptions, the stack trace, and the server run time

Trace.log The log containing the trace output of the application server, if trace is enabled

First Failure Data Capture (FFDC) log

Saves information that is generated from a processing failure

2.8.1 Log header

Each log has a header that contains important information about its contents, including these headers:

- Java Version
- Java Compiler
- Java VM Name
- Java Home
- ws.ext.dirs

- Java Library Path
- WebSphere Platform Version: Only included in `startServer.log` and `stopServer.log`
- Host Operating System: Only included in `startServer.log` and `stopServer.log` logs
- Process Name and Process Id: Only included in `native_stderr.log`, `native_stdout.log`, `SystemErr.log`, and `SystemOut.log`

Example 2-48 and Example 2-49 show the log header formats.

Example 2-48 Log file header format (Part 1 of 2)

```
************ Start Display Current Environment
************
Host Operating System is Windows XP, version 5.1
build 2600 Service Pack 3
Java version = J2RE 1.6.0 IBM J9 2.4 Windows XP
x86-32 jvmwi3260-20080816_22093 (JIT enabled, AOT
enabled) J9VM - 20080816_022093_lHdSMr JIT -
r9_20080721_1330ifx2 GC - 20080724_AA, Java
Compiler = j9jit24, Java VM name = IBM J9 VM
was.install.root = C:\WAS
user.install.root = C:\WAS\profiles\AppSrv01
Java Home = C:\WAS\java\jre
ws.ext.dirs =
C:\\WAS\\java\\lib;C:\\WAS\\classes;C:\\WAS\\lib;
C:\\WAS\\installedChannels;C:\\WAS\\lib\\ext;C:\\
WAS\\web\\help;C:\\WAS\\deploytool\\itp\\plugins\
\com.ibm.etools.ejbdeploy\\runtime
```

Example 2-49 Log file header format (Part 2 of 2)

```
Classpath =
C:\WAS\profiles\AppSrv01\properties;C:\WAS\proper
ties;C:\WAS\lib\startup.jar;C:\WAS\lib\bootstrap.
jar;C:\WAS/lib/lmproxy.jar;C:\WAS/lib/urlprotocol
s.jar;C:\was7000\java\lib\tools.jar
Java Library path =
C:\WAS\java\jre\bin;.;C:\WAS\bin;C:\WAS\java\bin;
C:\WAS\java\jre\bin;C:\WAS\bin;C:\WAS\java\bin;C:
\WAS\java\jre\bin;C:\WINDOWS\system32;C:\WINDOWS;
C:\WINDOWS\System32\Wbem;C:\Program
Files\IBM\Infoprint Select;C:\Notes;C:\Program
Files\XLView;C:\lotus\compnent;C:\Utilities;C:\Pr
ogram Files\Common Files\Lenovo;C:\program
files\ibm\personal communications\;C:\Program
Files\IBM\Trace Facility\;C:\Program
Files\Intel\WiFi\bin\;C:\Program
Files\ThinkPad\ConnectUtilities
Current trace specification =
*=info:com.ibm.*=all
************* End Display Current Environment
*************
```

2.8.2 JVM Log Message Format

`SystemOut.log` and `SystemErr.log` use the following format:

`timestamp threadId shortName eventType className methodName Message ID`

Where:

timestamp	The date and time of the event
threadID	The identifier (in Hexadecimal format) of the thread that generated the message
shortName	The abbreviated name of the component that generated the message
eventType	The type of message that was logged
className	The Java class that logged the message (optional)
methodName	The Java method that logged the message (optional)
Message ID	The identified type of the message and component

Event types

Table 2-2 describes the types of message IDs.

Table 2-2 Message IDs

ID	Description
F	Fatal message
W	Warning message
I	Informational message
D	Detail message
E	Error message
A	Audit message
C	Configuration message

2.8.3 Sample log messages

This section contains examples of informational, warning, and error messages. You can distinguish the type of the message by an `I` for informational, `W` for warning, and an `E` for error, as highlighted in Example 2-50, Example 2-51, and Example 2-52 on page 60.

Example 2-50 shows an example of an informational message.

Example 2-50 Informational message

```
[6/14/10 16:28:58:515 CDT] 00000000 ManagerAdmin
I TRAS0017I: The startup trace state is
*=info:com.ibm.*=all.
```

Example 2-51 shows an example of a warning message.

Example 2-51 Warning message

```
[6/8/10 13:54:11:312 CDT] 00000000 SSLConfig
```

```
W CWPKI0041W: One or more key stores are using
the default password.
```

Example 2-52 shows an example of an error message.

Example 2-52 Error message

```
[6/14/10 16:15:56:640 CDT] 00000000 WsServerLaunc
E ADMU3027E: An instance of the server may
already be running: server1
```

Native module log messages

The `native_stderr.log` and `native_stdout.log` files log native processes that lie outside of the JVM and are primarily invoked during server startup and shutdown. For example, processes involving class loading and operating system (OS)-specific operations are logged here. Example 2-53 shows a sample of these logs.

Example 2-53 Native module log messages

```
JVMVERB018W Invalid classpath entry:
C:\WAS/java/jre/lib/ext/ibmorb.jar (file I/O failed)
JVMVERB018W Invalid classpath entry:
C:\WAS/java/jre/lib/ext/ibmext.jar (file I/O failed)
...
class load: java/lang/Object
class load: java/lang/J9VMInternals
...
class unload:
sun/reflect/GeneratedSerializationConstructorAccessor3
class unload:
sun/reflect/GeneratedSerializationConstructorAccessor2
```

2.8.4 FFDC logs

FFDC stands for *First Failure Data Capture.* It is a framework that is used by the application server to collect errors early to an FFDC log, even if no tracing is enabled. These log files are deleted after a maximum number of days has passed, which can be set by performing the following steps:

1. Open the `ffdcRun.properties` file that is located in the *`install_root`*`/properties` directory.
2. Change the value for the `ExceptionFileMaximumAge` property to the number of days between the FFDC log file purges. The value must be a positive number. The default is seven days. For example, `ExceptionFileMaximumAge = 3` sets the default time to three days, and the log file is purged after three days.
3. Repeat the previous steps to modify the `ffdcStart.properties` and `ffdcStop.properties` files.
4. Save the `ffdcRun.properties` file, and exit.

If you see a message in the `SystemOut.log` file indicating that an FFDC incident stream file has been emitted, similar to what is shown in Example 2-54 on page 61, it is an indication that an FFDC log has been created for the incident.

Example 2-54 SystemOut.log sample

```
[8/17/10 14:34:50:187 CDT] 00000014 FfdcProvider
I com.ibm.ws.ffdc.impl.FfdcProvider logIncident
FFDC1003I: FFDC Incident emitted on
C:\was7000\profiles\dev\logs\ffdc\server1_24a424
a4_10.08.17_14.34.50.12527037.txt
com.ibm.ws.webcontainer.srt.BufferedWriter.write
Out 416
```

FFDC logs are located at <*install_root*>\profiles\<*profile_name*>\logs\ffdc. There are two types of files:

- Exception logs whose names are of the form *server_name*_Exception.log.
- Incident Stream logs whose file names are of the form *server_name_threadID_timestamp_sequence_number.txt.*

The exception logs have an entry for each of the FFDC incidents that happened since the server started. Example 2-55 shows a sample entry in an exception log.

Example 2-55 Exception log sample

```
Index Count Time of first Occurrence Time of last Occurrence
------+------+--------------------------+--------------------------+
0 1 8/17/10 14:34:50:109 CDT 8/17/10 14:34:50:109 CDT
Exception
-----------------------
com.ibm.wsspi.webcontainer.ClosedConnectionException
SourceId
-----------------------
com.ibm.ws.webcontainer.srt.BufferedWriter.writeOut
ProbeId
-----------------------
416
C:\was7000\profiles\dev\logs\ffdc\server1_24a424a4_10.08.17_14.34.50.12527037.t
xt
------+------+--------------------------+--------------------------+---------
--------------------
```

The incident stream contains more details about the exceptions. One incident stream file with a detailed thread dump of the thread where the exception occurred is created for each incident.

You can locate the incident file that corresponds to a particular exception log entry either by matching the Probe IDs or by matching the timestamp of the entry to the incident file. Example 2-56 shows an example of an incident stream.

Example 2-56 Incident stream

```
[8/17/10 14:34:50:125 CDT] FFDC
nonSourceId:com.ibm.ws.webcontainer.srt.BufferedWriter.writeOut
ProbeId:416
Reporter:com.ibm.wsspi.webcontainer.util.BufferedWriter@35903590
com.ibm.wsspi.webcontainer.ClosedConnectionException: OutputStream
encountered error during write
at
com.ibm.ws.webcontainer.channel.WCCByteBufferOutputStream.write(WCCByteBu
```

```
fferOutputStream.java:106)
at
com.ibm.ws.webcontainer.srt.SRTOutputStream.write(SRTOutputStream.java:97
)
at
sun.nio.cs.StreamEncoder$CharsetSE.writeBytes(StreamEncoder.java:355)
at
sun.nio.cs.StreamEncoder$CharsetSE.implFlushBuffer(StreamEncoder.java:425
)
at
sun.nio.cs.StreamEncoder$CharsetSE.implFlush(StreamEncoder.java:429)
at sun.nio.cs.StreamEncoder.flush(StreamEncoder.java:175)
at java.io.OutputStreamWriter.flush(OutputStreamWriter.java:223)
at
com.ibm.wsspi.webcontainer.util.BufferedWriter.writeOut(BufferedWriter.ja
va:486)
at
com.ibm.wsspi.webcontainer.util.BufferedWriter.flushChars(BufferedWriter.
java:373)
at
com.ibm.wsspi.webcontainer.util.BufferedWriter.flush(BufferedWriter.java:
348)
at java.io.PrintWriter.flush(PrintWriter.java:287)
at
org.apache.jasper.runtime.JspWriterImpl.flush(JspWriterImpl.java:233)
at java.io.PrintWriter.flush(PrintWriter.java:287)
at
org.apache.jasper.runtime.JspWriterImpl.flush(JspWriterImpl.java:233)
at
org.apache.struts.taglib.tiles.InsertTag$InsertHandler.doEndTag(InsertTag
.java:878)
at
org.apache.struts.taglib.tiles.InsertTag.doEndTag(InsertTag.java:473)
at.....
```

3

DictionaryApp example application

After installing WebSphere Application Server and learning the management tools that are included in the installation package, you are ready to explore the application deployment process. In this chapter, you create a sample application named `DictionaryApp` and deploy it on the application server.

DictionaryApp is an associative mapping application, which implements two functions of a dictionary: LOOKUP, which allows users to find the dictionary entry that is associated with the supplied word, and DEFINE, which allows users to update an existing entry or add a previously non-existent entry to the dictionary. DictionaryApp V1, the first version of DictionaryApp, implements the functionalities of a dictionary as a servlet that forwards user requests for LOOKUP and DEFINE to a database connection manager. The database connection manager performs these operations in a database containing the dictionary entries.

Sample material: See Appendix C, “Additional material” on page 235 for information about downloading the sample material that is used in this chapter.

© Copyright IBM Corp. 2010. All rights reserved.

3.1 What to know before proceeding

This guide assumes previous knowledge of Java and Java Platform 2, Enterprise Edition (J2EE) technologies. Therefore, this guide does not discuss in full detail the logic or the application programming interfaces (APIs) used by the code, because the major focus of the exercise is to obtain an understanding of how to deploy web applications to the application server. For this example application, we recommend using the Eclipse integrated development environment (IDE) for viewing the example application. See Appendix A, “Development tools reference” on page 219 for information about how to import DictionaryApp from the Enterprise Archive (EAR) files that are supplied within Eclipse with this document.

Because this exercise focuses mainly on the application deployment process, we do not require you to construct the example application, DictionaryApp. We have provided a compressed file that contains the EAR files with the contents of each version of DictionaryApp and the Derby script to create the database on which DictionaryApp will run. After you extract the contents of this archive, you can navigate to the `DictionaryApp` directory. This directory contains all of the available files. We refer to this directory as *Dictionary App Root* in the future whenever any of these files are needed.

3.2 DictionaryApp V1 overview

Figure 3-1 illustrates the overall architecture of DictionaryApp V1.

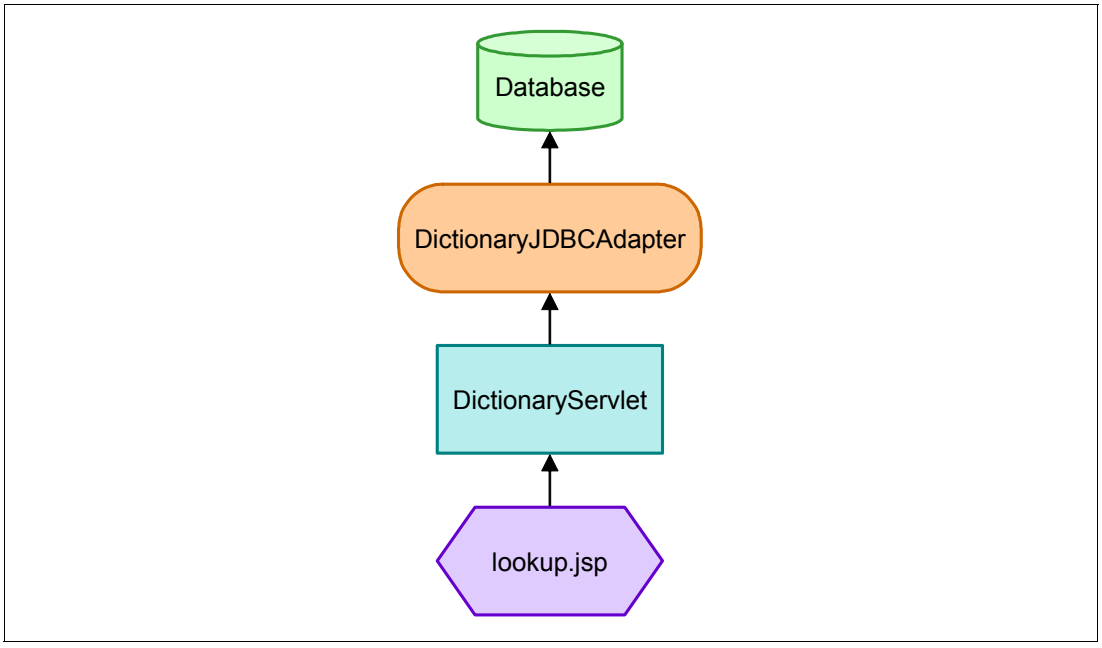


Figure 3-1 DictionaryApp V1

DictionaryApp V1 implements the Model View Controller (MVC) architecture by implementing `DictionaryJDBCAdapter.java`, which is a database adapter as the model, `showentry.jsp` as the view, and `DictionaryServlet.java` as the controller.

Whenever a user accesses DictionaryApp V1 through the DictionaryServlet entry point, the user is directed to `DictionaryServlet.java`. From there, `DictionaryServlet.java` processes

any requests that are specified by the user as tasks to be delegated to the database adapter, DictionaryJDBCAdapter.java. After DictionaryJDBCAdapter.java has finished the task, DictionaryServlet forwards the results to showentry.jsp, which formulates the appropriate response to send back to the user.

3.3 DictionaryApp V1 source files

This section provides source text for the files that you use to build DictionaryApp:

- showentry.jsp
- DictionaryServlet.java
- Dictionary.java
- DictionaryJDBCAdapter.java
- Entry.java
- Logger.java

These classes are separated in two archives: DictionaryWeb and DictionaryUtility. DictionaryWeb contains all of the controller and view components of the MVC architecture, including DictionaryServlet and showentry.jsp. DictionaryUtility contains the remaining classes that are needed to implement the model component of the architecture and other utility classes.

3.3.1 showentry.jsp

The showentry.jsp file is the view component in DictionaryApp V1. Example 3-1 shows the text of showentry.jsp.

Example 3-1 showentry.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1" import="java.util.Collection"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>WSDictionary</title>
</head>
<body>
<h1>Welcome to WSDictionary!</h1>
<h3 style="color: blue"><%=request.getAttribute("message") == null ? "" : request
              .getAttribute("message")%></h3>
<form action="DictionaryServlet">
<table>
   <tr>
      <td>Enter word:</td>
      <td><input name="word" type="text" size="40"
         value="<%=request.getAttribute("word") == null ? "" :
request.getAttribute("word")%>">
      <input type="submit" name="action" value="LOOKUP"
         style="height: 24px; width: 100px"></td>
   </tr>
   <tr>
      <td>Enter definition:</td>
```

```
      <td><textarea name="definition" rows="10" cols="30">
         <%=request.getAttribute("definition") == null ? "" :
request.getAttribute("definition")%></textarea> <input type="submit" name="action"
value="DEFINE" style="height: 24px; width: 100px"></td>
   </tr>
</table>
</form>
</body>
</html>
```

3.3.2 DictionaryServlet.java

The `DictionaryServlet.java` file is the controller component for DictionaryApp V1, which is shown in Example 3-2. `DictionaryServlet.java` processes all of the incoming user requests for DictionaryApp V1 and serves as the moderator for delegating each task to the components in DictionaryApp V1.

Example 3-2 DictionaryServlet.java

```
package com.ibm.dictionaryapp.servlet;

import java.io.IOException;

import javax.annotation.Resource;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

import com.ibm.dictionaryapp.database.DictionaryDatabaseAdapter;
import com.ibm.dictionaryapp.database.DictionaryJDBCAdapter;
import com.ibm.dictionaryapp.database.Entry;
import com.ibm.dictionaryapp.datautil.DataManager;

public class DictionaryServlet extends HttpServlet {

   private static final long serialVersionUID = 1L;

   @Resource(name = "jdbc/DictionaryDB")
   public DataSource datasource;

   private DictionaryJDBCAdapter jdbcadapter;

   public void init() throws ServletException {
      super.init();
      jdbcadapter = new DictionaryJDBCAdapter(datasource);
   }

   protected void doGet(HttpServletRequest request,
         HttpServletResponse response) throws ServletException, IOException {
      String action = request.getParameter("action");
      String description = "";
```

```
      String message = "";
      DictionaryDatabaseAdapter databaseadapter = jdbcadapter;
      try {
         if (action == null) {
            action = "NO ACTION";
         } else if (action.equals("LOOKUP")) {
            String word = request.getParameter("word");
            if (word == null || word.length() == 0) {
               message = "Please enter a word to look up.";
            } else {
               Entry entry = databaseadapter.lookup(word);
               if (entry == null) {
                  message = word + " was not found.  Please define it.";
                  request.setAttribute("word", word);
                  description = "FAILED";
               } else {
                  request.setAttribute("word", entry.getWord());
                  request.setAttribute("definition", entry
                        .getDefinition());
                  description = entry.toString();
               }
            }
         } else if (action.equals("DEFINE")) {
            String word = request.getParameter("word");
            String definition = request.getParameter("definition");
            if (word == null || word.length() == 0 || definition == null
                  || definition.length() == 0) {
               message = "Please enter a word and defintion.";
            } else {
               Entry entry = new Entry(word, definition);
               databaseadapter.define(entry);
               message = word + " has been defined.";
               request.setAttribute("word", word);
               request.setAttribute("definition", definition);
               description = entry.toString();
            }
         }
         DataManager.logAction(action, description);
      } catch (Exception e) {
         DataManager.logError(e);
         message = "ERROR : " + e.getClass().getName();
      }
      request.setAttribute("message", message);
      ServletContext context = getServletContext();
      RequestDispatcher dispatcher = context
            .getRequestDispatcher("/showentry.jsp");
      dispatcher.forward(request, response);
   }
}
```

3.3.3 Dictionary.java

The `Dictionary.java` file is the interface that serves as an abstract representation of a dictionary, as shown in Example 3-3 on page 68. Note that it only contains LOOKUP and

DEFINE, the two major functionalities of DictionaryApp V1. This abstraction of the internal workings of database access further separates the model from the controller and makes it easier for DictionaryServlet.java to switch database adapters in later versions.

Example 3-3 Dictionary.java

```
package com.ibm.dictionaryapp.database;

public interface Dictionary {
   public Entry lookup(String word) throws Exception;

   public void define(Entry entry) throws Exception;
}
```

3.3.4 DictionaryJDBCAdapter.java

The `DictionaryJDBCAdapter.java` file, as shown in Example 3-4, is an implementation of `DictionaryDatabaseAdapter.java` that incorporates the model component of DictionaryApp V1.

Example 3-4 DictionaryJDBCAdapter.java

```
package com.ibm.dictionaryapp.database;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.sql.DataSource;

public class DictionaryJDBCAdapter implements Dictionary {

   public DataSource datasource;

   public DictionaryJDBCAdapter(DataSource datasource) {
      this.datasource = datasource;
   }

   public Entry lookup(String word) throws SQLException {
      Connection connection = null;
      PreparedStatement statement = null;
      ResultSet resultset = null;
      try {
         connection = datasource.getConnection();
         statement = connection
               .prepareStatement("SELECT * FROM Dictionary WHERE WORD = ?");
         statement.setString(1, word);
         resultset = statement.executeQuery();
         if (resultset.next()) {
            return new Entry(resultset.getString("WORD"), resultset
                  .getString("DEFINITION"));
         } else {
            return null;
         }
```

```
      } finally {
         if (resultset != null)
            resultset.close();
         if (statement != null)
            statement.close();
         if (connection != null)
            connection.close();
      }
   }

   public void define(Entry entry) throws SQLException {
      Connection connection = null;
      PreparedStatement verifystatement = null;
      PreparedStatement updatestatement = null;
      PreparedStatement insertstatement = null;
      ResultSet verifyresults = null;
      try {
         connection = datasource.getConnection();
         verifystatement = connection
               .prepareStatement("SELECT * FROM DICTIONARY WHERE WORD = ?");
         updatestatement = connection
               .prepareStatement("UPDATE DICTIONARY SET DEFINITION = ? WHERE WORD
= ?");
         insertstatement = connection
               .prepareStatement("INSERT INTO DICTIONARY (WORD, DEFINITION) VALUES
(?, ?)");
         verifystatement.setString(1, entry.getWord());
         updatestatement.setString(1, entry.getDefinition());
         updatestatement.setString(2, entry.getWord());
         insertstatement.setString(1, entry.getWord());
         insertstatement.setString(2, entry.getDefinition());
         verifyresults = verifystatement.executeQuery();
         if (verifyresults.next()) {
            updatestatement.execute();
         } else {
            insertstatement.execute();
         }
      } finally {
         if (verifystatement != null)
            verifystatement.close();
         if (updatestatement != null)
            updatestatement.close();
         if (insertstatement != null)
            insertstatement.close();
         if (connection != null)
            connection.close();
      }
   }
}
```

3.3.5 Entry.java

The `Entry.java` file, as shown in Example 3-5 on page 70, is the Java bean representation of a dictionary entry in DictionaryApp V1.

Example 3-5 Entry.java

```
package com.ibm.dictionaryapp.database;

public class Entry {
   private String word;

   private String definition;

   public Entry() {
   }

   public Entry(String entrystring) {
      String[] entryfields = entrystring.split(" : ");
      word = entryfields[0];
      definition = entryfields[1];
   }

   public Entry(String word, String definition) {
      this.word = word;
      this.definition = definition;
   }

   public String getWord() {
      return word;
   }

   public void setWord(String word) {
      this.word = word;
   }

   public String getDefinition() {
      return definition;
   }

   public void setDefinition(String definition) {
      this.definition = definition;
   }

   public String toString() {
      return word + " : " + definition;
   }
}
```

3.3.6 Logger.java

The `Logger.java` file, as shown in Example 3-6, is the logging utility class, which uses the System.out print stream to output all user actions to the `SystemOut.log` file. The plan is for more functionality to be added to this class in later versions of DictionaryApp.

Example 3-6 Logger.java

```
package com.ibm.dictionaryapp.datautil;

import java.util.Date;

public class Logger {
   public static void logError(Exception e) {
      logAction("Error", e.getClass().getName() + ": " + e.getMessage());
   }

   public static void logAction(String action, String description) {
      System.out.println(new Date().toString() + "\t:" + action + "("
            + description + ")");
   }
}
```

3.4 Creating a Derby database

In DictionaryApp V1, `DictionaryJDBCAdapter.java` implements LOOKUP and DEFINE through calling SQL commands in the form of Java Database Connectivity (JDBC) queries against a data source object with the Java Naming and Directory Interface (JNDI) name jdbc/DictionaryDB. After DictionaryApp is deployed onto WebSphere, WebSphere attempts to map an actual data source resource to the DataSource object with the JNDI name jdbc/DictionaryDB in `DictionaryServlet.java`. This reference is then passed into `DictionaryJDBCAdapter.java` to provide a database connection for the database adapter.

Thus, in order for DictionaryApp V1 to run correctly, it is necessary to create a database to hold the entries of the dictionary and a data source to serve as the connection between the application server and the database. Derby is a relational database management system that is included with WebSphere. You can configure any application to use a Derby database when a database resource is required. Use the following procedure to create a Derby database:

1. From a command prompt, go to the directory:

   install root`\derby\bin\embedded`

2. Run **`ij.bat`**.

   Note: Inside `ij.bat`, end every command with a semicolon, because `ij.bat` does not process commands until a semi-colon is issued.

3. From a command prompt, type:

   `connect 'jdbc:derby:`*database directory*`;create=true';`

   This command opens a database connection to the database directory and creates a database if a database does not already exist in the database directory. For the DictionaryApp V1, use this command:

```
connect 'jdbc:derby:DictionaryDatabase;create =true';
```

In our example, because the exact file path has not been specified, Derby creates a new database inside this path:

install root\derby\DictionaryDatabase

This file path serves as the database name attribute when you configure a data source with WebSphere.

When running **ij.bat**, you can execute SQL commands against DictionaryDatabase as long as the connection with DictionaryDatabase remains open. For the purposes of DictionaryApp, create a table named Dictionary, which will contain all of the entries that need to be stored. To create this table, after creating the database, issue the following command:

```
create table Dictionary(word varchar(32),definition varchar(250), primary
key(word));
```

Then, populate the Dictionary table with a few events. For example, use this command to add IBM as an entry in the dictionary:

```
insert into Dictionary(word, definition)values('IBM', 'International Business
Machines');
```

You might have noticed that the columns of the table Dictionary match the queries made in DictionaryApp in `DictionaryJDBCAdapter.java`. It is crucial that the table Dictionary is created under these specifications in order for Dictionary App V1 to run correctly.

When you have finished, disconnect from DictionaryDatabase and exit **ij.bat** using this command:

```
exit;
```

Alternatively, you can run the prepared SQL script that is provided in the DictionaryApp attachment by using this command:

```
ij.bat Dictionary AppRoot\DictionaryAppDerbyScript.sql
```

This script contains all of the instructions for **ij.bat** to accomplish all of the tasks in this section and to prepare DictionaryDatabase for DictionaryApp V1.

3.5 Creating a data source

After creating DictionaryDatabase, you can specify how the application server accesses DictionaryDatabase by configuring a data source: DictionaryDatasource. There are three ways to configure a data source:

- “Administrative console” on page 72
- “wsadmin scripting” on page 78
- “Property file-based configuration” on page 80

3.5.1 Administrative console

The administrative console is a simple and easy to access tool for managing WebSphere resources, including data sources. For users new to WebSphere, you can use the administrative console using the user friendly browser interface and step-by- step approach to configuring a data source. In this section, we demonstrate how to create and test a data source through the administrative console.

Creating DictionaryDatasource

Use the following procedure to configure DictionaryDatasource using the administrative console:

1. Access the administrative console.
2. In the navigation tree, click **Resources** → **JDBC** → **Data sources**. This option takes you to the Data sources page, as shown in Figure 3-2.

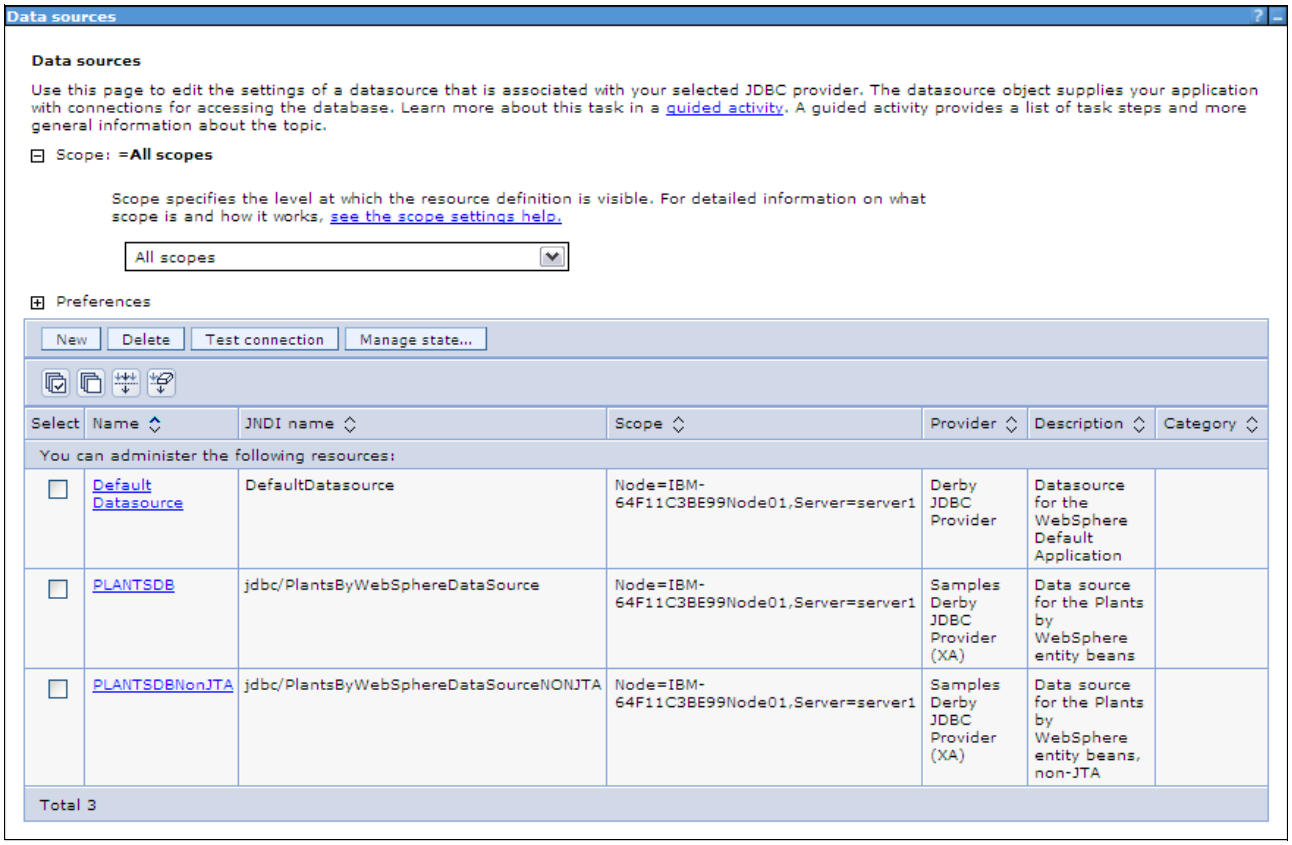


Figure 3-2 Administrative console: Data sources

3. Ensure that **Scope** is expanded, and then expand the drop-down list box. This drop-down list box provides the list of all of the scopes for which a data source can be created. Select the most appropriate scope for the data source. For DictionaryApp V1, select the scope that includes `server1`. Click **New** to add the new data source, as shown in Figure 3-3.

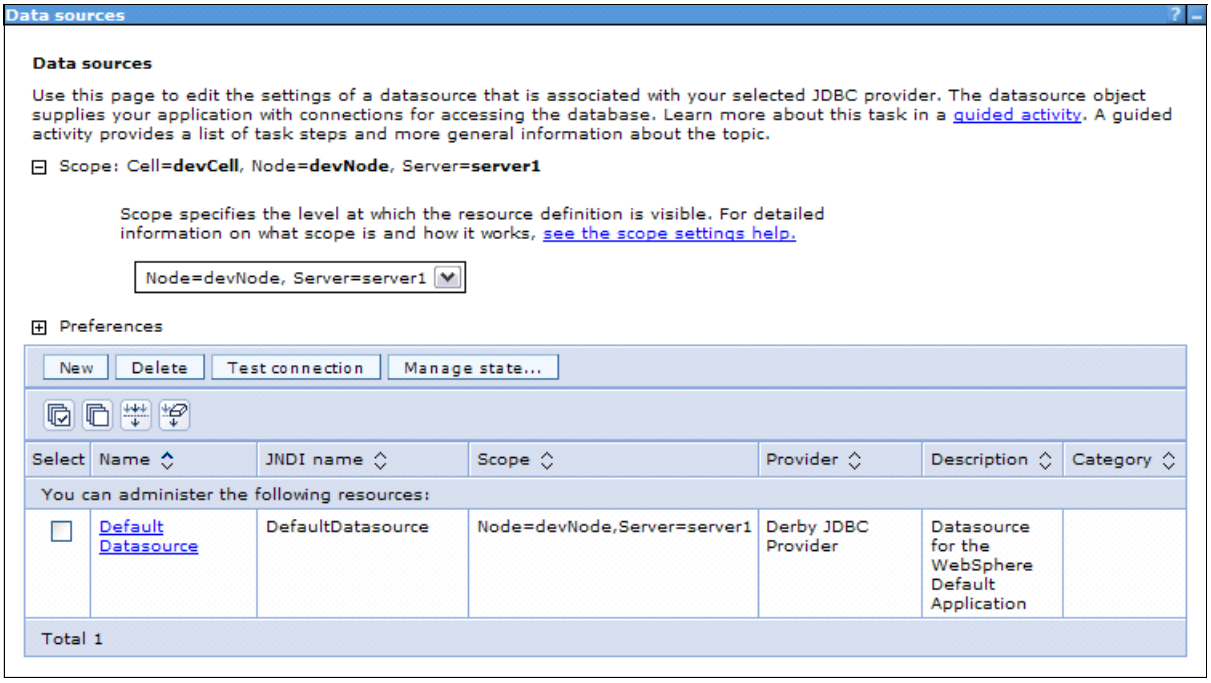


Figure 3-3 Administrative console: Create a data source

4. Select the desired data source name and JNDI name for the data source. For DictionaryApp, use `DictionaryDatasource` and `JNDI jdbc/DictionaryDB`, as shown in Figure 3-3. Click **Next**.

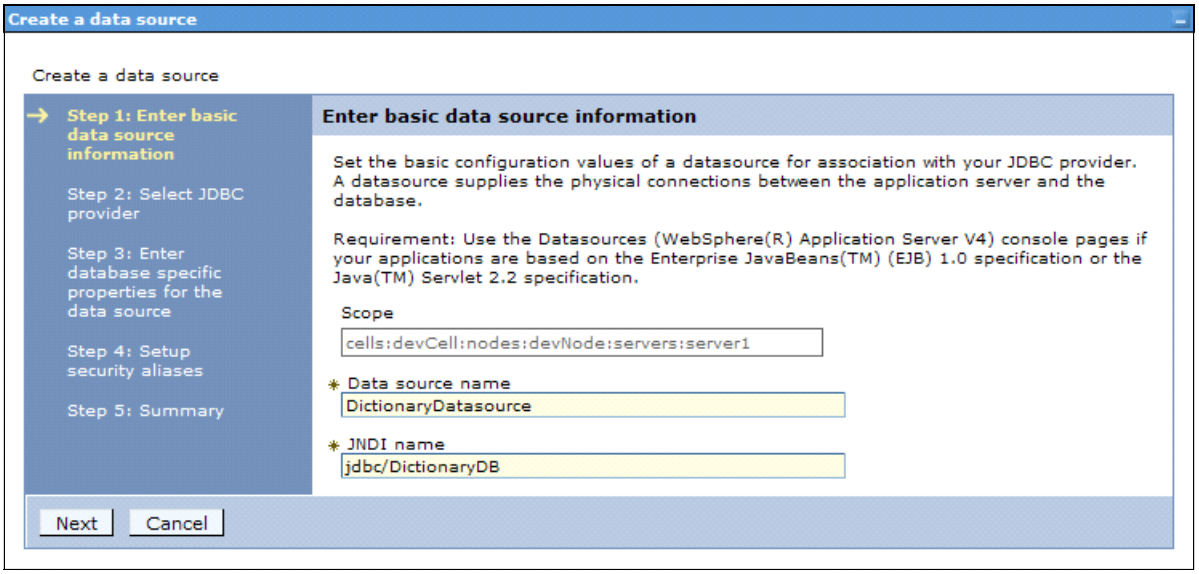


Figure 3-4 Administrative console: Create a data source

5. Select the desired JDBC provider for the data source. For DictionaryApp V1, choose **Select an existing JDBC provider** and select **Derby JDBC Provider** from the drop-down list box, as shown in Figure 3-5. Click **Next**.

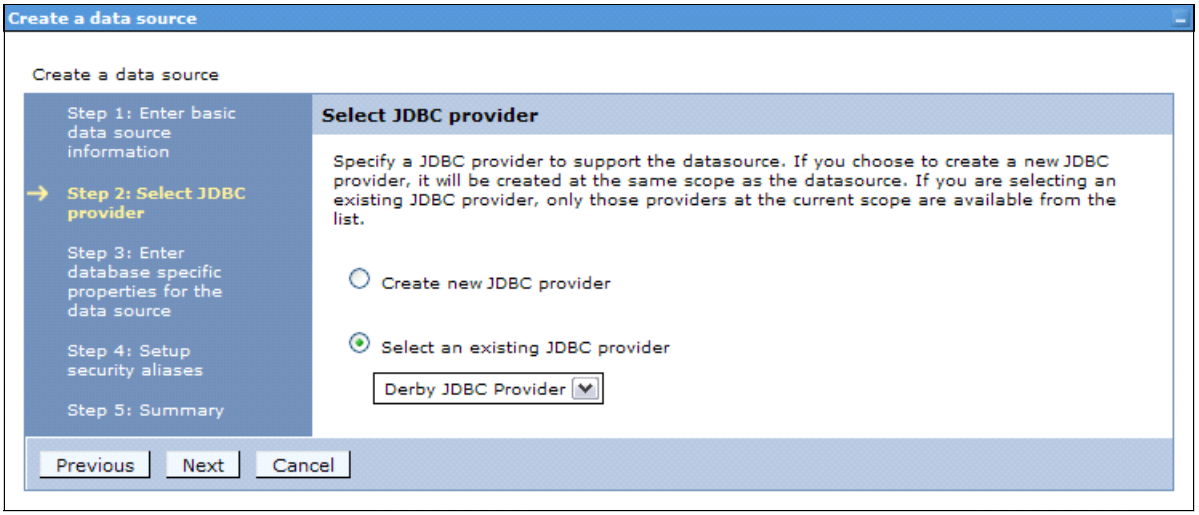


Figure 3-5 Administrative console: Select JDBC provider

6. In the Value field for Database name, type the value for the database directory that was created in Derby, as shown in Figure 3-6. For DictionaryApp V1, the file path is the same as the directory that was specified when you created DictionaryDatabase:

   WAS_INSTALL_ROOT\derby\DictionaryDatabase

   Click **Next**.

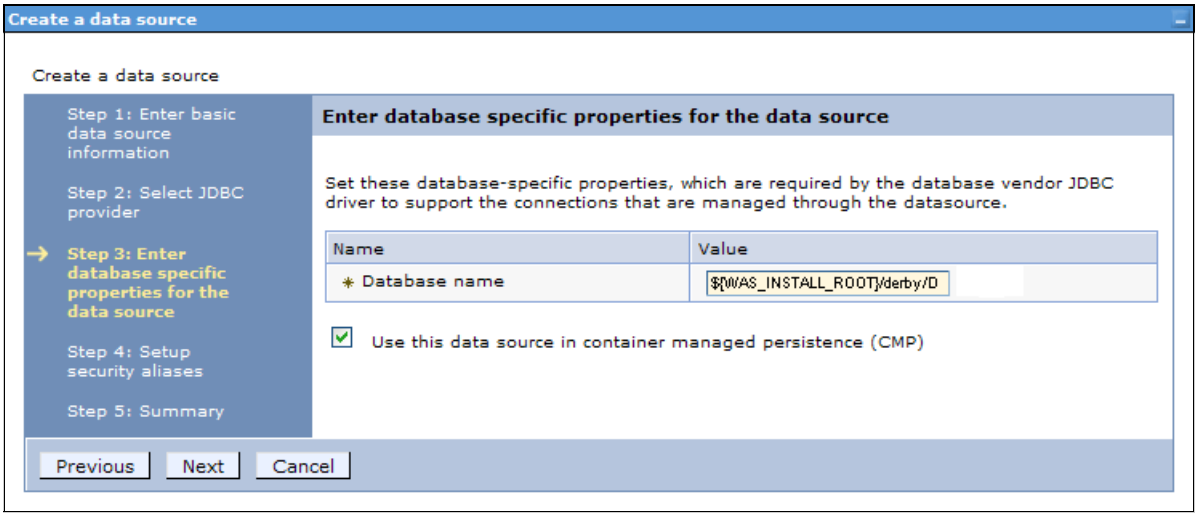


Figure 3-6 Administrative console: Create a data source

7. Choose the security settings as necessary, as shown in Figure 3-7. These settings dictate how applications access the data source. For DictionaryApp V1, accept the defaults. Click **Next**.

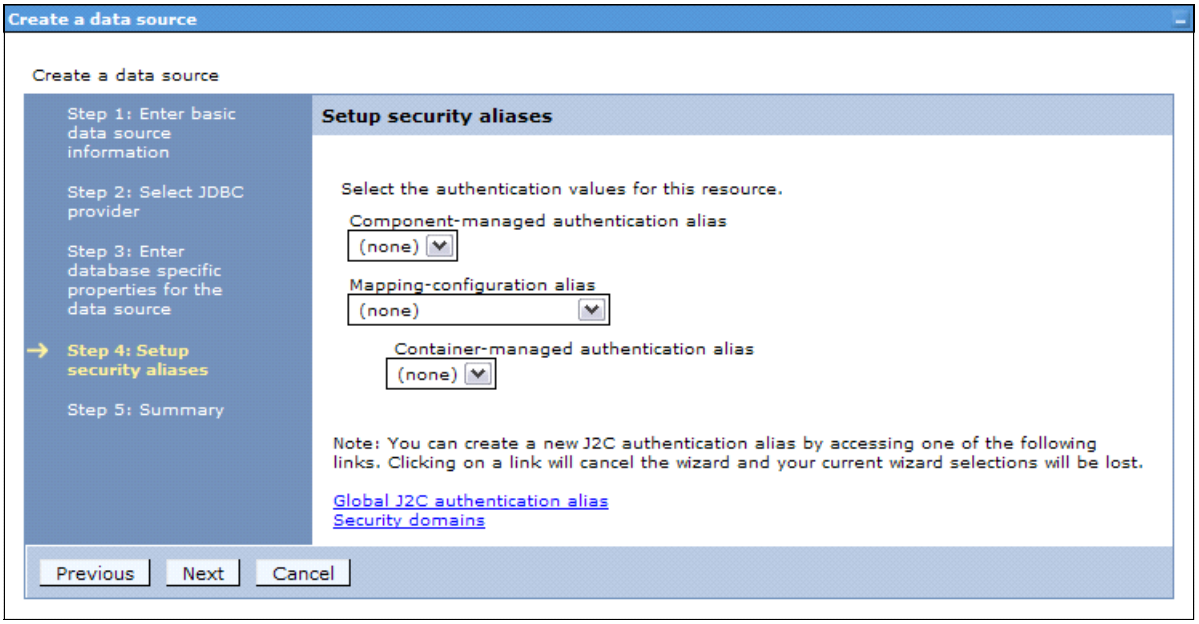


Figure 3-7 Administrative console: Create a data source

8. Review the list of selected options, as shown in Figure 3-8, and select **Finish** when you are ready to create the data source.

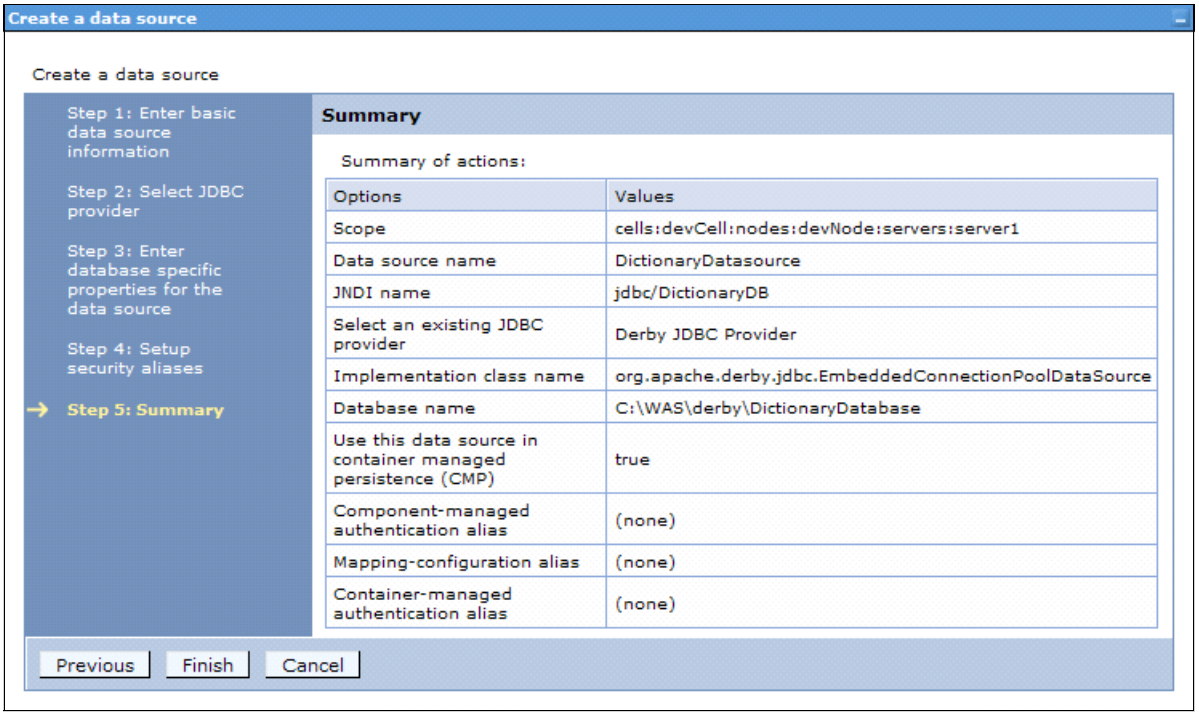


Figure 3-8 Administrative console: Create a data source summary

After creating DictionaryDatasource, you can get command assistance for creating DictionaryDatasource by clicking **View administrative scripting command for last action** in the help box. The output looks similar to Figure 3-9. Remember to return to save your changes after viewing the scripting commands.

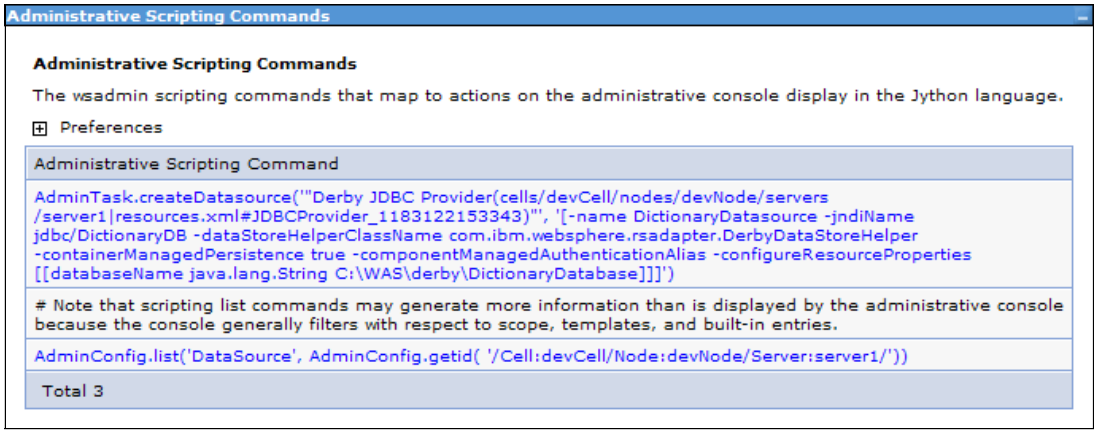


Figure 3-9 Administrative console: Administrative Scripting Commands

You can use the script that is generated through the administrative console command assistance as the template for creating future data sources.

Testing DictionaryDatasource

You can test the newly configured DictionaryDatasource through the administrative console by going to the Data sources page, selecting the target data source, and selecting the Test connection command. The application server tests whether it can establish a connection with the database that is specified under the database name. Afterward, the administrative console outputs the results of the test, as shown in Figure 3-10.

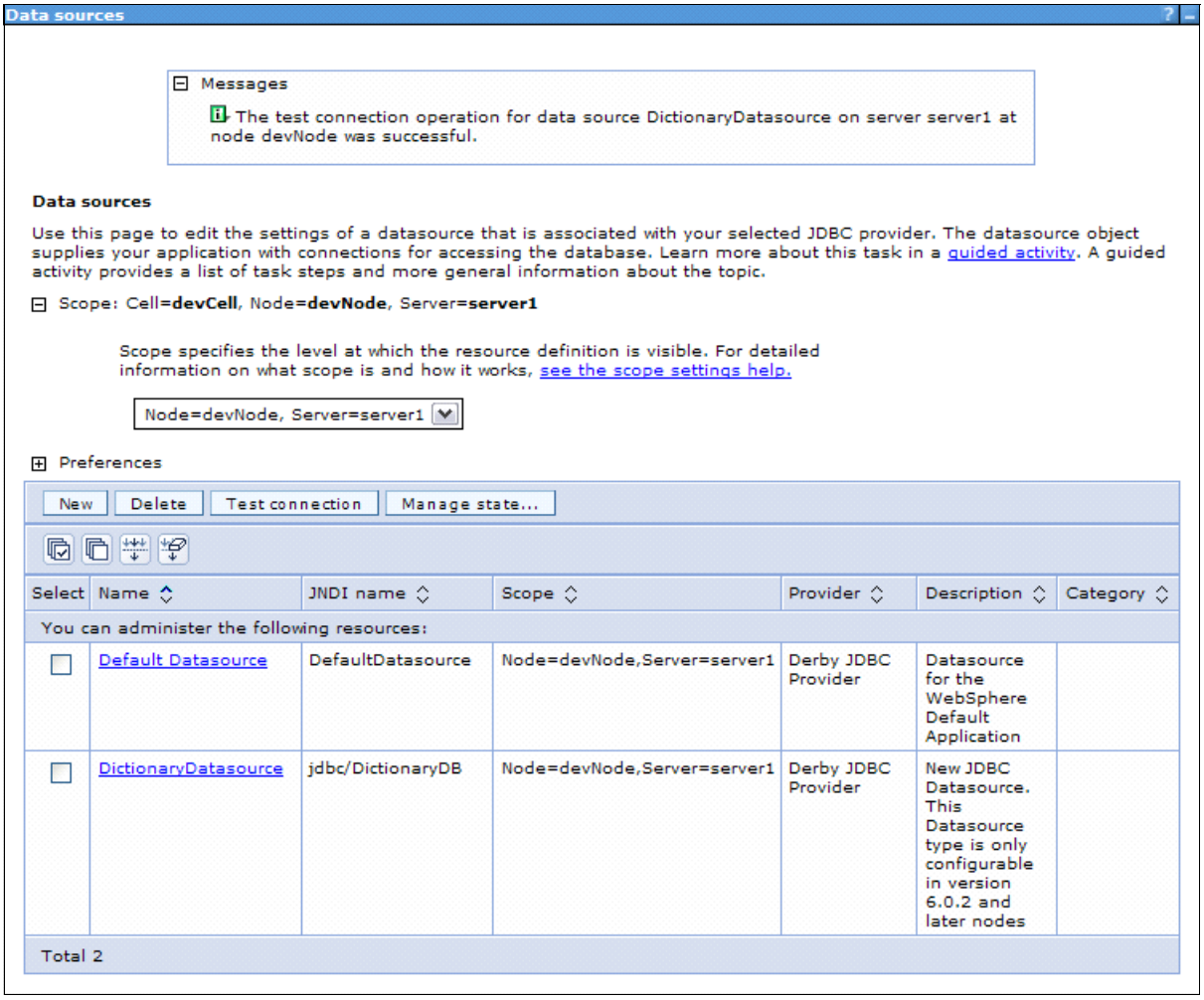


Figure 3-10 Administrative console: Data sources

Important: Test every data source that you configure before you use it to ensure that you have specified the database name correctly and that no other programs currently use the database. One of the most common reasons that the connection test fails is if you did not disconnect from the database after you configured it with Derby.

3.5.2 wsadmin scripting

You can use **`wsadmin`** to create and manage data sources. This section tells you how to create and test data sources through **`wsadmin`**.

Creating DictionaryDatasource

To create a data source in **wsadmin**, you must first obtain the configuration ID of the JDBC Provider to be used for creating the data source and assign it to a variable, for example, jdbcProvider, as shown in Example 3-7.

Example 3-7 jdbcProvider (Example 1)

```
jdbcProvider = AdminConfig.getid('/Cell:cell
name/Node:devNode/Server:server1/JDBCProvider:
JDBC provider/')
```

For DictionaryApp V1, the command looks similar to Example 3-8.

Example 3-8 jdbcProvider (Example 2)

```
jdbcProvider =
AdminConfig.getid('/Server:server1/JDBCProvider:Der
by JDBC Provider/')
```

Because Derby JDBC Provider is on the server scope, WebSphere is able to infer the node and cell scope from the server name. Afterward, create the new data source, as shown in Example 3-9.

Example 3-9 AdminTask.createDatasource (Example 1)

```
AdminTask.createDatasource(jdbcProvider, ['-
name','Datasource name','-jndiName','JNDI
name','-dataStoreHelperClassName','JDBC
Provider helper class name', '-
configureResourceProperties', [['databaseName',
'java.lang.String', 'database directory']]]);
```

For DictionaryApp V1, the command looks similar to Example 3-10.

Example 3-10 AdminTask.createDatasource (Example 2)

```
AdminTask.createDatasource(jdbcProvider, ['-
name', 'DictionaryDatasource', '-jndiName',
'jdbc/DictionaryDB','-dataStoreHelperClassName',
'com.ibm.websphere.rsadapter.DerbyDataStoreHelper
', '-configureResourceProperties',
[['databaseName', 'java.lang.String',
'${WAS_INSTALL_ROOT}/Derby/DictionaryDatabase']]]
);
AdminConfig.save();
```

Testing DictionaryDatasource

To test a data source in **wsadmin**, you must first obtain the configuration ID of the data source, as shown in Example 3-11.

Example 3-11 AdminConfig.getid (Example1)

```
datasource = AdminConfig.getid('/JDBCProvider:JDBC
provider/DataSource:data source name');
```

JDBC provider specifies the JDBC provider under which the data source was configured. For DictionaryApp V1, the command looks similar to Example 3-12.

Example 3-12 AdminConfig.getid (Example 2)

```
datasource = AdminConfig.getid('/JDBCProvider:Derby
JDBC Provider/DataSource:DictionaryDatasource');
```

Afterward, you can test the data source by invoking the following command:

```
print AdminControl.testConnection(datasource);
```

If a connection was established successfully, **wsadmin** outputs the following message:

```
WASX7217I: Connection to provided datasource was successful.
```

3.5.3 Property file-based configuration

Create a new text file named `properties.prop`. Copy and paste the text from Example 3-13 with the necessary fields defined.

Example 3-13 Properties file-based configuration (Example 1)

```
ResourceType=DataSource
ImplementingResourceType=JDBCProvider
ResourceId=Cell=!{cellName}:Node=!{nodeName}:Serv
er=!{serverName}:JDBCProvider=Derby JDBC
Provider:DataSource=jndiName#jdbc/<JNDI name>
jndiName="<JNDI name>"
name=<data source name>
ResourceType=J2EEResourcePropertySet
ImplementingResourceType=JDBCProvider
ResourceId=Cell=!{cellName}:Node=!{nodeName}:Serv
er=!{serverName}:JDBCProvider=Derby JDBC
Provider:DataSource=jndiName#<JNDI
name>:J2EEResourcePropertySet=
AttributeInfo=resourceProperties(name,value)
databaseName=<database directory>
EnvironmentVariablesSection
serverName=server1
cellName=devCell
nodeName=devNode
```

The configuration script contains two sections of configuration data. The first section represents the data source resource. This section includes the data source name, the JNDI name, and other options pertaining to the data source. The second section represents additional properties pertaining to the resource in a separate configuration object. This section includes the database name, transaction attributes, and connection variables. The configuration file that is listed Example 3-13 only contains the minimum properties to create the data source. More properties can be specified, but we have left them out for brevity.

To create DictionaryDatasource, as specified for DictionaryApp, `properties.prop` looks similar to Example 3-14.

Example 3-14 Properties file-based configuration (Example 2)

```
ResourceType=DataSource
ImplementingResourceType=JDBCProvider
ResourceId=Cell=!{cellName}:Node=!{nodeName}:Serv
er=!{serverName}:JDBCProvider=Derby JDBC
Provider:DataSource=jndiName#jdbc/DictionaryDB
jndiName="jdbc/DictionaryDB"
name=DictionaryDatasource
ResourceType=J2EEResourcePropertySet
ImplementingResourceType=JDBCProvider
ResourceId=Cell=!{cellName}:Node=!{nodeName}:Serv
er=!{serverName}:JDBCProvider=Derby JDBC
Provider:DataSource=jndiName#jdbc/DictionaryDB:J2
EEResourcePropertySet=
AttributeInfo=resourceProperties(name,value)
databaseName=${WAS_INSTALL_ROOT}\\derby\\Dictiona
ryDatabase
EnvironmentVariablesSection
serverName=server1
cellName=devCell
nodeName=devNode
```

3.6 Deploying, updating, and accessing DictionaryApp V1

The last step is to deploy DictionaryApp V1 to the application server. You can use either of these methods:

- “Administrative console” on page 81
- “wsadmin scripting” on page 94

3.6.1 Administrative console

This section tells you how to install, start and stop, view, update, and uninstall our sample application through the administrative console.

Installing DictionaryApp V1

Use the following procedure to install DictionaryApp V1 through the administrative console:

1. Prepare the application module that will be uploaded to the server. For DictionaryApp V1, you can either use the prepared EAR file that is included with this document or export your own EAR file through Eclipse, as documented in Appendix A, “Development tools reference” on page 219.
2. In the administrative console, click **Applications** → **New Application** in the navigation tree. The New Application page displays, as shown in Figure 3-11.

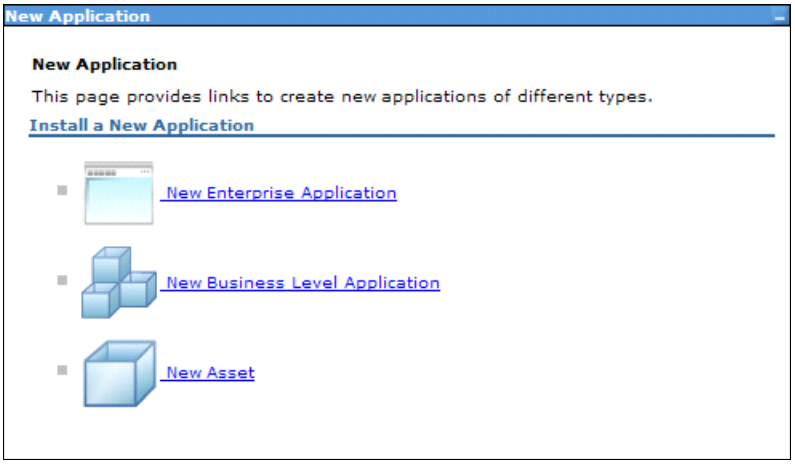


Figure 3-11 Administrative console: New Application

3. Select **New Enterprise Application**, and click **Next**.
4. Select **Local file system**, and specify the file path to where DictionaryApp V1 is located, as shown in Figure 3-12. Click **Next**.

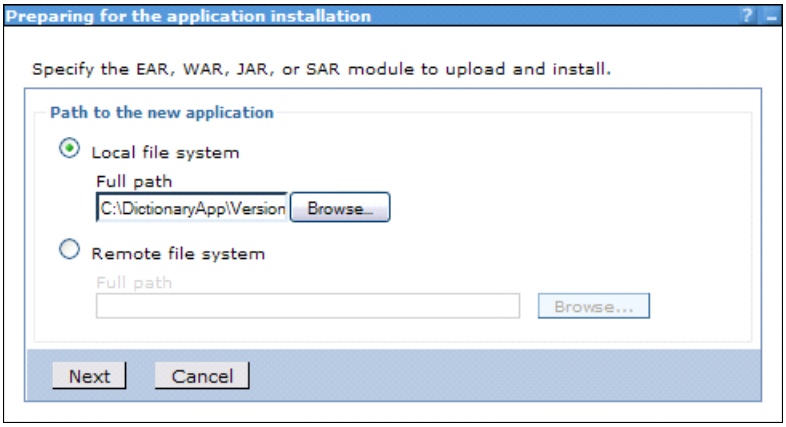


Figure 3-12 Administrative console: Preparing for the application installation (Step 1)

5. Choose **Fast Path - Prompt only when additional information is required**, as shown in Figure 3-13. Using the fast path, the application server pulls information from previous installations and deployment descriptors within the application to configure the application for deployment. Click **Next**.

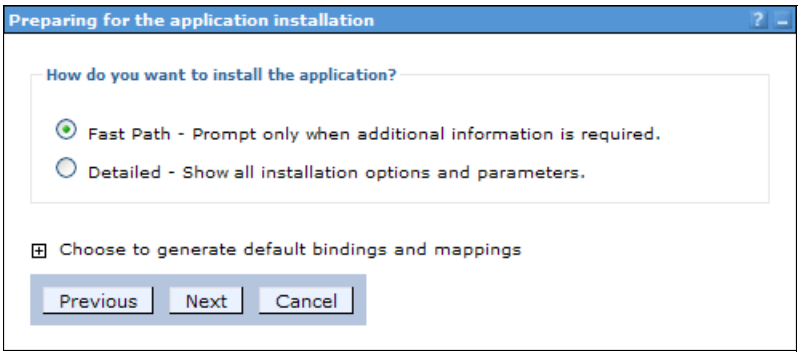


Figure 3-13 Administrative console: Preparing for the application installation (Step 2)

6. Configure the installation options as needed, as shown in Figure 3-14. For DictionaryApp V1, accept the defaults. Click **Next**.

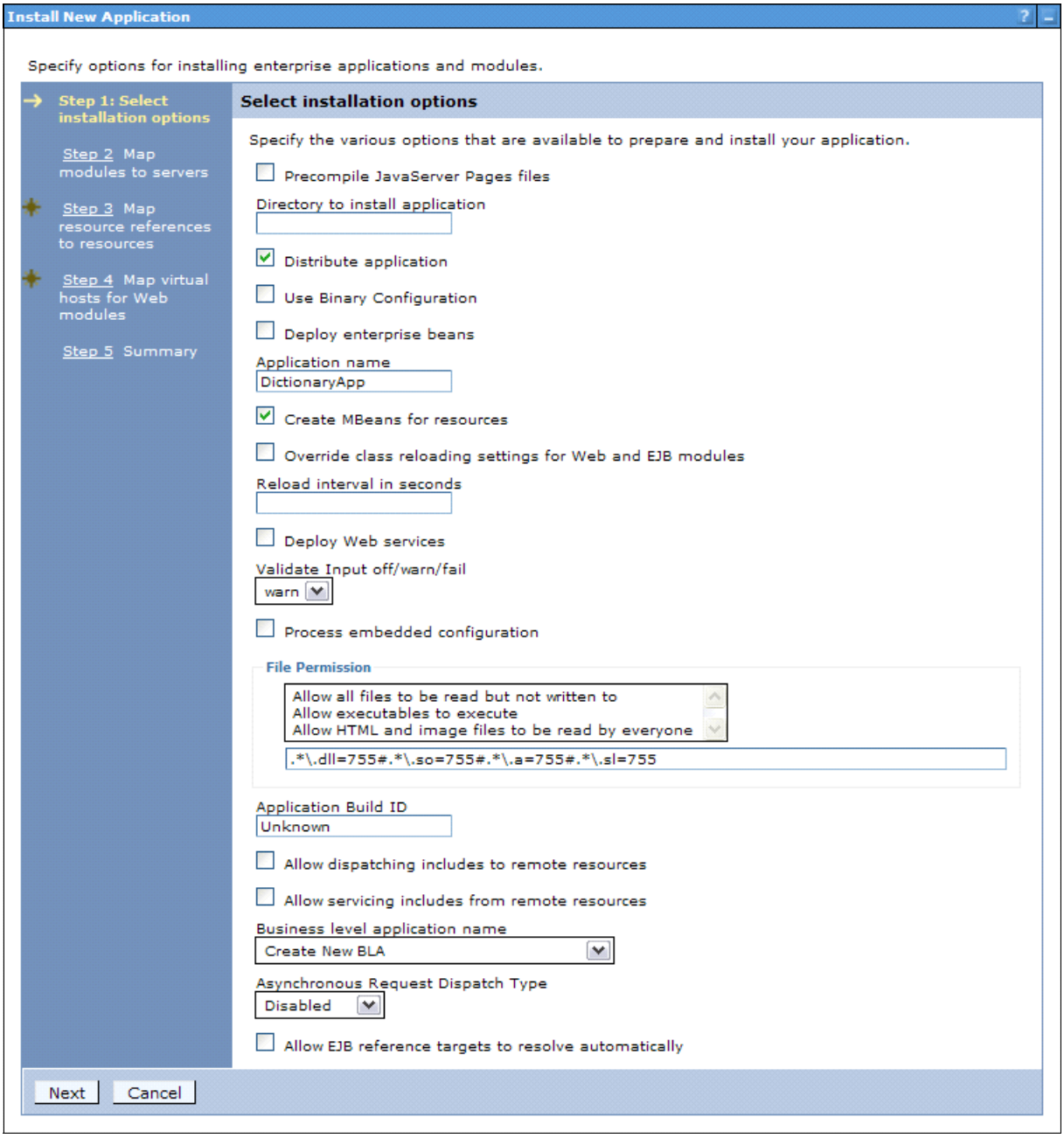


Figure 3-14 Administrative console: New application installation options

7. Select the server on which the application will run, as shown in Figure 3-15. For DictionaryApp V1, accept the defaults. Click **Next**.

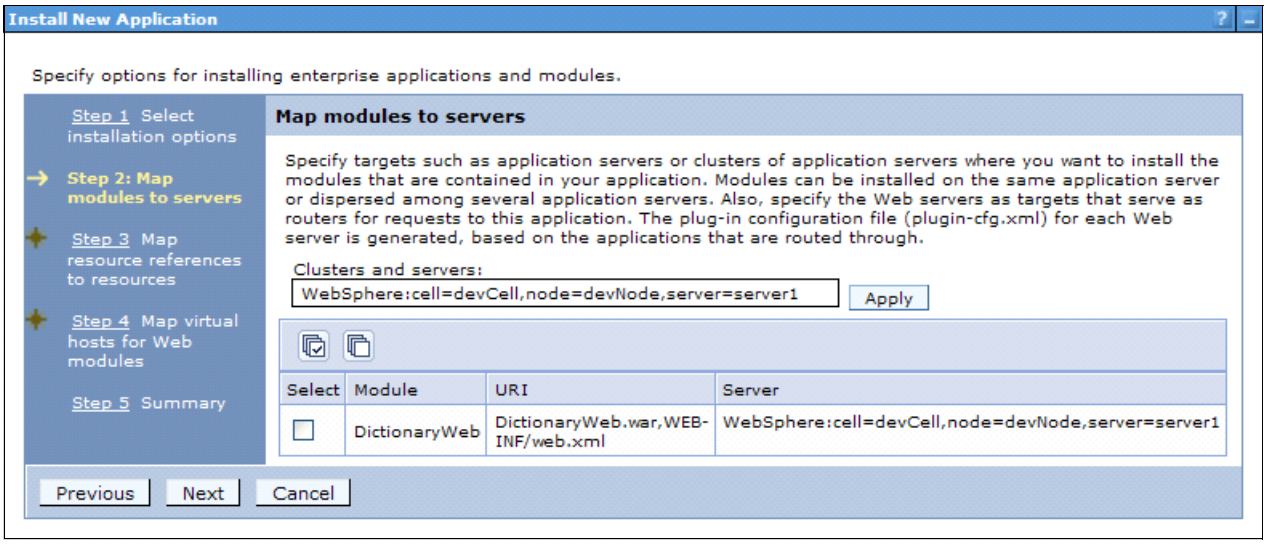


Figure 3-15 Administrative console: New application map modules to servers

8. In the Map resource references to resources section, you are prompted to specify the JNDI names for all resource references within your application, as shown in Figure 3-16. For DictionaryApp V1, use **jdbc/DictionaryDB**, the JNDI name that you specified for DictionaryDatasource, for the Target Resource JNDI Name. Click **Next**.

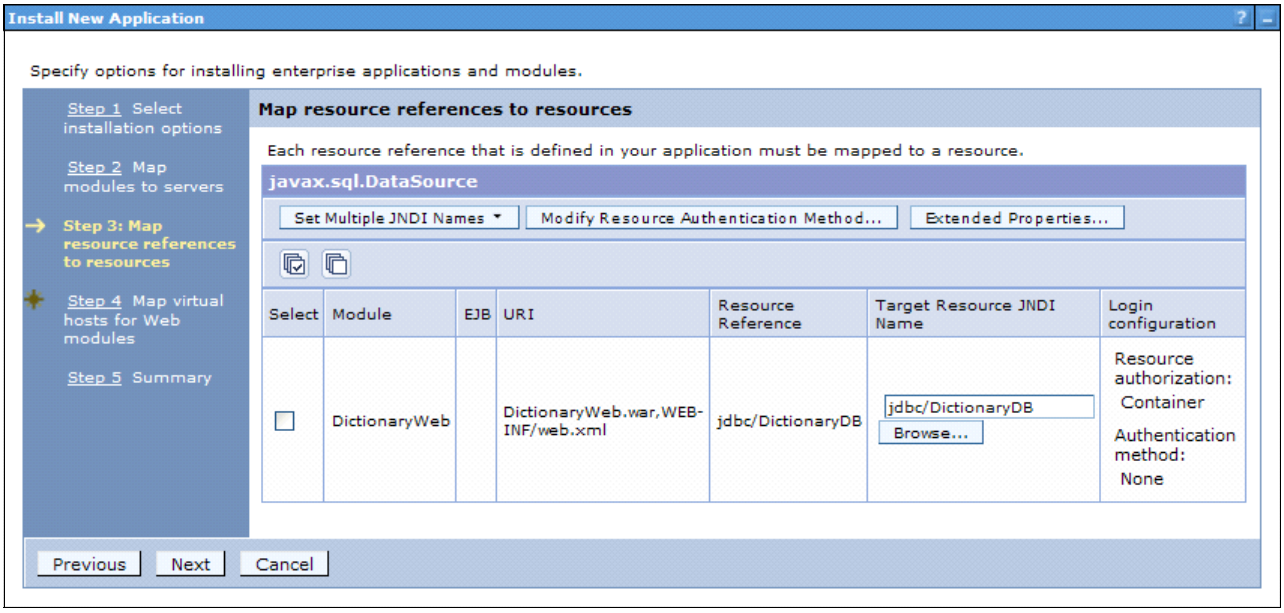


Figure 3-16 Administrative console: New application map references to resources

9. In the Map virtual host for Web modules section, you are prompted to specify the virtual host for each web module in your application, as shown in Figure 3-17. For DictionaryApp V1, accept the default, which is `default_host`. Click **Next**.

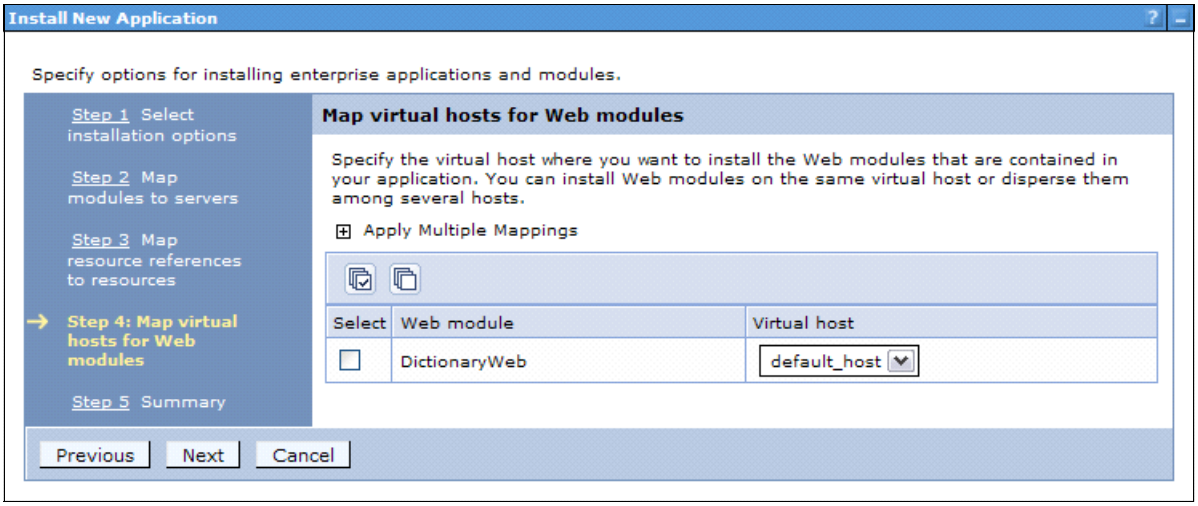


Figure 3-17 Administrative console: New application map virtual hosts for web modules

10. In the Summary, review your selections, as shown in Figure 3-18. Do not click **Finish** yet.

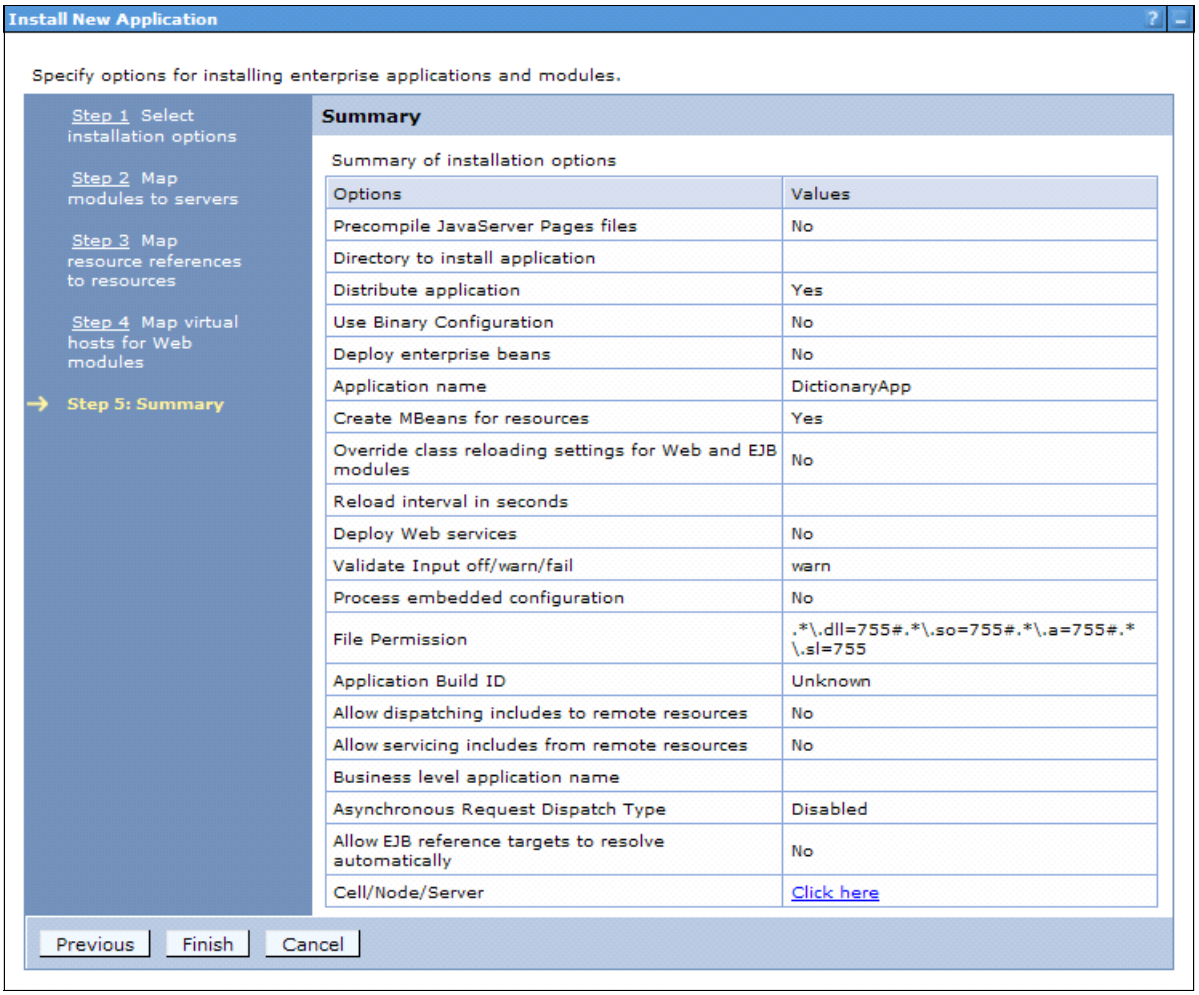


Figure 3-18 Administrative console: Installing new application summary

11. At this point, you can also get command assistance from the administrative console by clicking **View administrative scripting command for last action** in the help box. The output looks similar to Figure 3-19. Remember to return to save your changes after viewing the scripting commands.

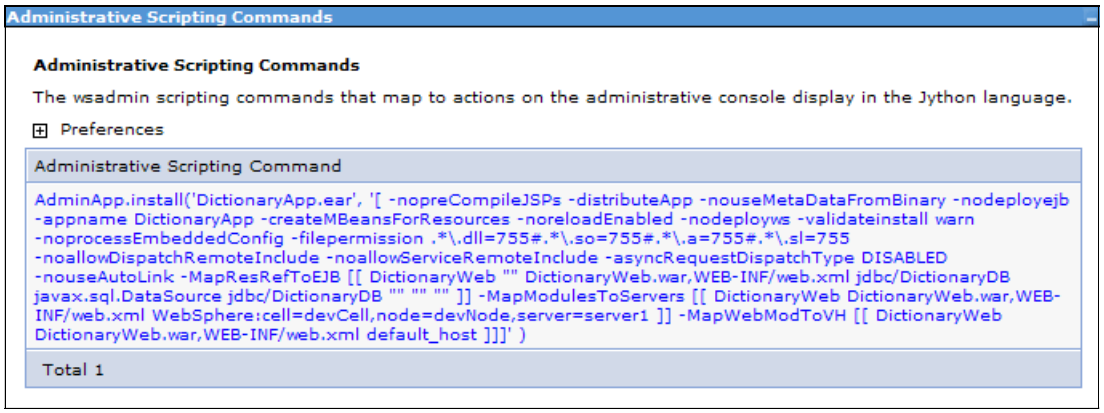


Figure 3-19 Administrative console: Administrative scripting commands

12. Click **Finish** to install the application. A new page appears where you can save your changes, as shown in Figure 3-20.

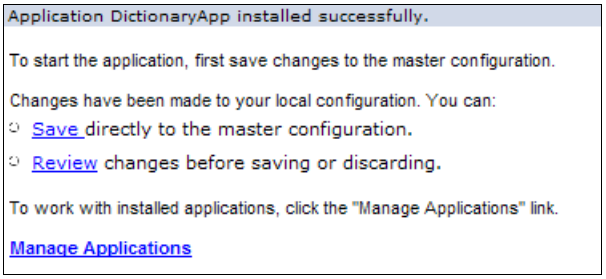


Figure 3-20 Saving DictionaryApp application changes

Starting and stopping DictionaryApp V1

Click **Applications** → **Enterprise Applications**. After installing DictionaryApp V1, it is added to Enterprise Applications, as shown in Figure 3-21.

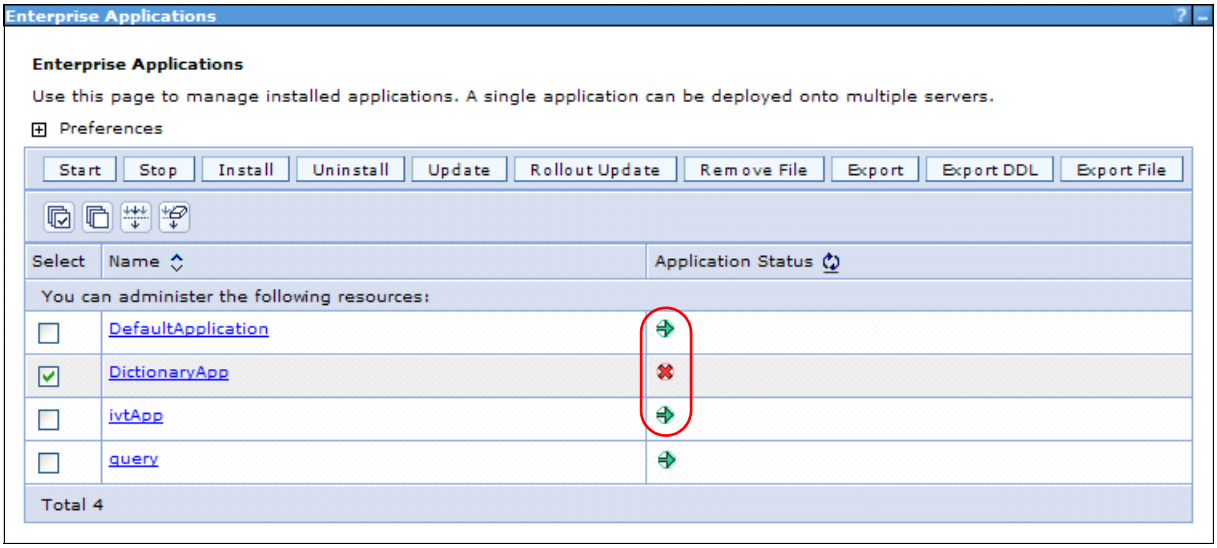


Figure 3-21 Administrative console Enterprise Applications

Note that the application is marked with a red X symbol, indicating that even though DictionaryApp V1 has been installed, it has not been started. You must start DictionaryApp to access it on the application server. Use the following procedure to start and stop the application:

1. Check the box next to DictionaryApp, and click **Start**, as shown in Figure 3-22.

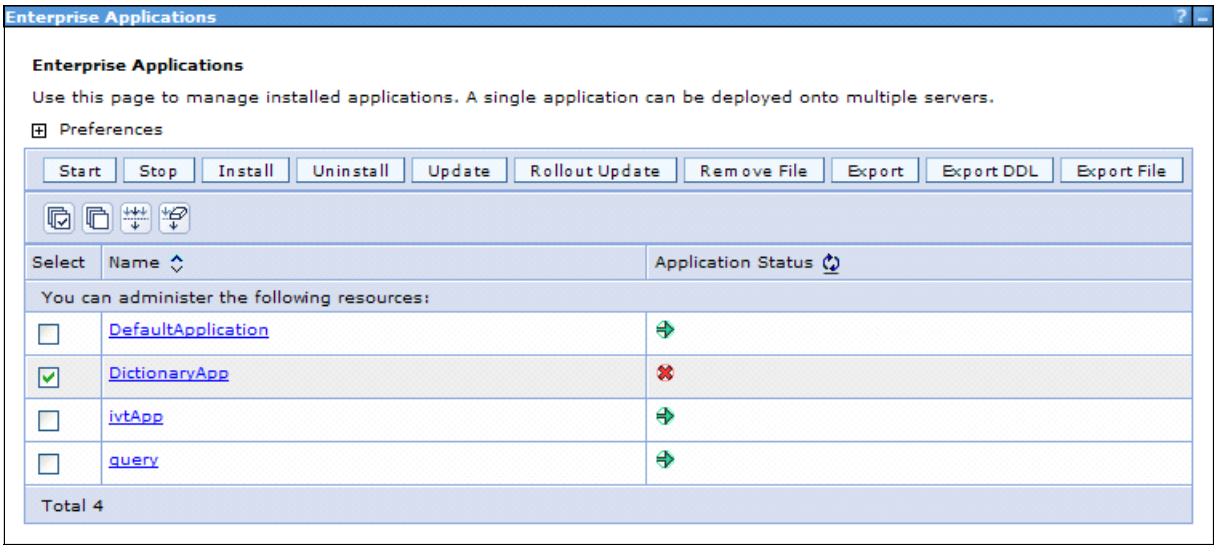


Figure 3-22 Administrative console Enterprise Applications: Starting an application

2. If the application server is able to start DictionaryApp V1 successfully, you get a message similar to Figure 3-23.

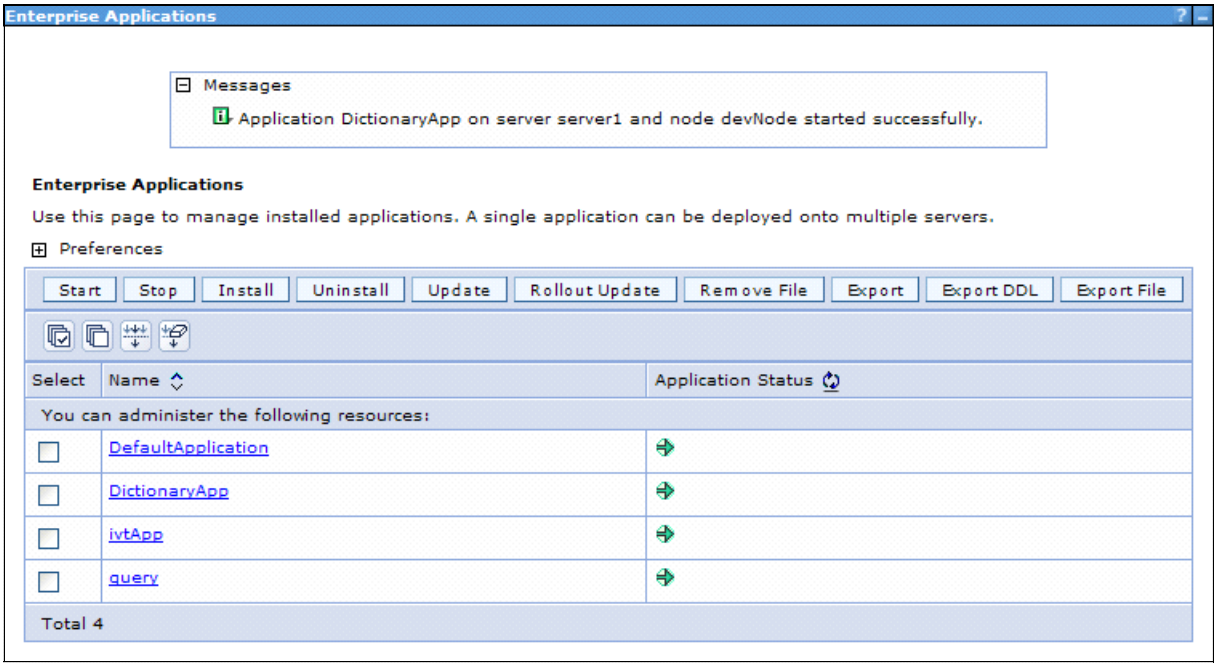


Figure 3-23 Administrative console Enterprise Applications: Showing the application started message

3. To stop an application that is running, click **Stop**.

To access the DictionaryApp, see 3.7, “Accessing DictionaryApp V1” on page 96.

Updating DictionaryApp V1

You can also update any application that you deploy on the application server. Several ways are available to update applications. In this section, we focus on updating the entire application, which is similar to uninstalling and reinstalling the application.

Before updating the application, we change `showentry.jsp` to make the color of the LOOKUP and DEFINE buttons blue. You can make this change by manually putting a new version of `showentry.jsp` in the application. Or, you can import the application to Eclipse to make the changes and export a revised copy of the application to be updated. The steps to set up an Eclipse environment are in Appendix A, “Development tools reference” on page 219.

The change is straightfoward. Simply add `"color: blue;"` to the style attribute of each input, as shown in Example 3-15.

Example 3-15 Changing style attributes

```
<tr>n
<td>Enter word:</td>
<td><input name="word" type="text" size="40"
value="<%=request.getAttribute("word") == null ? "" : request
.getAttribute("word")%>">
<input type="submit" name="action" value="LOOKUP"
style="color: blue; height: 24px; width: 100px"></td>
</tr>
<tr>
<td>Enter definition:</td>
<td><textarea name="definition" rows="10" cols="30"><
%=request.getAttribute("definition") == null ? "" : request
.getAttribute("definition")%></textarea> <input
type="submit" name="action"
value="DEFINE" style="color: blue; height: 24px; width:
100px"></td>
</tr>
```

Use the following procedure to update the DictionaryApp V1 application:

1. Go to the administrative console. Click **Applications** → **Application Types** → **WebSphere enterprise applications** in the navigation tree. The Enterprise Application page displays.
2. Check the box next to DictionaryApp and click **Update**, as shown in Figure 3-24.

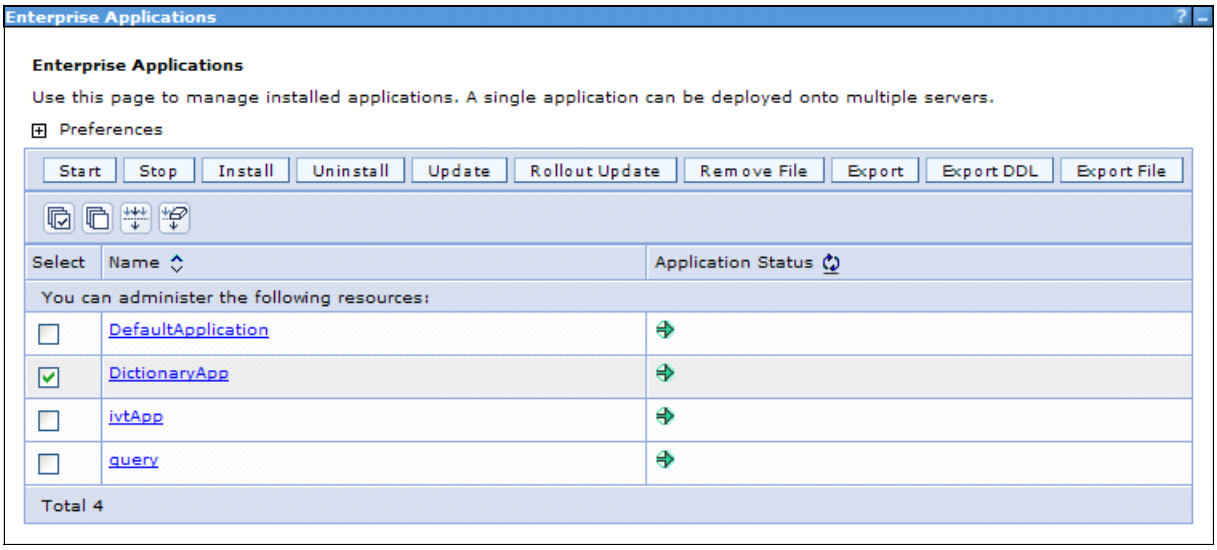


Figure 3-24 Administrative console Enterprise Applications: Updating an application

3. Select **Replace the entire application** and **Local file system**. Specify the file path to the newly updated application module, as shown in Figure 3-25. Click **Next**.

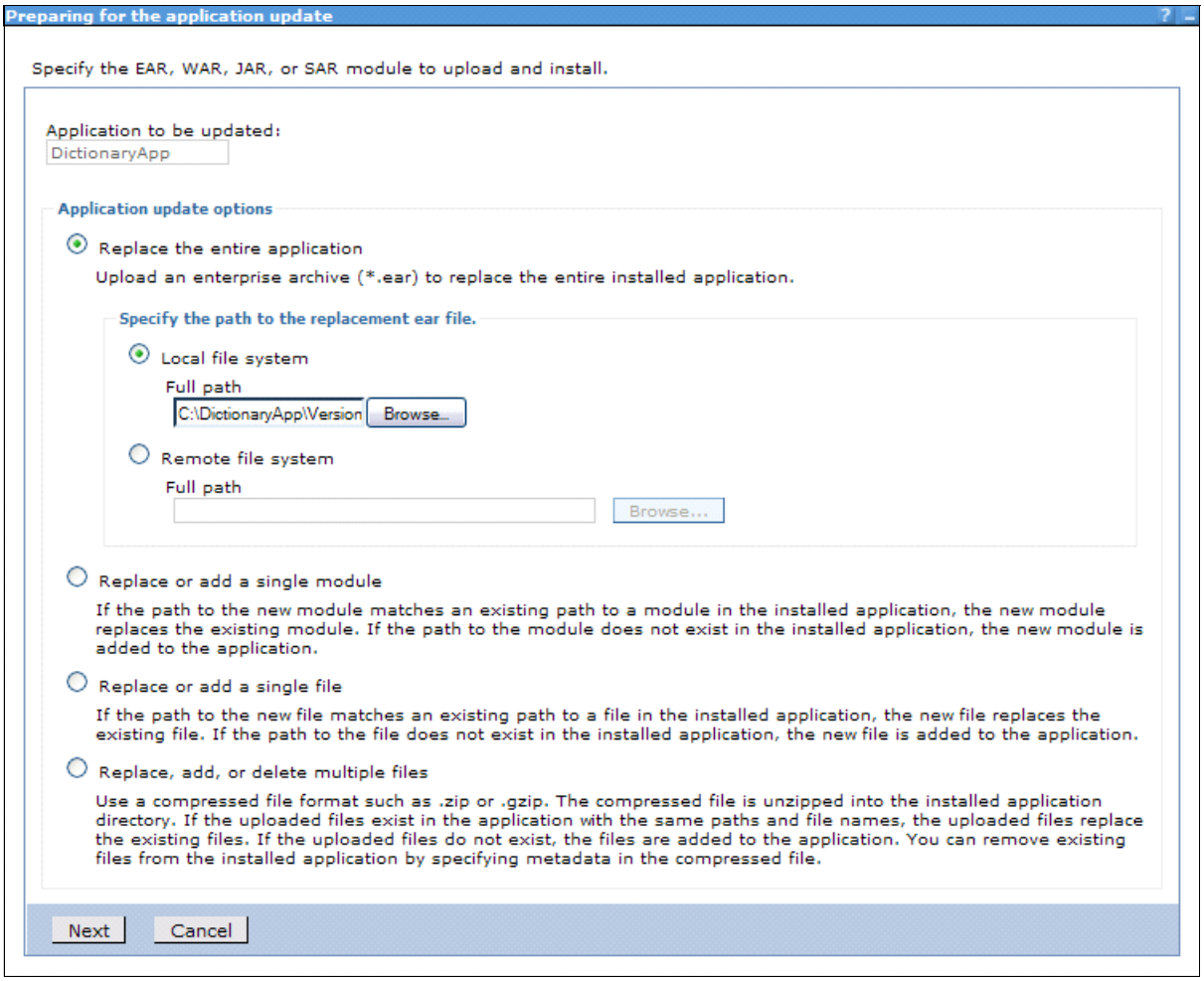


Figure 3-25 Administrative console Enterprise Applications: Replacing an application

4. After you click **Next**, complete the remainder of the procedure by going to Step 5 on page 83.

If the installed application was running before you updated it, the application server attempts to start the application after the update. Now, you can access the application URL to view the new updated application.

Uninstalling DictionaryApp V1

You can uninstall an application through the administrative console. Use the following procedure to uninstall DictionaryApp:

1. Go to the administrative console. Click **Applications** → **Application Types** → **WebSphere enterprise applications** in the navigation tree. The Enterprise Application page displays.
2. Check the box next to DictionaryApp and click **Uninstall**, as shown in Figure 3-26.

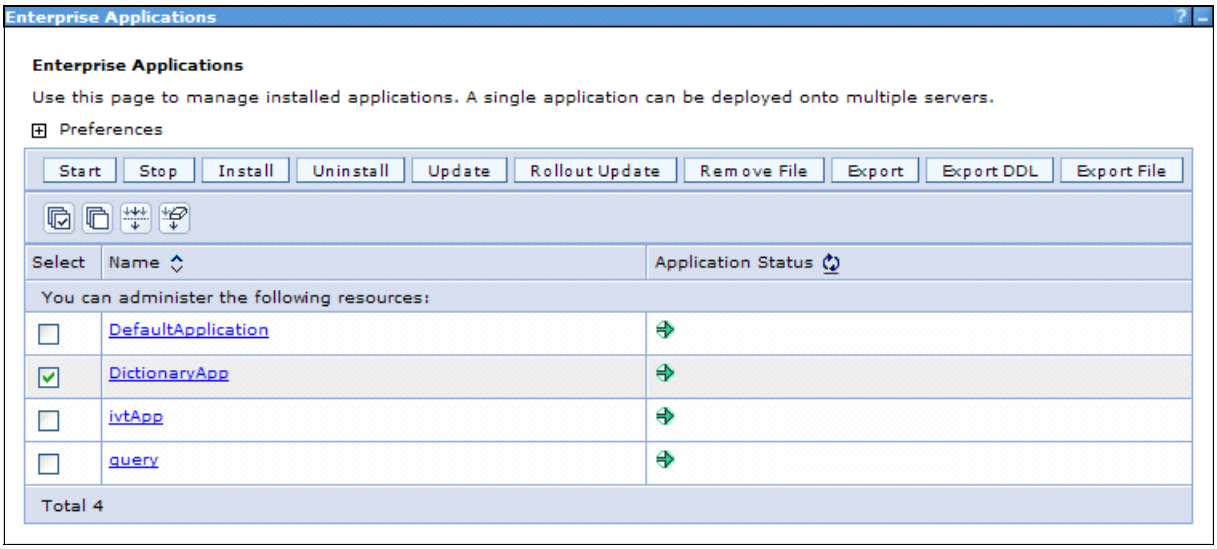


Figure 3-26 Administrative console Enterprise Applications: Uninstalling an application

3. Confirm the uninstall by clicking **OK**, as shown in Figure 3-27.

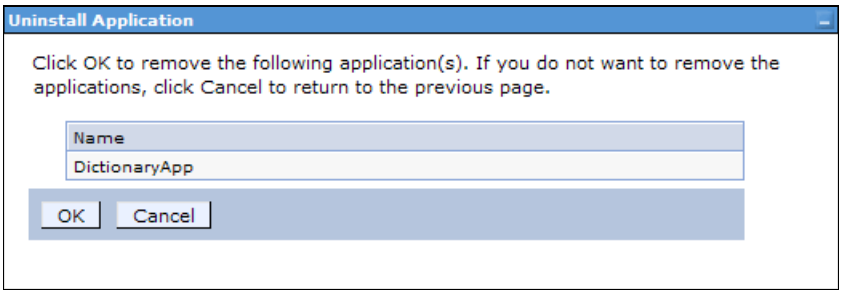


Figure 3-27 Administrative console Enterprise Applications: Confirming the application uninstall

4. You return to the Enterprise Applications page. DictionaryApp has been removed, as shown in Figure 3-28. Click **Save** to confirm your changes.

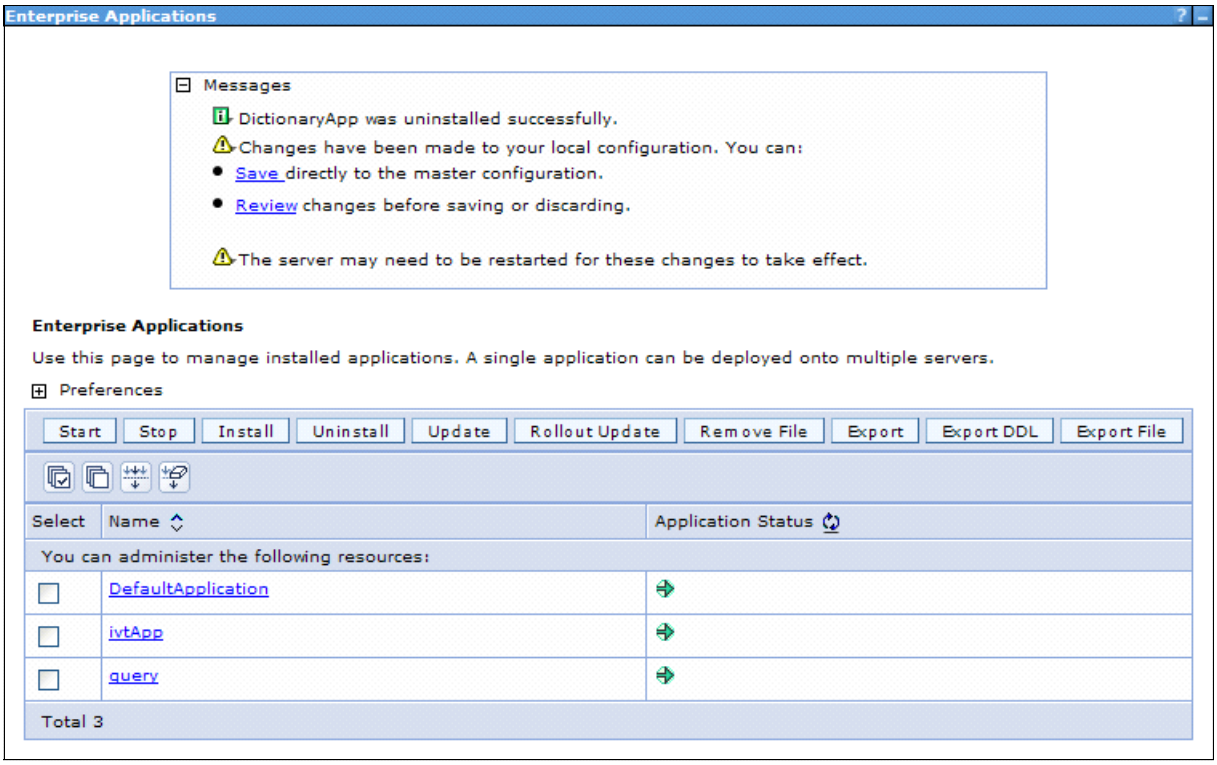


Figure 3-28 Administrative console Enterprise Applications: Application was uninstalled successfully message

3.6.2 wsadmin scripting

You can install and manage applications through **wsadmin**. If you are already familiar with the deployment process, it is not necessary to go through the administrative console each time that an update or install is needed. This section describes how to install, start and stop, and update the sample application. It also explains how to check the status of our sample application and uninstall our sample application through **wsadmin**.

Installing DictionaryApp V1

To install an application in **wsadmin**, use the following command:

```
AdminApp.install(module filepath,[options(optional)])
```

The *options(optional)* parameter specifies the options for installing the application. For a complete list of these options, go to this website:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/rxml_taskoptions.html

The following options are the most commonly used options:

'-server *server*'	Specifies on which server to install the application.
'-cell *cellname*'	Specifies on which cell to install the application.
'-node *nodeName*'	Specifies on which node to install the application.
'-appname *name*'	Specifies under what name the application is installed. It must include the desired application name as the next argument in the list.

Install options vary from application to application, depending on the resource dependencies and security settings. If you have already installed an application using the administrative console, you can obtain command assistance for the command to be used in **wsadmin**. In general, for applications that reference a data source resource on the application server, the install script must be in the format that is shown in Example 3-16.

Example 3-16 AdminApp.install

```
AdminApp.install('application file path',['-appname', 'application name',
'-MapResRefToEJB', [[Module Name,EJB Name,URI,Reference Binding,Resource Type,JNDI
name,Login Configuration,Properties]],'-MapWebModToVH', [[Web Module Name,
URI,Host Name]]]);
```

The `-MapResRefToEJB` options allow you to map resource references to resources within Enterprise JavaBeans (EJB) and web modules. In this version of DictionaryApp, we use it to map data source references within the web module DictionaryWeb to DictionaryDatasource. You must provide the necessary parameters that define the mapping afterward in the list format:

Module Name	Name of the EJB or web module that contains the EJB or web module that references the resource
EJB Name	EJB name or “”
URI	Module URI for EJB module
Reference Binding	How the resource is referenced in the application
Resource Type	Type of resource that is defined by the Java class path
JNDI name	JNDI name of the mapped resource
Login Configuration	Option to create a custom login configuration
Properties	Option to specify special properties

The `-MapWebModToVH` options allow you to map web modules to virtual hosts in the application server. You must provide the necessary parameters that define the mapping afterward in the list format:

Module Name	Name of the web module that will be mapped to the virtual host
URI	Module URI for web module
Host Name	Host name to which the Web Module is to be mapped

For DictionaryApp V1, the command looks similar to Example 3-17.

Example 3-17 AdminApp.install

```
AdminApp.install('C:/DictionaryApp/version
1/DictionaryApp.ear',['-appname',
'DictionaryApp', '-
MapResRefToEJB','[[DictionaryWeb ""
DictionaryWeb.war,WEB-INF/web.xml
jdbc/DictionaryDB javax.sql.DataSource
jdbc/DictionaryDB "" "" ""]]','-
MapWebModToVH','[[DictionaryWeb
DictionaryWeb.war,WEB-INF/web.xml
default_host]]']);
AdminConfig.save();
```

Starting and stopping DictionaryApp V1

To start DictionaryApp, use the ApplicationManager MBean, as shown in Example 3-18.

Example 3-18 wsadmin start an application

```
AppManager =
AdminControl.queryNames('type=ApplicationManager,
process=server1,*')
AdminControl.invoke(appManager,
'startApplication', 'DictionaryApp')
```

To stop the application, call the stopApplication method.

Updating DictionaryApp V1

To update DictionaryApp, use the example that is shown in Example 3-19.

Example 3-19 wsadmin update an application

```
AdminApp.update('DictionaryApp','app',['-
operation', 'update', '-contents',
'C:/DictionaryApp/Version 1/DictionaryApp.ear']);
AdminConfig.save();
```

Checking the state of DictionaryApp V1

To check if DictionaryApp is running, check for its Application MBean, as shown in Example 3-20. If it is running, the MBean ObjectName is returned.

Example 3-20 wsadmin check the application state

```
AdminControl.completeObjectName('type=Application
, name=DictionaryApp')
```

Uninstalling Dictionary App V1

To uninstall DictionaryApp, use the example that is shown in Example 3-21.

Example 3-21 wsadmin uninstall an application

```
AdminApp.uninstall('DictionaryApp');
AdminConfig.save();
```

Note: All install, start, update, and uninstall operations are logged in the `SystemOut.log` file. Refer to this log for debugging information concerning your application.

3.7 Accessing DictionaryApp V1

You can access DictionaryApp V1 at the following URL:

`http://localhost:9080/DictionaryWeb/DictionaryServlet`

DictionaryApp V1 needs to look similar to Figure 3-29 on page 97.

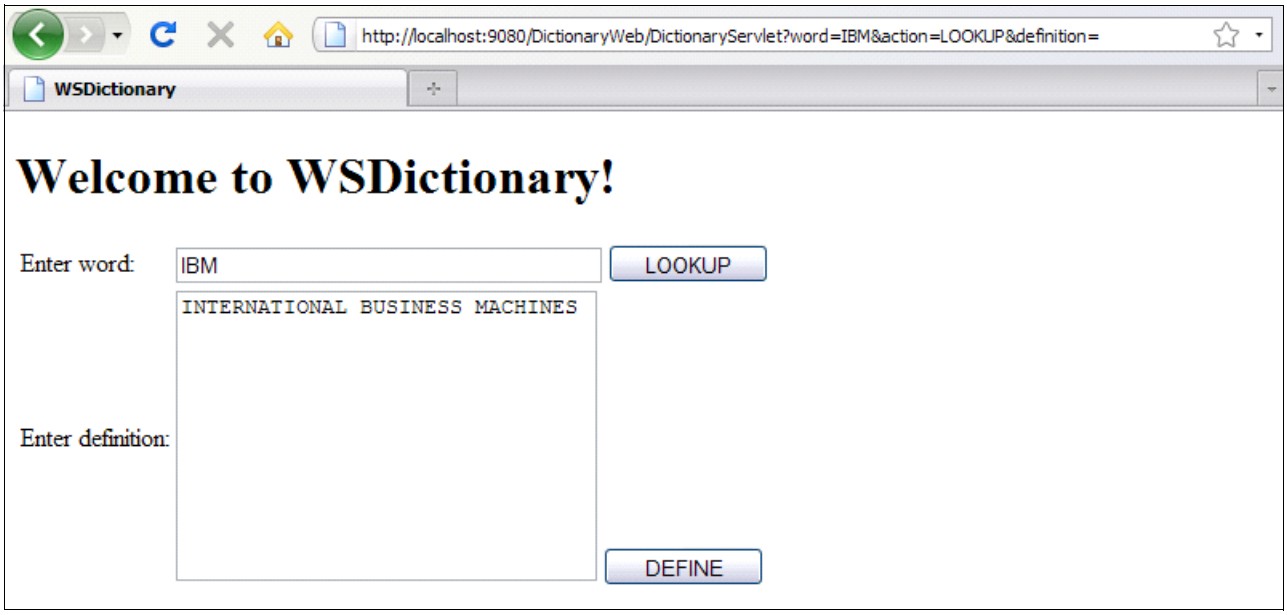


Figure 3-29 DictionaryApp V1

4

Incorporating EJB3 and JPA into DictionaryApp V2

In this chapter, we use the Enterprise JavaBeans 3 (EJB3) and Java Persistence API (JPA) framework to further enhance DictionaryApp V1 to illustrate how WebSphere handles the EJB and JPA artifacts. WebSphere also provides support for previous versions of EJB. If you are more comfortable with a previous version of EJB, skip the annotations marked in the sample code and create your own deployment descriptors for the provided code.

We incorporate EJB3 and the JPA into DictionaryApp V2. JPA offers an alternative to Java Database Connectivity (JDBC) by providing a further abstraction layer on top of JDBC between the business logic and database setup. EJB provides an architecture for scalability and reliability within the enterprise application. We increase the functionalities that are provided in DictionaryApp V1 by implementing them through JPA. DictionaryApp V2 is enhanced to allow users to choose to perform LOOKUP and DEFINE with either JDBC or JPA.

Sample material: See Appendix C, “Additional material” on page 235 for information about downloading the sample material that is used in this chapter.

JPA implementation: The examples in this chapter use the Apache OpenJPA implementation of JPA. You can use whichever implementation you want. You can specify these configurations in `persistence.xml`.

© Copyright IBM Corp. 2010. All rights reserved.

4.1 DictionaryApp V2 overview

Figure 4-1 illustrates the overall architecture of DictionaryApp V2.

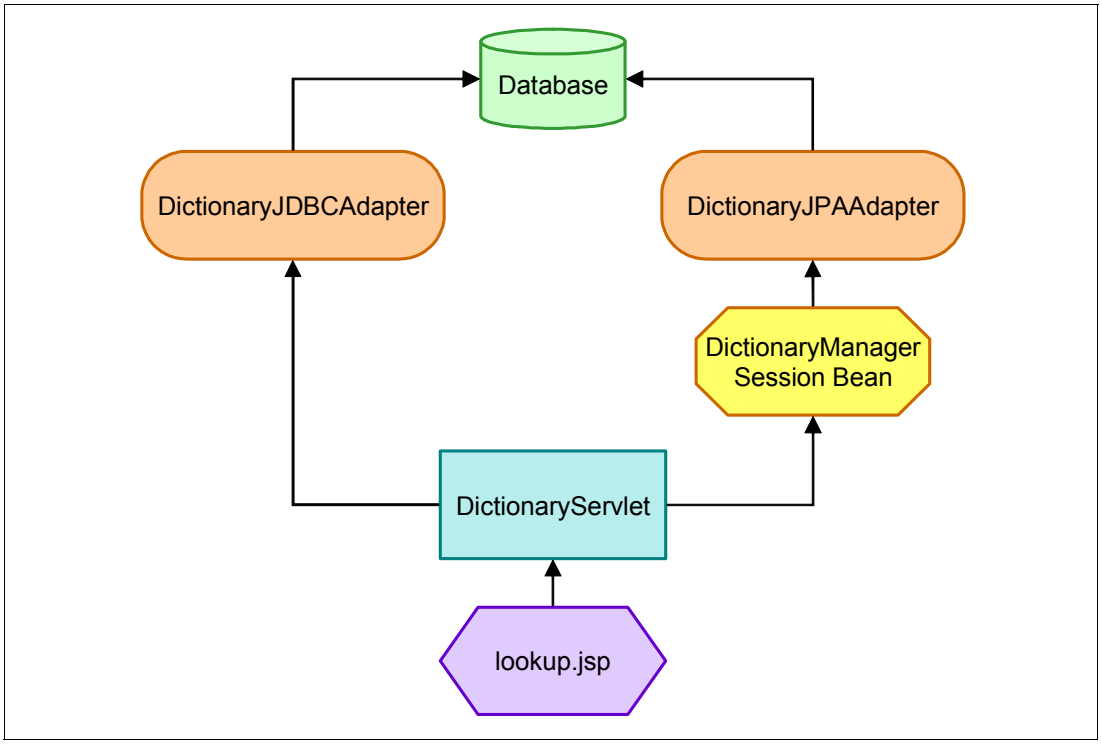


Figure 4-1 DictionaryApp V2

For DictionaryApp V2, we use Entry.java as the sole JPA Entity in the application. We also implement DictionaryJPAAdapter.java to complement DictionaryJDBCAdapter.java as a wrapper that provides access to DictionaryDatabase through JPA.

We also implement DictionaryManagerBean.java, a stateless session bean, which serves as the domain layer mediator between DictionaryServlet.java and business logic. For Dictionary V2, it only supports access to JPA functionality, but in future versions of DictionaryApp, it will be enhanced with additional function.

4.2 DictionaryApp source files

This section provides source text for the following files that you use to build DictionaryApp V2:

- "DictionaryManagerLocal.java" on page 101
- "DictionaryManagerBean.java" on page 101
- "DictionaryJPAAdapter.java" on page 102
- "Entry.java" on page 103
- "DictionaryServlet.java" on page 103
- "showentry.jsp" on page 104
- "persistence.xml" on page 104

4.2.1 DictionaryManagerLocal.java

DictionaryManagerLocal.java is the local EJB interface through which DictionaryServlet.java accesses DictionaryManagerBean.java. This interface extends DictionaryDatabaseAdapter.java, which allows DictionaryServlet.java to easily switch between database access implementations.

Example 4-1 shows the text of DictionaryManagerLocal.java.

Example 4-1 DictionaryManagerLocal.java

```
package com.ibm.dictionaryapp.ssb;

import java.sql.SQLException;

import javax.ejb.Local;

import com.ibm.dictionaryapp.database.DictionaryDatabaseAdapter;
import com.ibm.dictionaryapp.database.Entry;

@Local
public interface DictionaryManagerLocal extends DictionaryDatabaseAdapter {
   public void define(Entry entry) throws SQLException;

   public Entry lookup(String word) throws SQLException;
```

4.2.2 DictionaryManagerBean.java

DictionaryManagerBean.java is the implementation of the local EJB interface DictionaryManagerLocal that serves as the mediator between DictionaryServlet.java and JPAAdapter.java. Example 4-2 shows the text of DictionaryManagerBean.java.

Example 4-2 DictionaryManagerBean.java

```
package com.ibm.dictionaryapp.ssb;

import java.sql.SQLException;

import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;

import com.ibm.dictionaryapp.database.DictionaryJPAAdapter;
import com.ibm.dictionaryapp.database.Entry;

@Stateless
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public class DictionaryManagerBean implements DictionaryManagerLocal {
   private DictionaryJPAAdapter jpaadapter;

   public DictionaryManagerBean() {
      jpaadapter = new DictionaryJPAAdapter();
   }

   public void define(Entry entry) throws SQLException {
```

```
      jpaadapter.define(entry);
   }

   public Entry lookup(String word) throws SQLException {
      return jpaadapter.lookup(word);
   }
}
```

4.2.3 DictionaryJPAAdapter.java

DictionaryJPAAdapter.java is the JPA entities manager. This class uses Application-Managed Persistence to access DictionaryDatabase. Example 4-3 shows the text of DictionaryJPAAdapter.java.

Example 4-3 DictionaryJPAAdapter.java

```
package com.ibm.dictionaryapp.database;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class DictionaryJPAAdapter implements DictionaryDatabaseAdapter {
   private EntityManagerFactory entitymanagerfactory;

   public DictionaryJPAAdapter() {
      entitymanagerfactory = Persistence
            .createEntityManagerFactory("DictionaryDatabase");
   }

   public void define(Entry entry) {
      EntityManager entitymanager = entitymanagerfactory
            .createEntityManager();
      EntityTransaction entitytransaction = entitymanager.getTransaction();
      entitytransaction.begin();
      if (entitymanager.find(Entry.class, entry.getWord()) == null) {
         entitymanager.persist(entry);
      } else {
         entitymanager.merge(entry);
      }
      entitytransaction.commit();
   }

   public Entry lookup(String word) {
      EntityManager entitymanager = entitymanagerfactory
            .createEntityManager();
      EntityTransaction entitytransaction = entitymanager.getTransaction();
      entitytransaction.begin();
      Entry entry = entitymanager.find(Entry.class, word);
      entitytransaction.commit();
      return entry;
   }
}
```

4.2.4 Entry.java

This file is updated with annotations to convert `Entry.java` into a JPA entity. Note that the object relational mappings (ORMs) that are specified by the annotations match the way in which dictionary entries are organized in DictionaryDatabase. This way, we can make JPA and JDBC interchangeable within the application.

Example 4-4 shows the text of `Entry.java`.

Example 4-4 Entry.java

```
package com.ibm.dictionaryapp.database;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "DICTIONARY")
public class Entry {
   @Id
   @Column(name = "WORD")
   private String word;

   @Column(name = "DEFINITION")
   private String definition;

   public Entry() {
   }
```

4.2.5 DictionaryServlet.java

This file is updated to process requests for JPA DEFINE and JPA LOOKUP. Example 4-5 shows the text of `DictionaryServlet.java`.

Example 4-5 DictionaryServlet.java

```
...

@Resource(name = "jdbc/DictionaryDB")
public DataSource datasource;

@EJB
private DictionaryManagerLocal dictionarymanager;

private DictionaryJDBCAdapter jdbcadapter;

...

String action = request.getParameter("action");
String option = request.getParameter("option");
DictionaryDatabaseAdapter databaseadapter = (option != null && option
      .equals("JPA")) ? dictionarymanager : jdbcadapter;
try {
```

```
   if (action == null) {
      action = "NO ACTION";

...
```

4.2.6 showentry.jsp

This file is updated with functionalities for users to choose between JPA and JDBC. Example 4-6 shows the text of `Showentry.jsp`.

Example 4-6 Showentry.jsp

```
...

   <input type="submit" name="action" value="LOOKUP"
      style="height: 24px; width: 100px"></td>
   <td><input type="checkbox" name="option" value="JPA" /> JPA</td>
</tr>
<tr>
   <td>Enter definition:</td>

...
```

4.2.7 persistence.xml

This file is the JPA descriptor for DictionaryApp. Example 4-7 shows the text of `persistence.xml`.

Example 4-7 JPA descriptor for DictionaryApp

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0">
   <persistence-unit name = "DictionaryDatabase">
      <jta-data-source>jdbc/DictionaryDB</jta-data-source>
      <class>com.ibm.dictionaryapp.database.Entry</class>
   </persistence-unit>
</persistence>
```

4.3 Redeploying DictionaryApp V2

Because DictionaryApp V2 uses DictionaryDatasource (as specified by <*jta-data-source*>jdbc/DictionaryDB</*jta-data-source*> in `persistence.xml`), no resource configurations are necessary for application deployment. You merely need to redeploy and view DictionaryApp V2 on the application server by using either the administrative console or **`wsadmin`**.

4.3.1 Administrative console redeployment

Like DictionaryApp V1, you can choose to redeploy DictionaryApp V2 through the administrative console. With the Fast Path deployment option, the application server provides you with the exact same choices as DictionaryApp V1, because the application server is able

to pull the default properties for EJBs and JPA mappings from annotations and deployment descriptors.

For this deployment, we deploy through the detailed installation path to see the other options that are offered by the application during deployment. To redeploy DictionaryApp V2, you can either update DictionaryApp V1 or uninstall and then reinstall DictionaryApp V1. Both methods yield the same results.

Use the following procedure to deploy DictionaryApp V2 through the detailed installation path after you have specified the application file path to update or reinstall:

1. In the Preparing for the application installation page, select **Detailed**, and click **Next**, as shown in Figure 4-2.

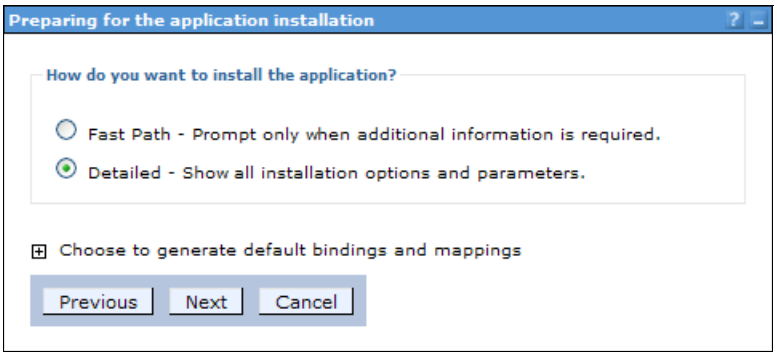


Figure 4-2 Administrative console: Preparing for the application installation

2. Configure the installation options as needed, as shown in Figure 4-3. For DictionaryApp V2, leave the options blank. Click **Next**.

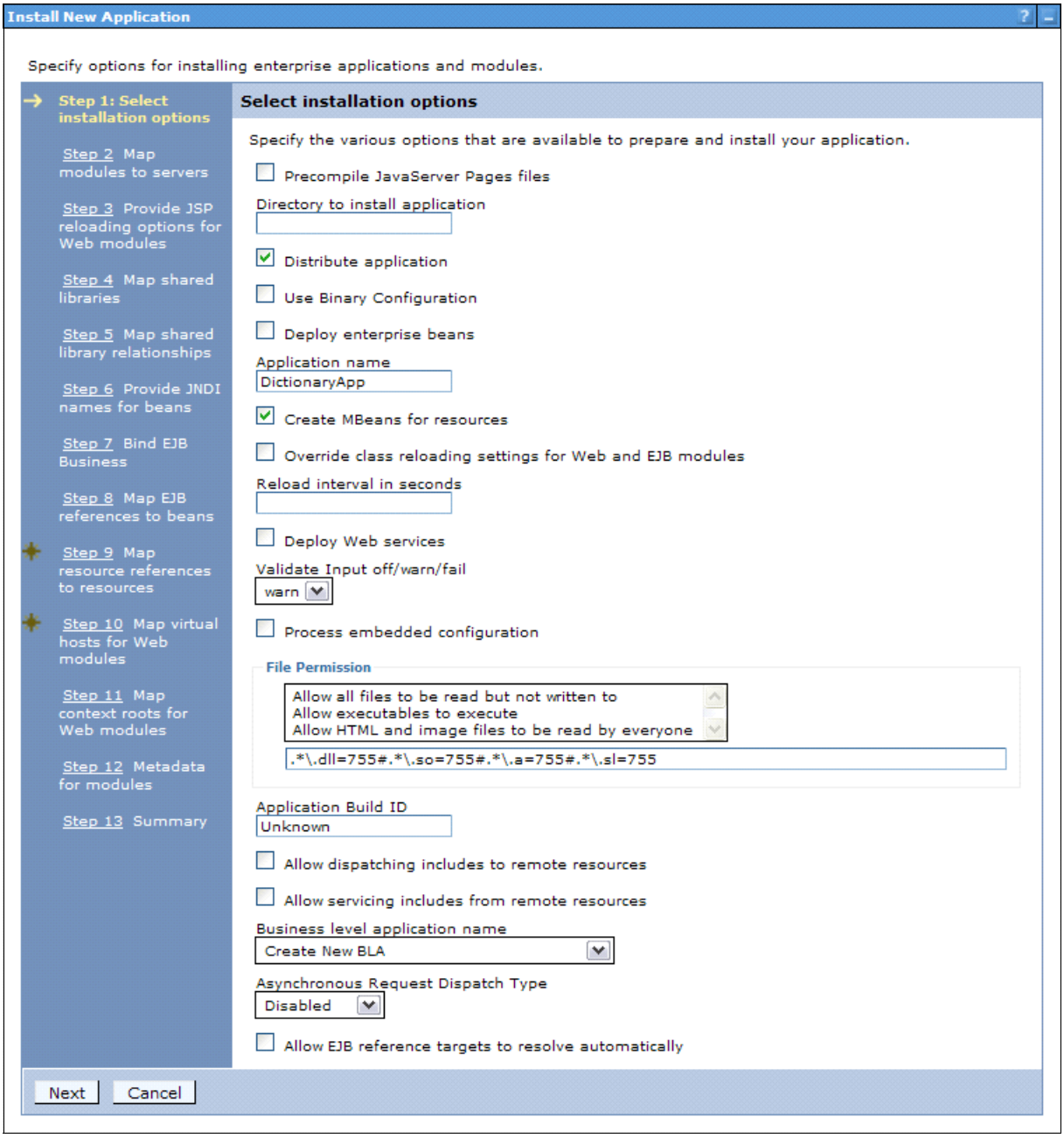


Figure 4-3 Administrative console: Install New Application

3. Select `server1` as the application server to run, as shown in Figure 4-4. Click **Next**.

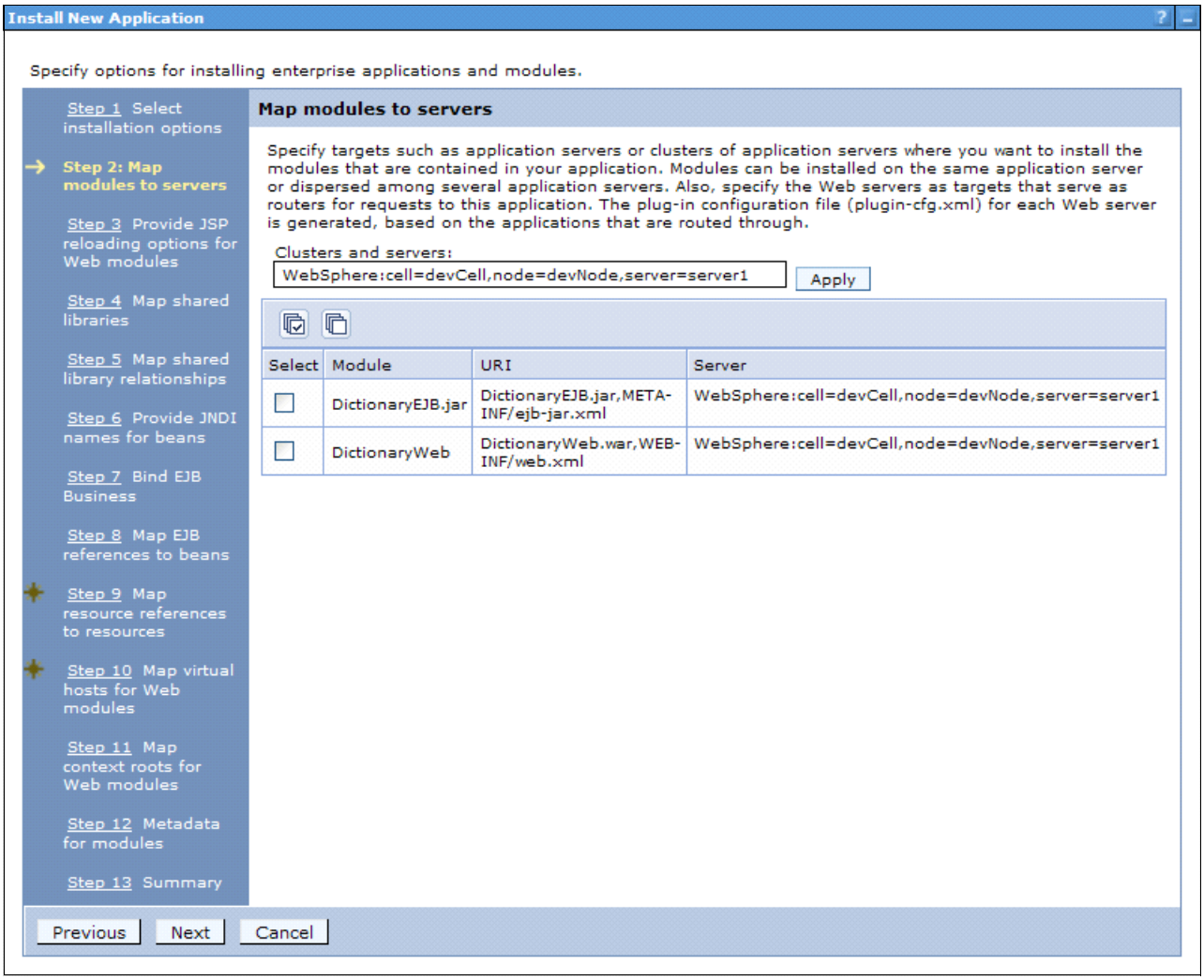


Figure 4-4 Administrative console: Map modules to servers

4. Specify a value for the JavaServer Pages (JSP) reloading interval as necessary for performance, as shown in Figure 4-5. For DictionaryApp V2, accept the defaults. Click **Next**.

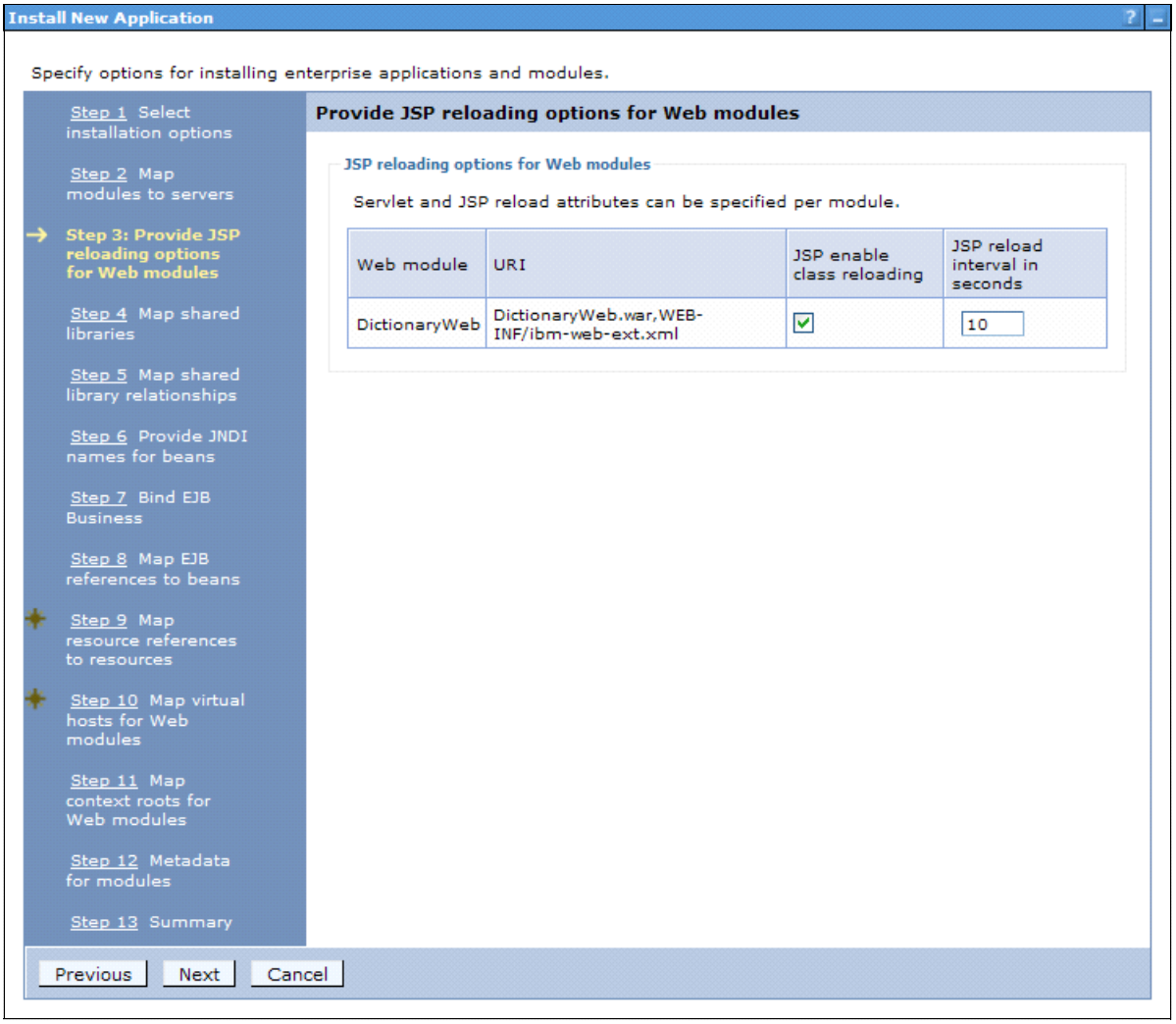


Figure 4-5 Administrative console: Provide JSP reloading options for Web modules

5. If your application requires any external libraries configured in the application server environment, you can reference them here. You can manage shared libraries by accessing the Shared Libraries page in the Environment section in the administration console. For DictionaryApp V2, accept the defaults, as shown in Figure 4-6. Click **Next**.

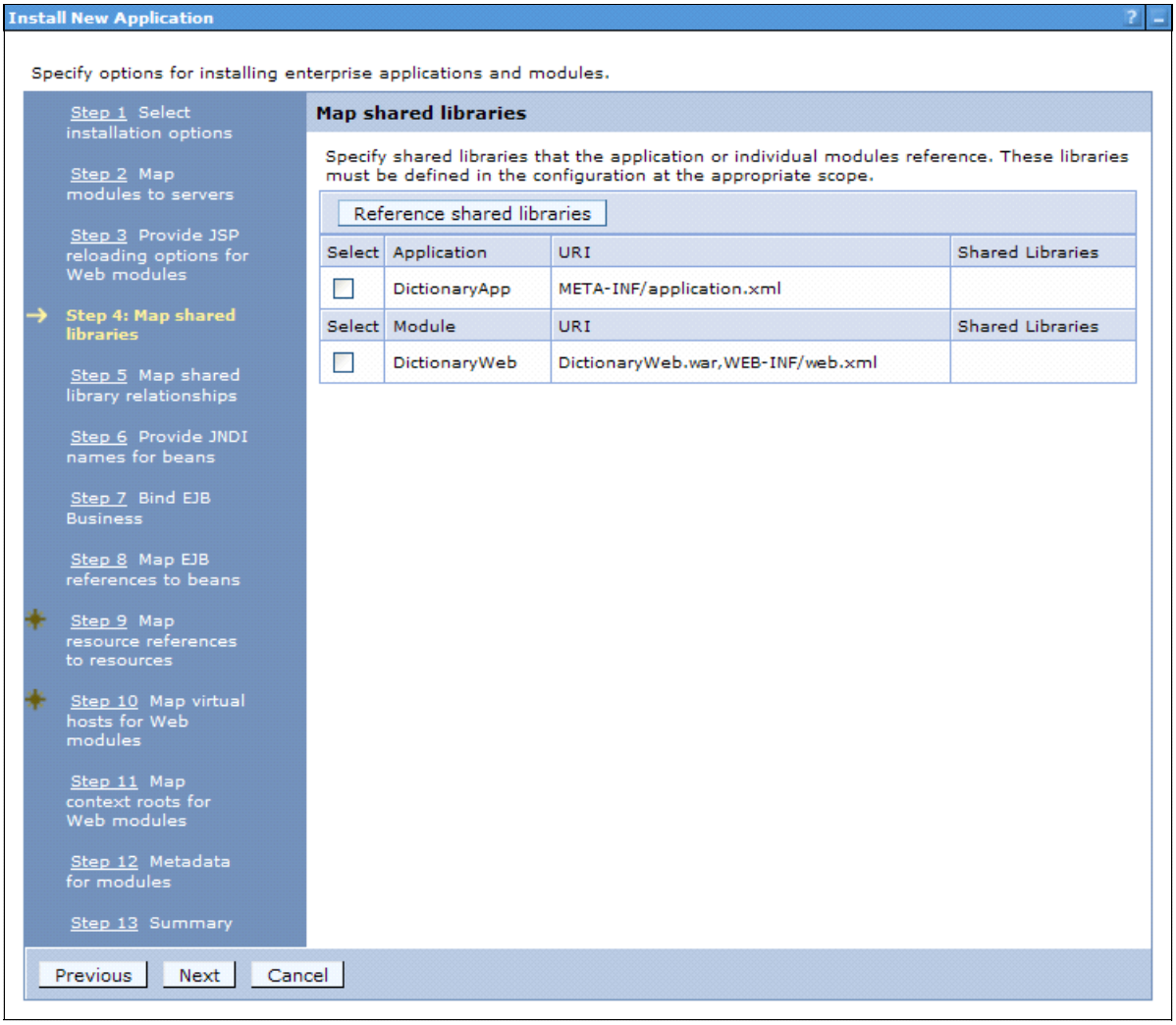


Figure 4-6 Administrative console: Map shared libraries

6. Specify asset or composition unit IDs as shared libraries that the application can reference. For DictionaryApp V2, accept the defaults, as shown in Figure 4-7. Click **Next**.

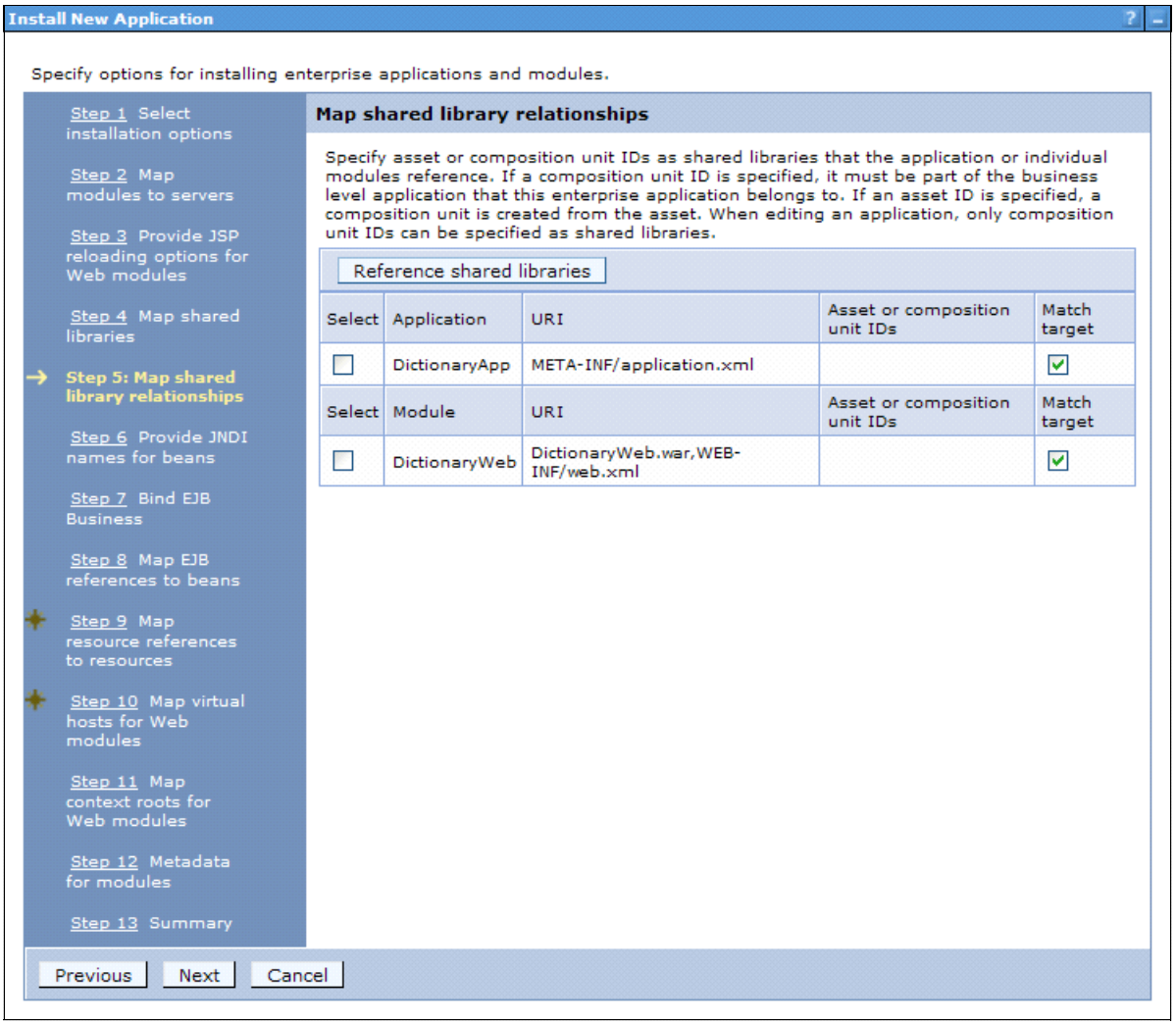


Figure 4-7 Administrative console: Map shared library relationships

7. Specify certain Java Naming and Directory Interface (JNDI) names for session EJBs in your application, as shown in Figure 4-8. These specifications override both deployment descriptors and annotations in the application. For DictionaryApp V2, accept the defaults. By accepting the defaults, you notify the application server to use the default JNDI names from the application. Click **Next**.

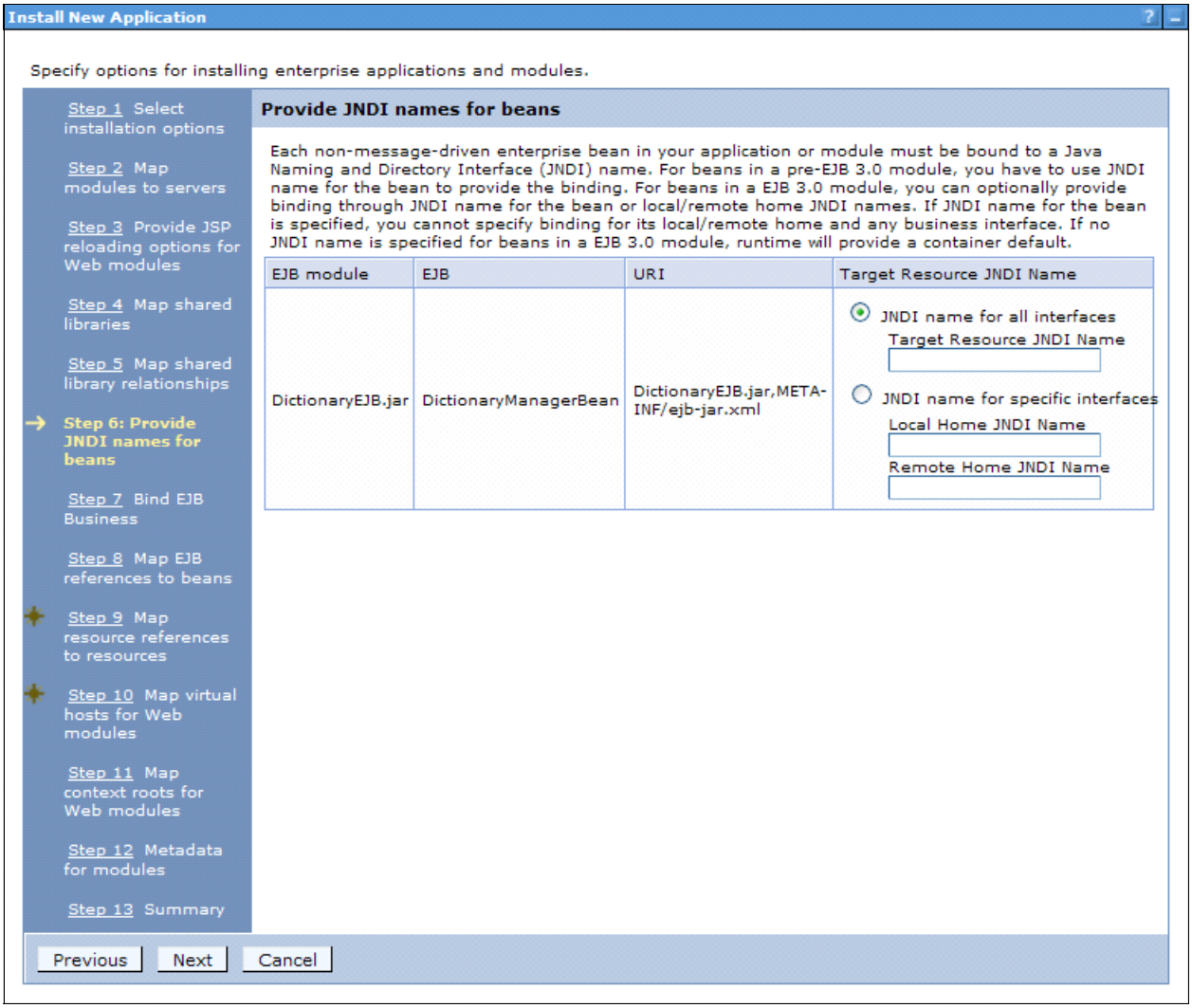


Figure 4-8 Administrative console: Provide JNDI names for beans

8. Specify JNDI names for EJB interfaces in your application, as shown in Figure 4-9. These specifications override both deployment descriptors and annotations. For DictionaryApp V2, accept the defaults to notify the application server to use default JNDI names from the application. Click **Next**.

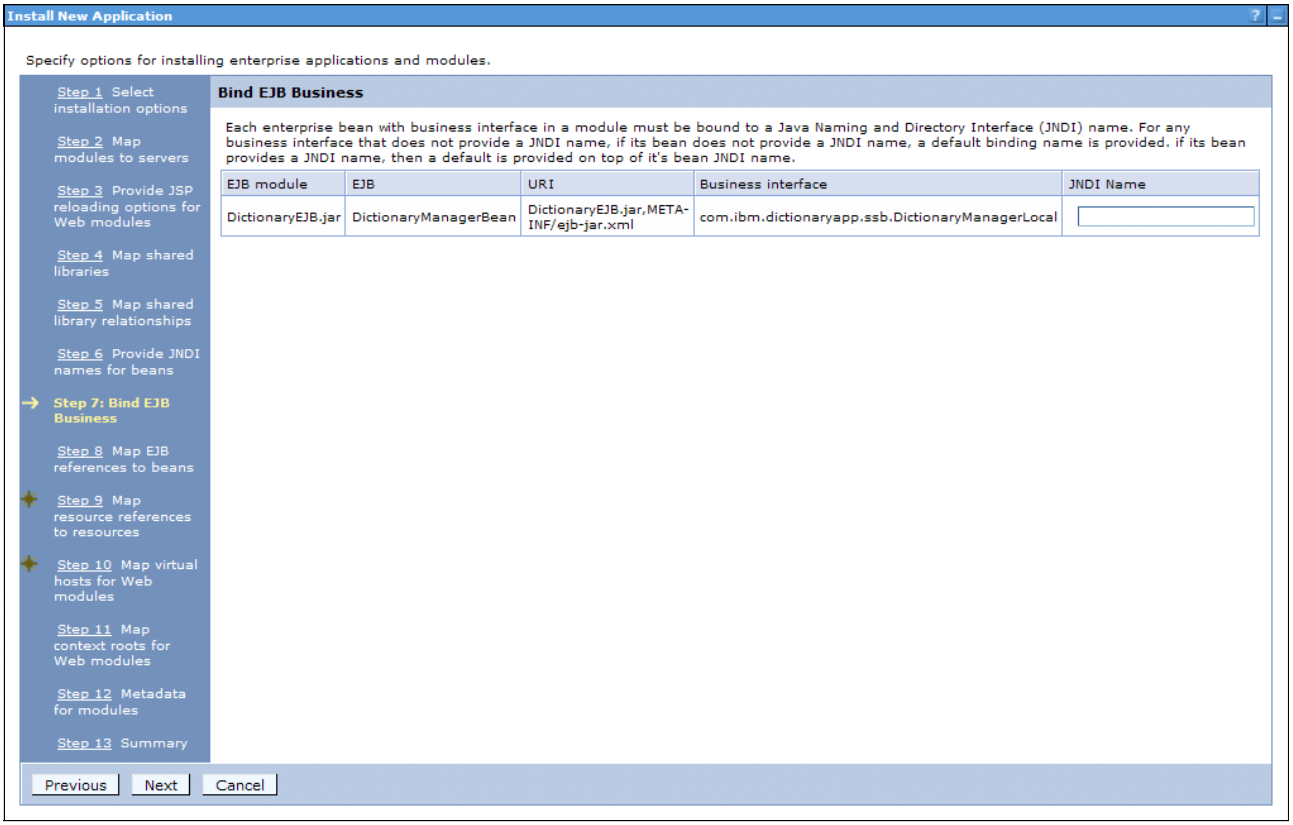


Figure 4-9 Administrative console: Bind EJB Business

9. Specify how EJBs are mapped to references to EJBs in your application, as shown in Figure 4-10. For DictionaryApp V2, accept the defaults. Click **Next**.

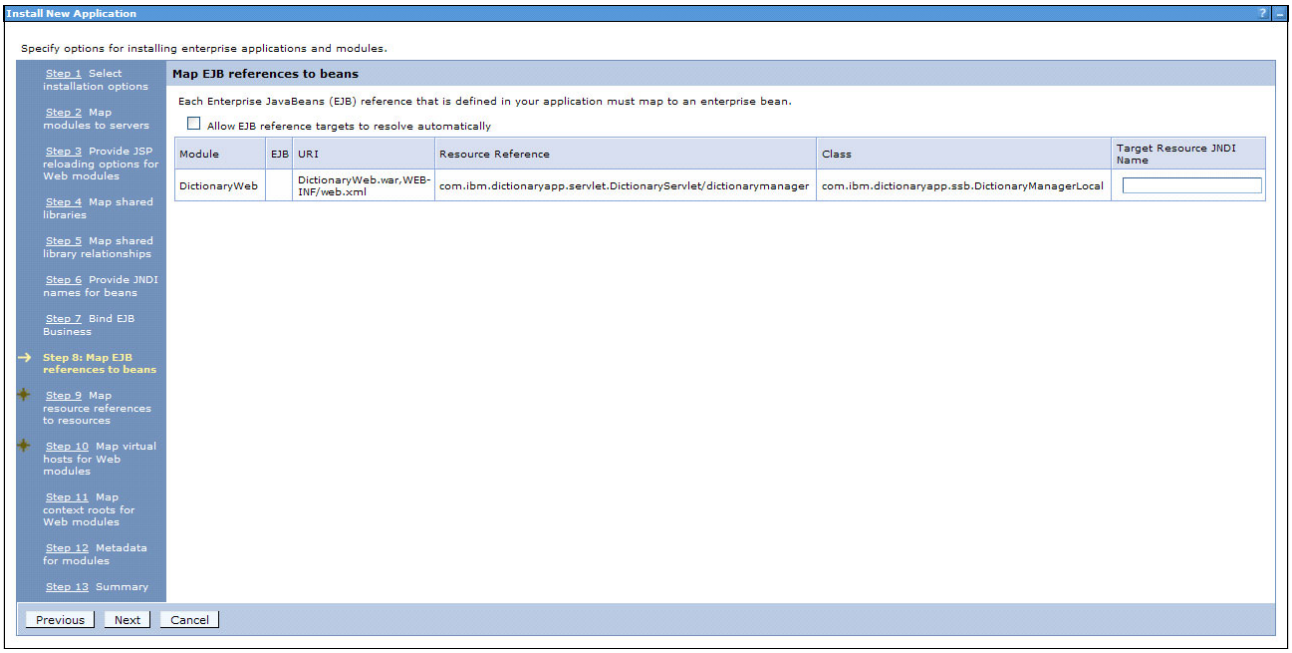


Figure 4-10 Administrative console: Map EJB references to beans

10. Map resource references to their resource JNDI names, as shown in Figure 4-11. For DictionaryApp V2, specify `jdbc/DictionaryDB` for the Target Resource JNDI Name. Click **Next**.

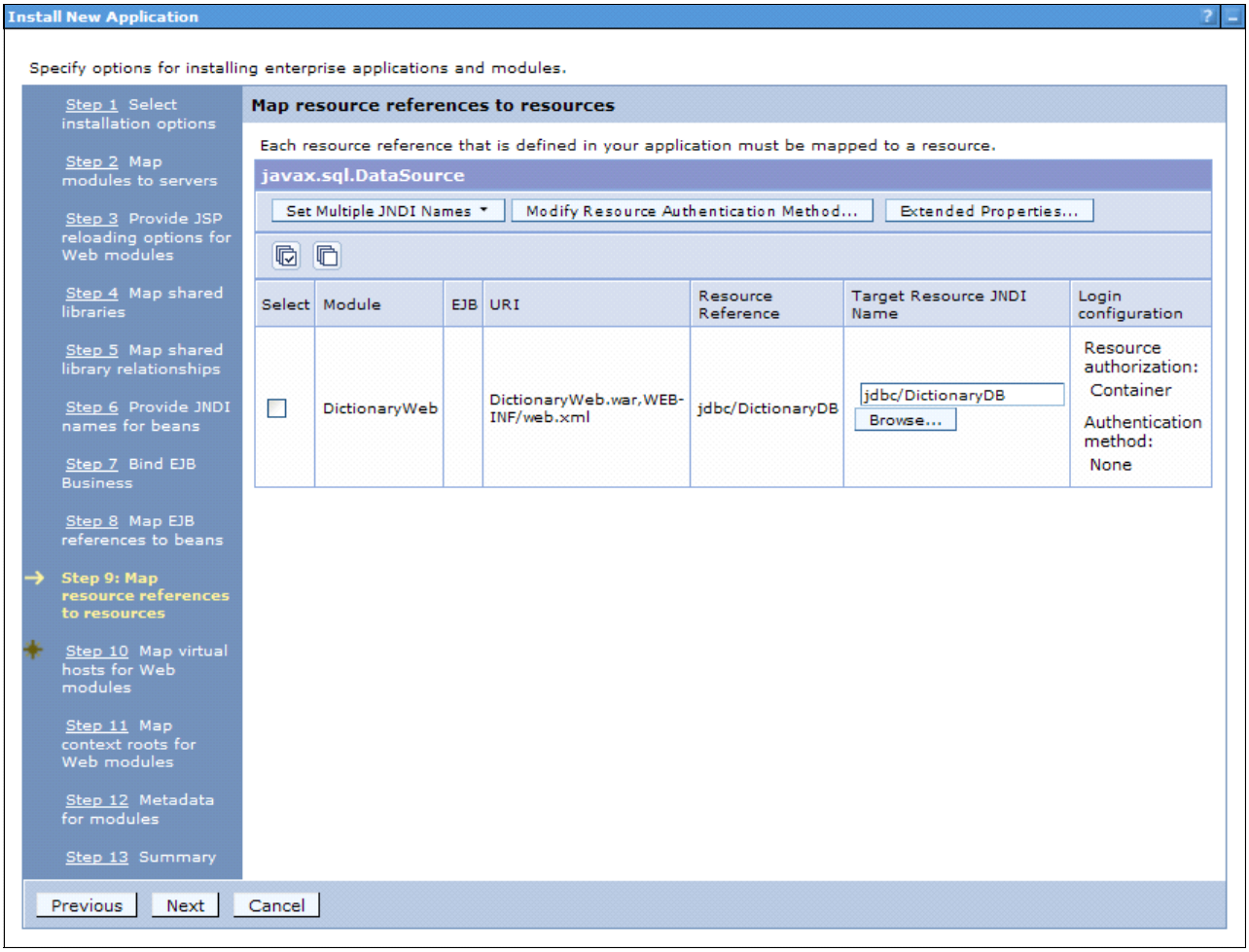


Figure 4-11 Administrative console: Map resource references to resources

11. Specify the virtual host on which the application runs, as shown in Figure 4-12. or DictionaryApp V2, accept the defaults. Click **Next**.

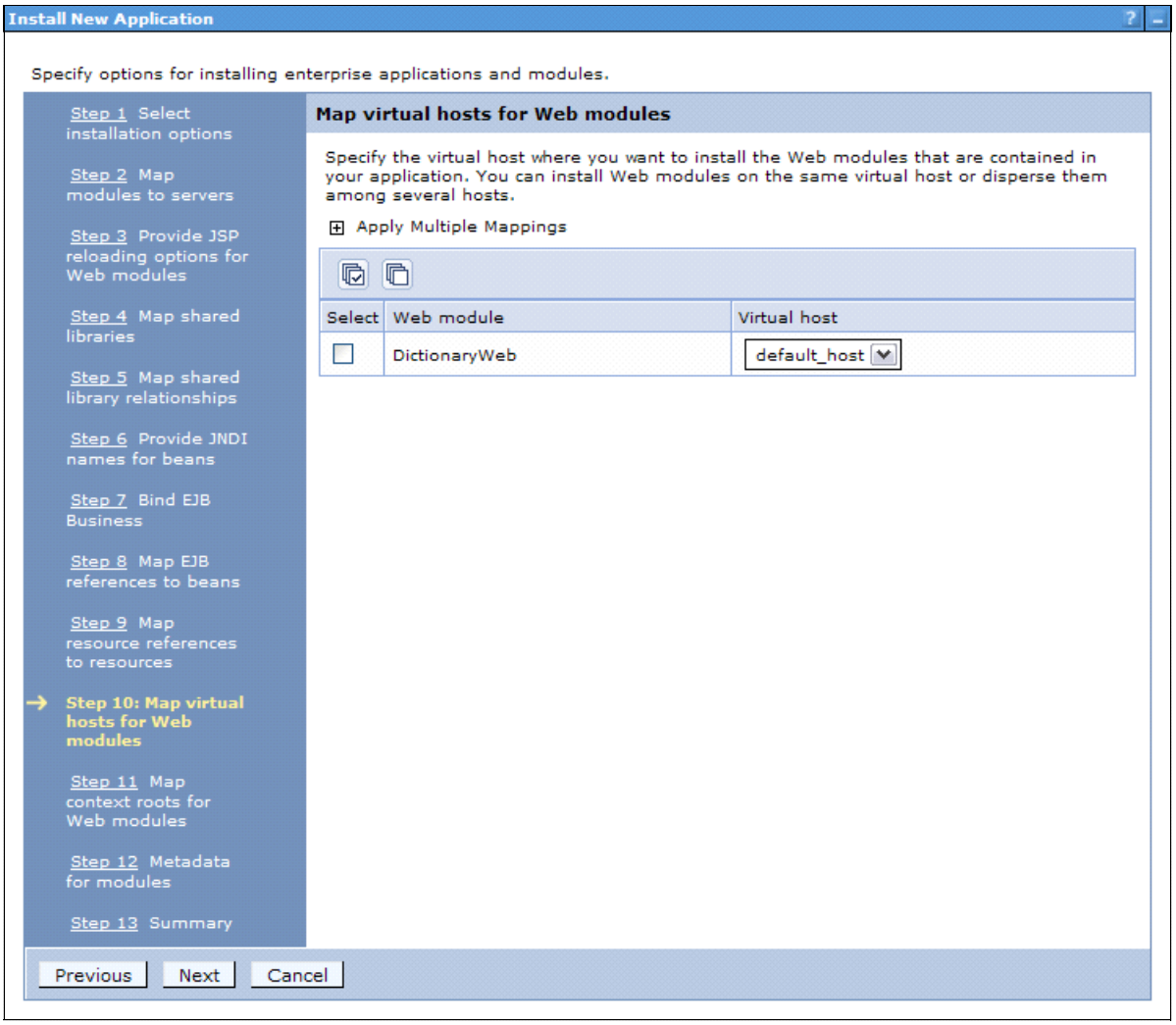


Figure 4-12 Administrative console: Map virtual hosts for Web modules

12. Specify the context root for your web module, as shown in Figure 4-13. For DictionaryApp V2, because this field must match what has already been specified in `web.xml`, accept the defaults. Click **Next**.

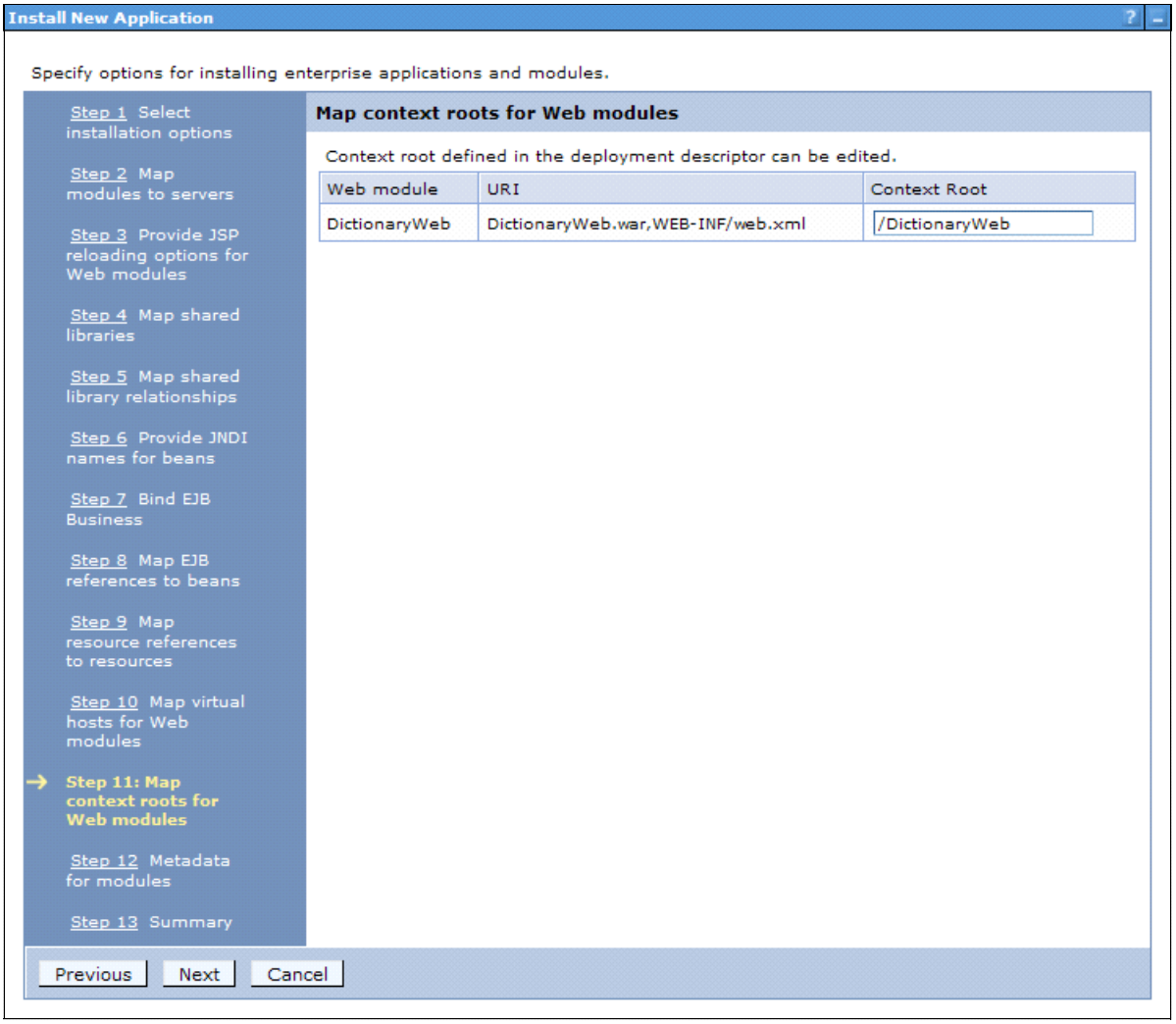


Figure 4-13 Administrative console: Map context roots for Web modules

13. Specify whether your application depends solely on XML deployment descriptors. For DictionaryApp V2, because the code that is provided uses EJB3 annotations heavily, leave every check box unchecked, as shown in Figure 4-14. Click **Next**.

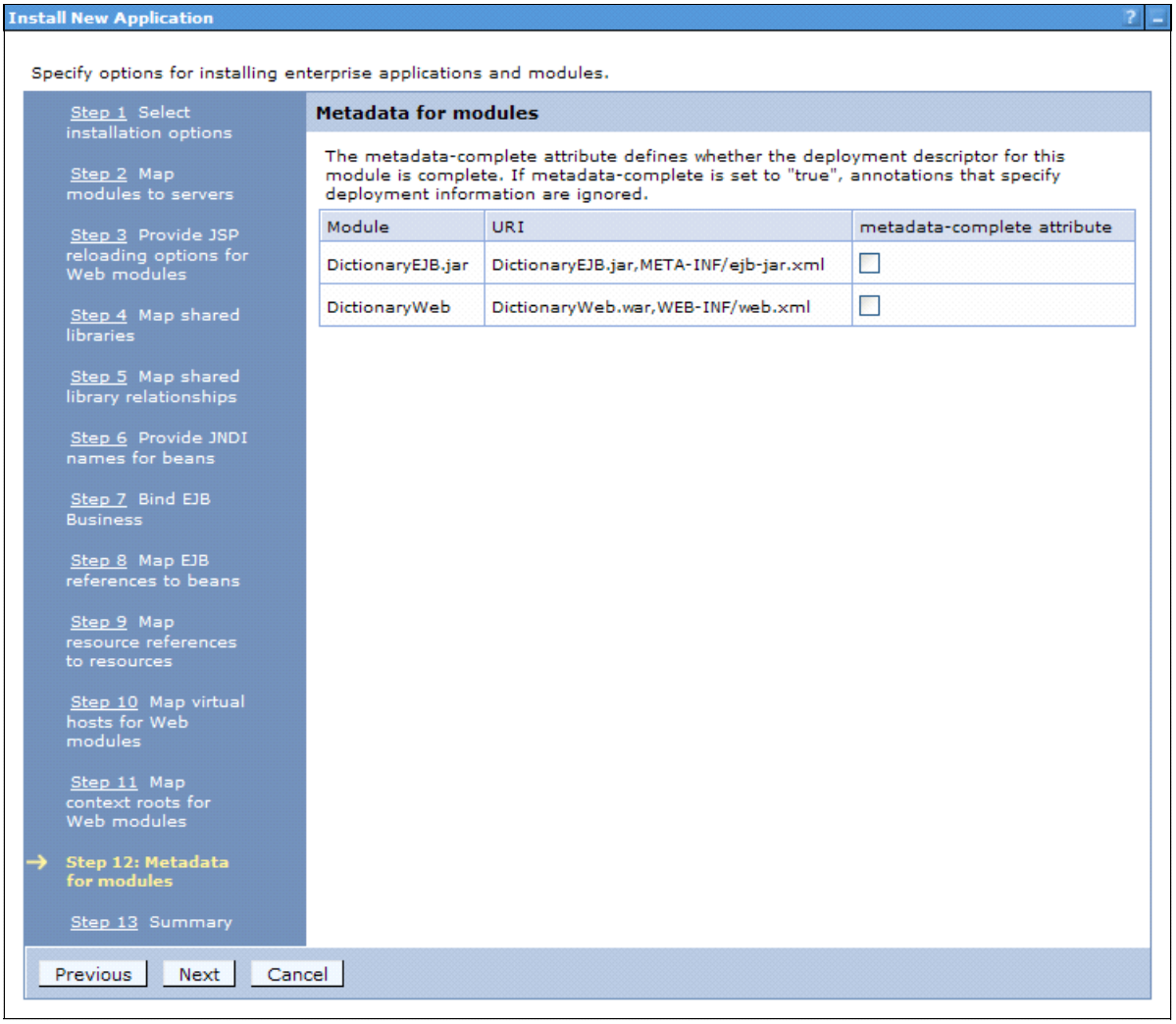


Figure 4-14 Administrative console: Metadata for modules

14. In the Summary, review the selections, and click **Finish** to install the application, as shown in Figure 4-15.

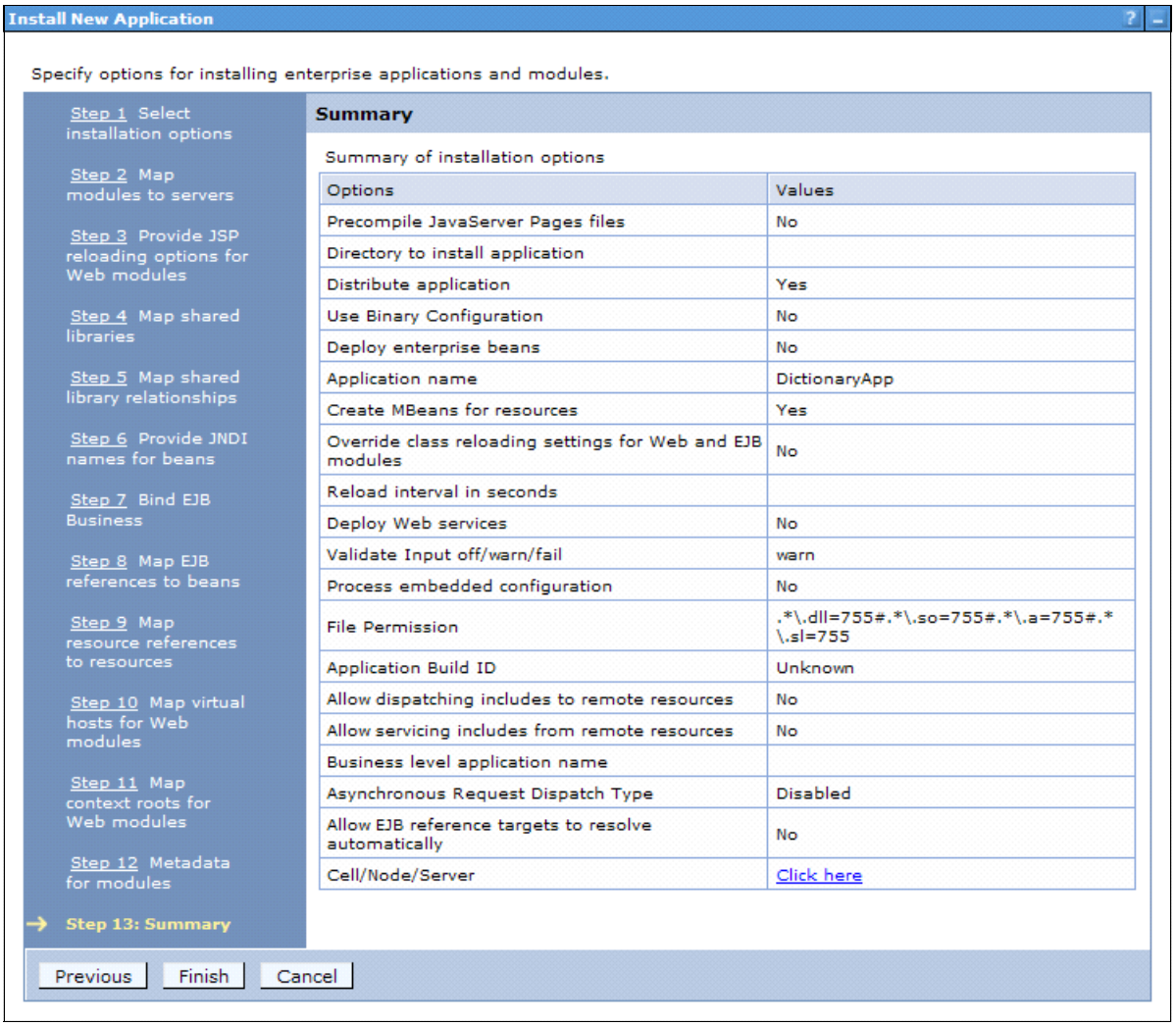


Figure 4-15 Administrative console: Install New Application Summary

4.3.2 wsadmin scripting

After first uninstalling the application, make sure that the script for redeploying DictionaryApp V2 looks similar to Example 4-8.

Example 4-8 wsadmin application redeployment script

```
AdminApp.install('C:/DictionaryApp/version
2/DictionaryApp.ear',['-appname',
'DictionaryApp', '-
MapResRefToEJB','[[DictionaryWeb ""
DictionaryWeb.war,WEB-INF/web.xml
jdbc/DictionaryDB javax.sql.DataSource
jdbc/DictionaryDB "" "" ""]]','-
MapWebModToVH','[[DictionaryWeb
DictionaryWeb.war,WEB-INF/web.xml
default_host]]']);
AdminConfig.save(); #We must save the
```

```
application to the configuration repository
before it can be started
AppManager =
AdminControl.queryNames('type=ApplicationManager,
process=server1,*');
AdminControl.invoke(AppManager,
'startApplication','DictionaryApp');
```

Accessing DictionaryApp V2

You can access DictionaryApp V2 at the following URL:

```
http://localhost:9080/DictionaryWeb/DictionaryServlet
```

DictionaryApp V2 looks similar to Figure 4-16.

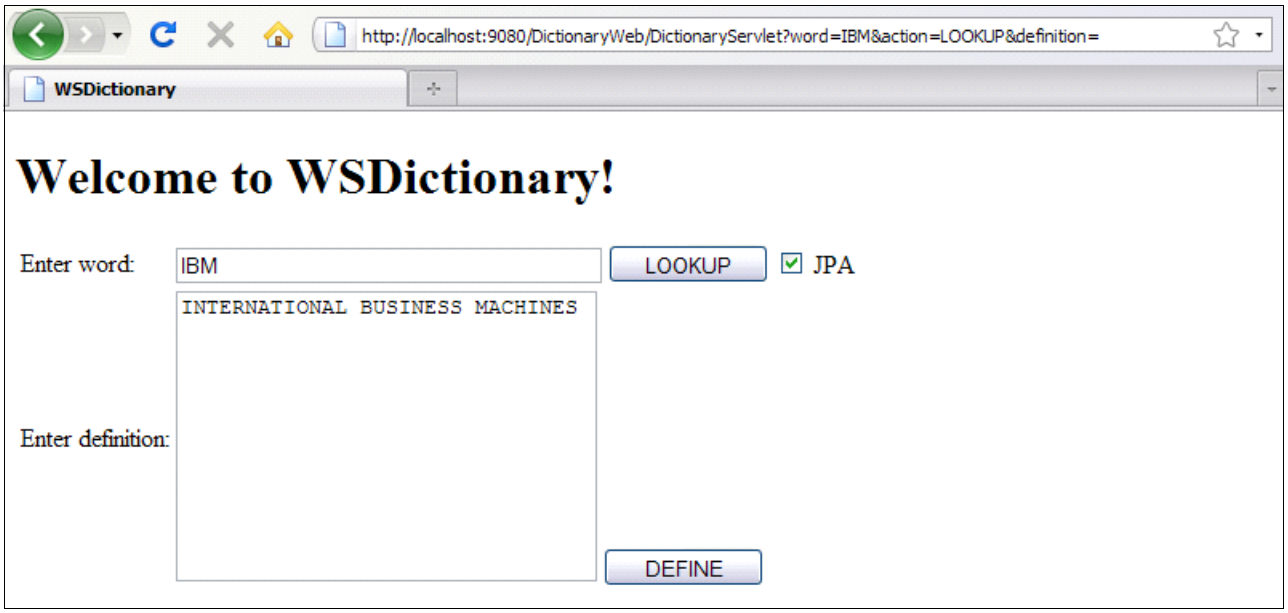


Figure 4-16 DictionaryApp V2

5

Incorporating JMS and MDBs into DictionaryApp V3

In this chapter, we add a messaging component to DictionaryApp V3 by incorporating the Java Message Service (JMS) and message-driven beans (MDBs) to allow for asynchronous messaging. DictionaryApp V3 supports logging differently by sending user actions to look up or update the dictionary as messages instead of logging it in the main flow of the application. This task demonstrates how to configure the messaging provider and the necessary resources to enable WebSphere messaging capabilities.

Sample material: See Appendix C, “Additional material” on page 235 for information about downloading the sample material that is used in this chapter.

© Copyright IBM Corp. 2010. All rights reserved.

5.1 DictionaryApp V3 overview

Figure 5-1 illustrates the overall architecture of DictionaryApp V3.

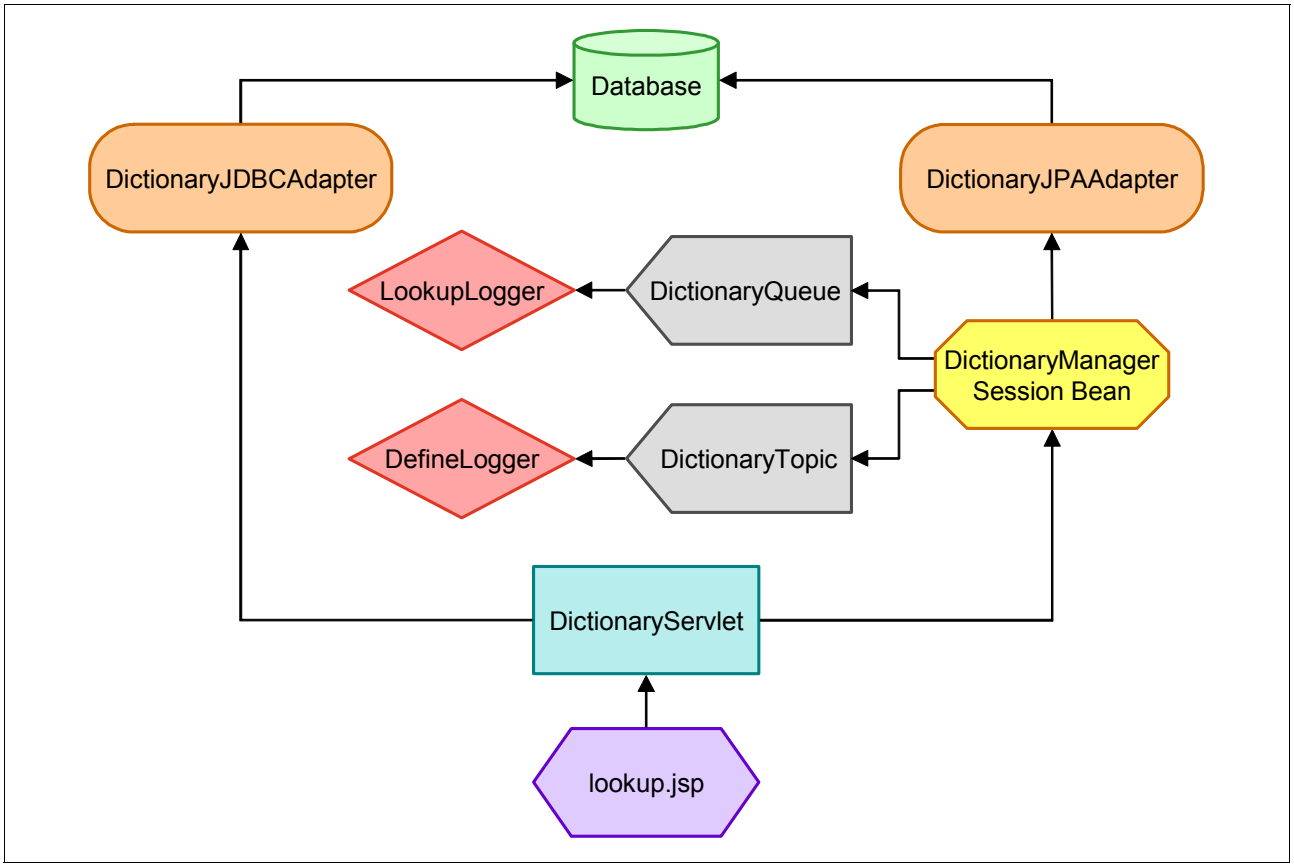


Figure 5-1 DictionaryApp3

DictionaryApp V3 includes two new classes to support messaging: LookupLoggerBean and DefineLoggerBean. LookupLoggerBean is an MDB that listens on its own destination, DictionaryJMSQueue, for messages from DictionaryServlet.java. DefineLoggerBean is an MDB that listens on its own destination, DictionaryJMSTopic, for messages from DictionaryServlet.java.

DictionaryManagerBean.java acts as the message sender on behalf of DictionaryServlet.java. At the end of every call to the service method in DictionaryServlet.java, it sends information regarding user activity to DictionaryManagerBean.java. Then, DictionaryManagerBean.java sends the user action information as messages to DictionaryJMSQueue and DictionaryJMSTopic for the MDBs to process.

5.2 DictionaryApp V3 source files

The following files are additions in DictionaryApp V3 to the existing source files in DictionaryApp V2:

- “DictionaryManagerLocal.java” on page 123
- “DictionaryManagerBean.java” on page 123

- “LookupLoggerBean.java” on page 125
- “DefineLoggerBean.java” on page 125
- “DictionaryServlet.java” on page 126

5.2.1 DictionaryManagerLocal.java

This file is updated to include the handleSessionData method, which `DictionaryManagerBean.java` implements. DictionaryServlet.java uses the handleSessionData method for logging, replacing direct calls to Logger.java in prior versions of the application.

Example 5-1 shows the text of `DictionaryManagerLocal.java`.

Example 5-1 DictionaryManagerLocal.java

```
...

public Entry lookup(String word) throws Exception;

public void handleSessionData(String action, String description)
      throws Exception;

...
```

5.2.2 DictionaryManagerBean.java

This file is updated to send user action information as messages to their intended destinations. With life cycle methods init() and destroy(), this file is used to construct and close message connections.

Example 5-2 shows the text of `DictionaryManagerBean.java`.

Example 5-2 DictionaryManagerBean.java

```
...

@Stateless
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public class DictionaryManagerBean implements DictionaryManagerLocal {

   @Resource(name = "jms/DictionaryQCF")
   private QueueConnectionFactory queueconnectionfactory;

   @Resource(name = "jms/DictionaryTCF")
   private TopicConnectionFactory topicconnectionfactory;

   @Resource(name = "jms/DictionaryQ")
   private Queue queuedestination;

   @Resource(name = "jms/DictionaryT")
   private Topic topicdestination;

   private QueueConnection queueconnection;
   private TopicConnection topicconnection;
   private DictionaryJPAAdapter jpaadapter;
```

```
...

public Entry lookup(String word) throws Exception {
      return jpaadapter.lookup(word);
   }

@TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
   public void handleSessionData(String action, String description)
         throws Exception {
      Session session = null;
      MessageProducer producer = null;
      try {
         if (action.equals("LOOKUP")) {
            session = queueconnection.createSession(true,
                  Session.AUTO_ACKNOWLEDGE);
            producer = session.createProducer(queuedestination);
         } else if (action.equals("DEFINE")) {
            session = topicconnection.createSession(true,
                  Session.AUTO_ACKNOWLEDGE);
            producer = session.createProducer(topicdestination);
         }
         Message message = session.createMessage();
         message.setStringProperty("description", description);
         producer.send(message);
      } finally {
         if (producer != null) {
            producer.close();
         }
         if (session != null) {
            session.close();
         }
      }
   }

@PostConstruct
public void init() {
   try {
      this.queueconnection = queueconnectionfactory
            .createQueueConnection();
      this.topicconnection = topicconnectionfactory
            .createTopicConnection();
   } catch (Exception e){
      DataManager.logError(e);
   }
}

@PreDestroy
public void destroy() {
   try {
      if (this.queueconnection != null)
         this.queueconnection.close();
      if (this.topicconnection != null)
         this.topicconnection.close();
   } catch (Exception e) {
```

```
            DataManager.logError(e);
         }
      }
}
```

5.2.3 LookupLoggerBean.java

This MDB receives user data information concerning each LOOKUP operation. This MDB listens on the JMS queue called DictionaryJMSQueue. Upon receiving user action information, it invokes the Logger class, which then logs the information in the `SystemOut log`.

Example 5-3 shows the text of `LookupLoggerBean.java`.

Example 5-3 LookupLoggerBean.java

```
package com.ibm.dictionaryapp.mdb;

import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;

import com.ibm.dictionaryapp.datautil.Logger;

@MessageDriven(activationConfig = {
      @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
"javax.jms.Queue"),
      @ActivationConfigProperty(propertyName = "destination", propertyValue =
"jms/DictionaryQ") })
public class LookupLoggerBean implements MessageListener {
   public void onMessage(Message message) {
      try {
         String description = message.getStringProperty("description");
         Logger.logAction("LOOKUP", description);
      } catch (Exception e) {
         Logger.logError(e);
      }
   }
}
```

5.2.4 DefineLoggerBean.java

This MDB receives user action information concerning each DEFINE operation. This MDB listens on the JMS topic called DictionaryJMSTopic. Upon receiving user action information, it invokes the Logger class, which then logs the information in the `SystemOut log`.

Example 5-4 shows the text of `DefineLoggerBean.java`.

Example 5-4 DefineLoggerBean.java

```
package com.ibm.dictionaryapp.mdb;

import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
```

```
import javax.jms.MessageListener;

import com.ibm.dictionaryapp.datautil.Logger;

@MessageDriven(activationConfig = {
      @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
"javax.jms.Topic"),
      @ActivationConfigProperty(propertyName = "destination", propertyValue =
"jms/DictionaryT") })
public class DefineLoggerBean implements MessageListener {
   public void onMessage(Message message) {
      try {
         String description = message.getStringProperty("description");
         Logger.logAction("DEFINE", description);
      } catch (Exception e) {
         Logger.logError(e);
      }
   }
}
```

5.2.5 DictionaryServlet.java

DictionaryServlet.java is the updated file to invoke asynchronous logging. Example 5-5 shows the text of `DictionaryServlet.java`.

Example 5-5 DictionaryServlet.java

```
...
      else {
         request.setAttribute("word", entry.getWord());
         request.setAttribute("definition", entry
                  .getDefinition());
         description = entry.toString();
         }
      }
      dictionarymanager.handleSessionData(action, description);
      ...
         Entry entry = new Entry(word, definition);
         databaseadapter.define(entry);
         message = word + " has been defined.";
         request.setAttribute("word", word);
         request.setAttribute("definition", definition);
         description = entry.toString();
      }
      dictionarymanager.handleSessionData(action, description);
      }
   } catch (Exception e) {
      Logger.logError(e);
      message = "ERROR : " + e.getClass().getName();
   }
```

5.3 WebSphere Service Integration Bus

The WebSphere Service Integration Bus (SIBus) is the default built-in messaging provider for WebSphere. The messaging provider serves as the back-end message-oriented middleware (MOM) for applications that require messaging services. This separation of application logic and messaging functionality allows for interchangeability between the application layer and the messaging layer. Therefore, an application that uses the WebSphere SIBus can easily switch to MQ, another message provider in WebSphere. Figure 5-2 illustrates how the SIBus interacts with other components in the application server.

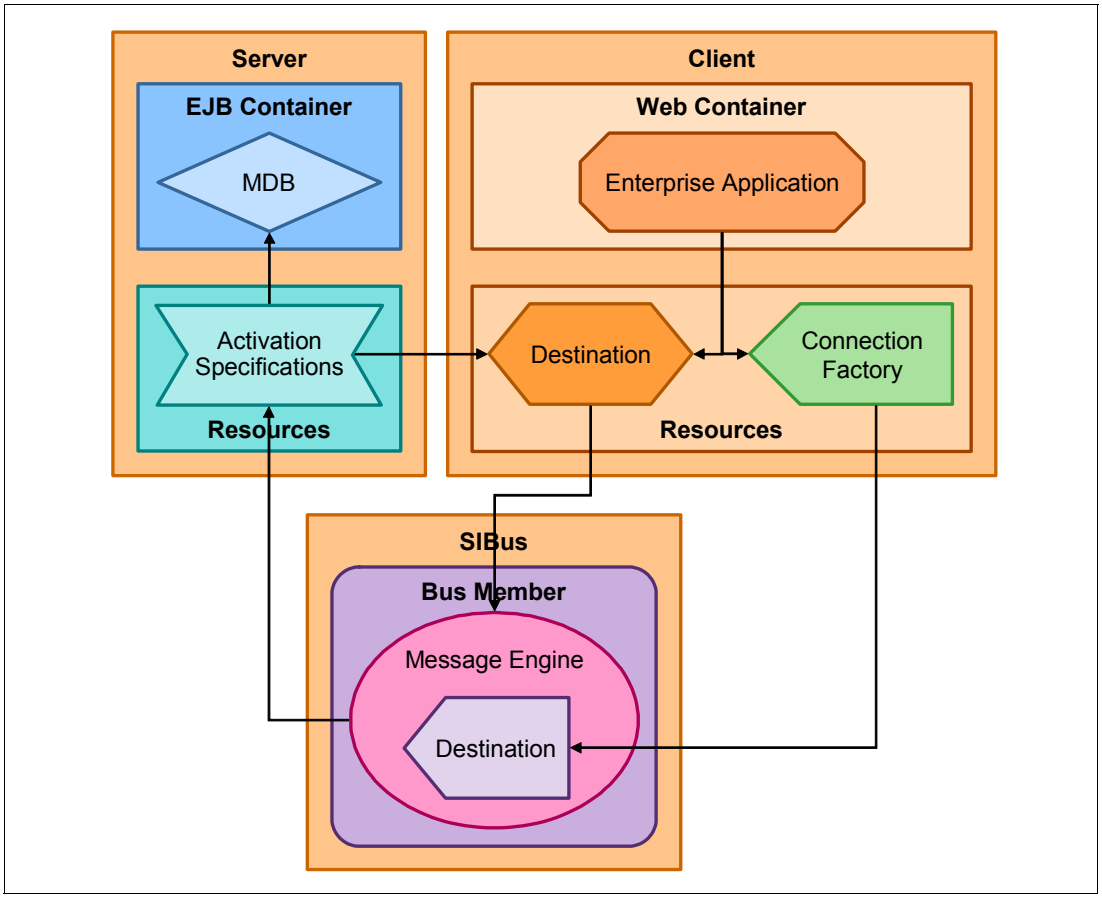


Figure 5-2 WebSphere SIBus

The SIBus handles many important tasks, including sending and receiving messages, persisting messages, and setting up mediations between sender and receivers. For our single-server environment, we must configure our application server as a member in the SIBus. After a server becomes a member of the SIBus, the SIBus establishes a message engine for the server. Within the messaging engine, you can configure messaging destinations for the server member.

In order for applications within the web container to access the messaging services of the SIBus, you must configure resources within the application server both on the client side and on the server side. On the client side, you must configure a connection factory resource and destination resource to allow for access to the messaging engine within the SIBus member and the destinations hosted within it. This access allows the client to send messages to the SIBus. On the server side, you must configure a set of activation specifications for client

MDBs to receive messages from the messaging engine, which allows the server to retrieve the messages from the SIBus.

5.4 Setting up the service integration bus for DictionaryApp V3

For DictionaryApp V3, a SIBus, DictionarySIBus, is configured as its messaging provider. Within the SIBus, the server hosting DictionaryApp V3 is included as a bus member for its messaging capabilities. You configure a queue destination, DictionarySIBQueue, and a topic space destination, DictionarySIBTopic, to process logging the actions to look up or define the dictionary.

5.4.1 Administrative console

Use the following procedure to configure DictionarySIBus using the administrative console:

1. In the administrative console, click **Service Integration** → **Buses** in the navigation tree.
2. In the Buses page, click **New** to create a new SIBus, as shown in Figure 5-3.

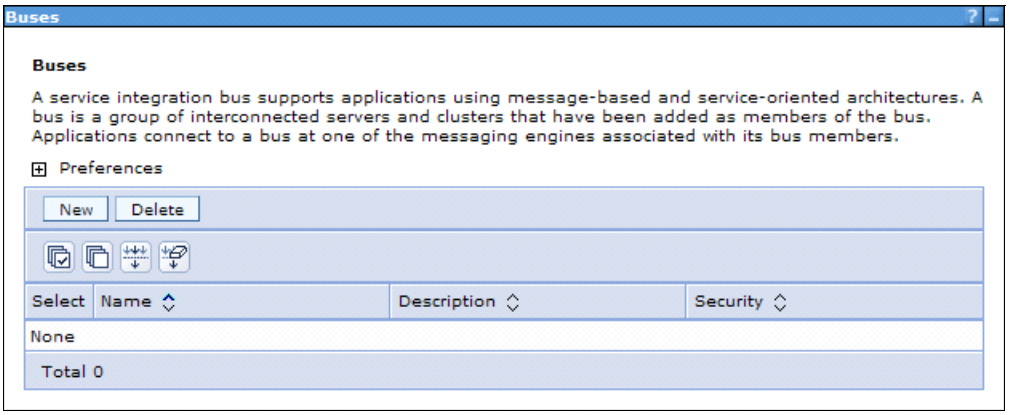


Figure 5-3 Administrative console: Buses

3. Select a name for the SIBus, and check Bus security, if needed. For DictionaryApp V3, specify `DictionarySIBus` as the name (Figure 5-4) and ensure that you do not select Bus security; otherwise, your application will not be able to start later. Click **Next**.

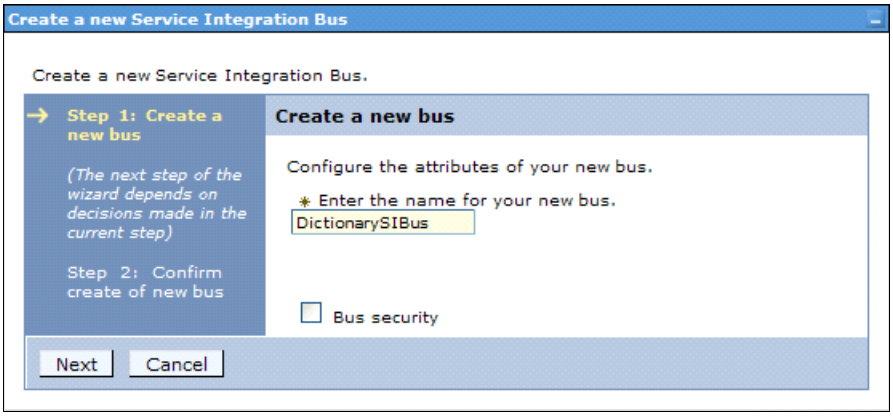


Figure 5-4 Administrative console: Create a new Service Integration Bus

4. In the Summary, review your choices, and click **Finish** to create the SIBus, as shown in Figure 5-5.

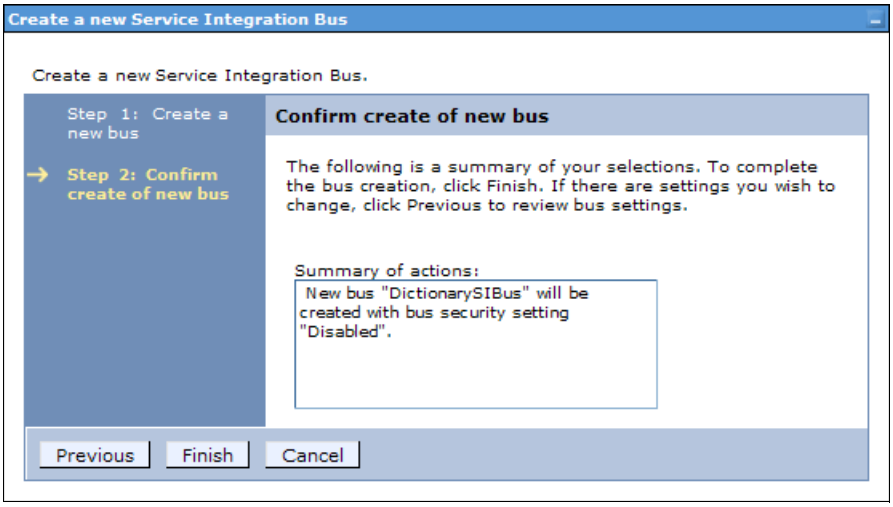


Figure 5-5 Administrative console: Create a new Service Integration Bus summary

After creating DictionarySIBus, you can get command assistance for creating DictionaryDatasource by clicking **View administrative scripting command for last action** in the help box. The output looks similar to Figure 5-6. Remember to return to save your changes after viewing the scripting commands.

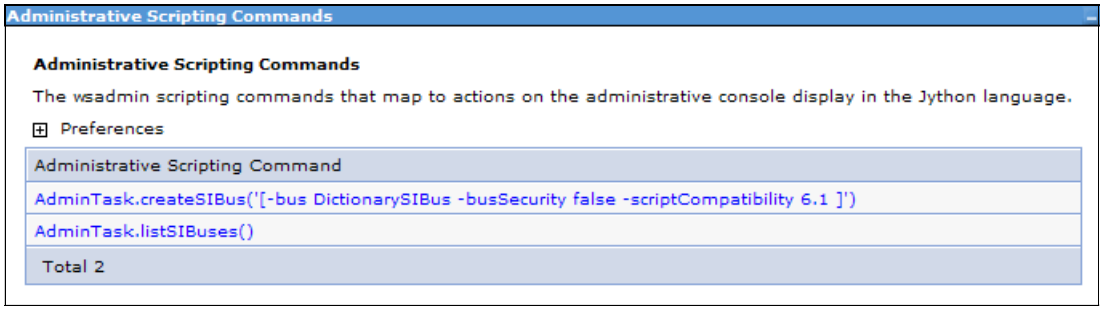


Figure 5-6 Administrative console: Administrative Scripting Commands

5. After creating DictionarySIBus, go back to the Buses page. You can see DictionarySIBus in the list of SIBuses, as shown in Figure 5-7.

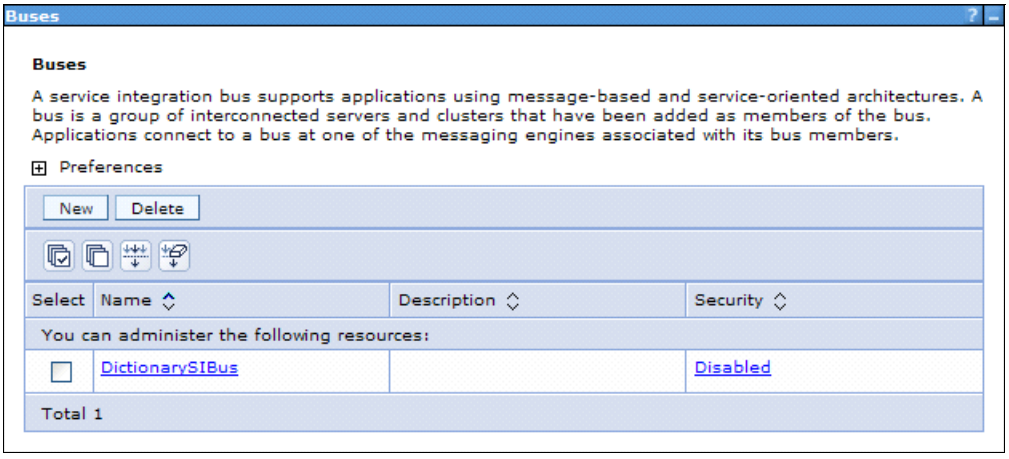


Figure 5-7 Administrative console: Buses

6. Select **DictionarySIBus**. This selection takes you to **Buses** → **DictionarySIBus**, as shown in Figure 5-8.

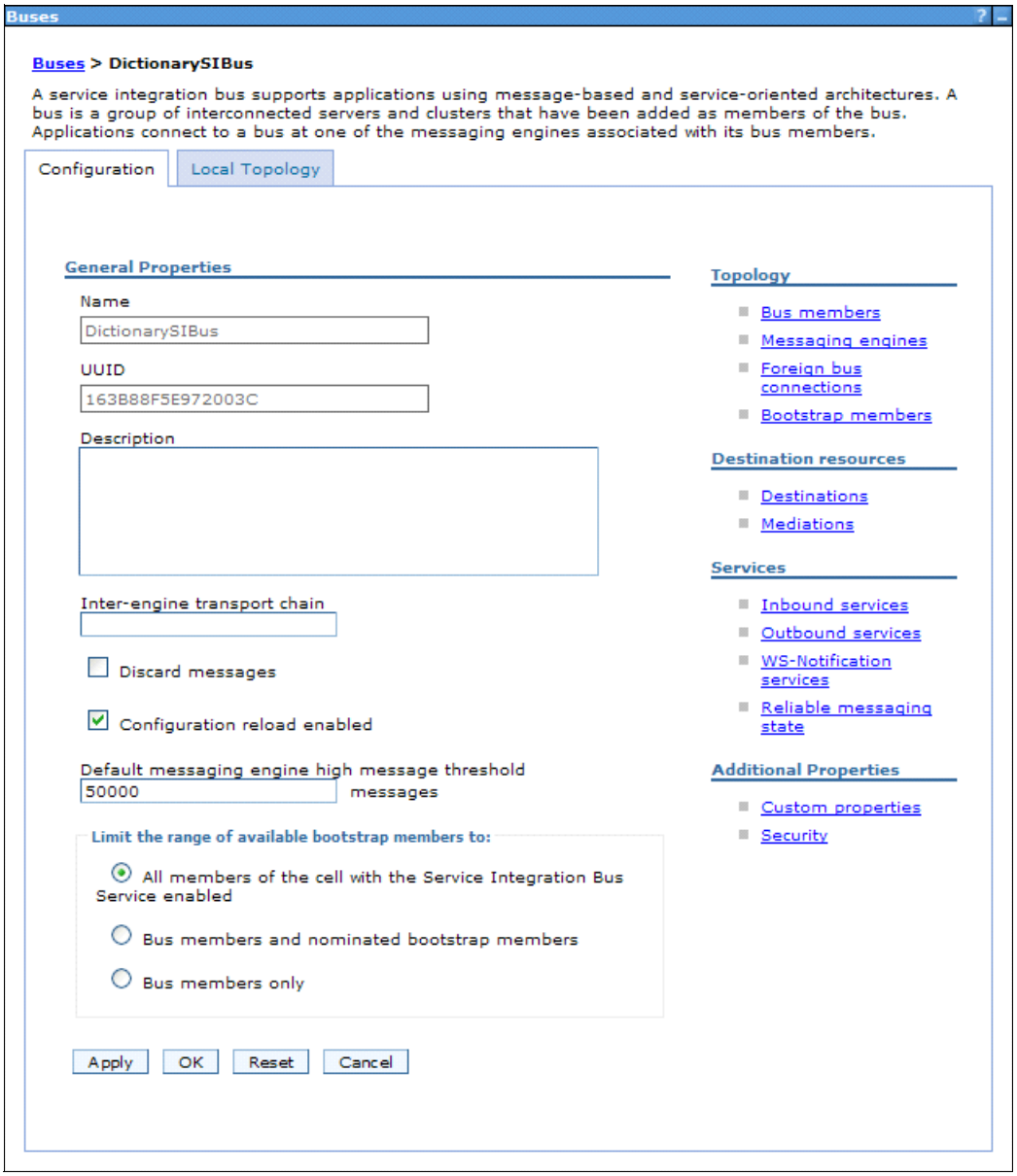


Figure 5-8 Administrative console: DictionarySIBus

7. Under the Topology section, select **Bus members**. This option takes you to **Buses** → **DictionarySIBus** → **Bus members**, which contains a list of bus members configured for the DictionarySIBus. Because you have just created DictionarySIBus, the list is empty, as shown in Figure 5-9. Click **Add**.

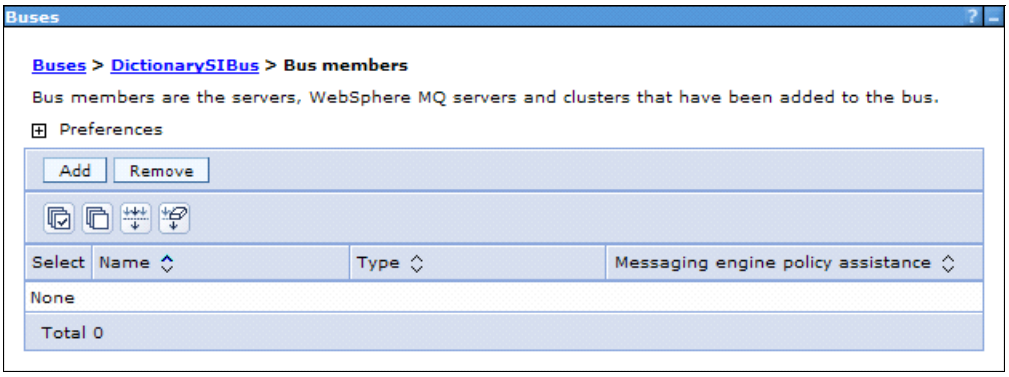


Figure 5-9 Administrative console: Bus members

8. Select the server that will become the member of the SIBus. For DictionaryApp V3, select **Server**, and choose your server (select **server1** if you use the default profile that was configured during installation) as the member for the SIBus, as shown in Figure 5-10. Click **Next**.

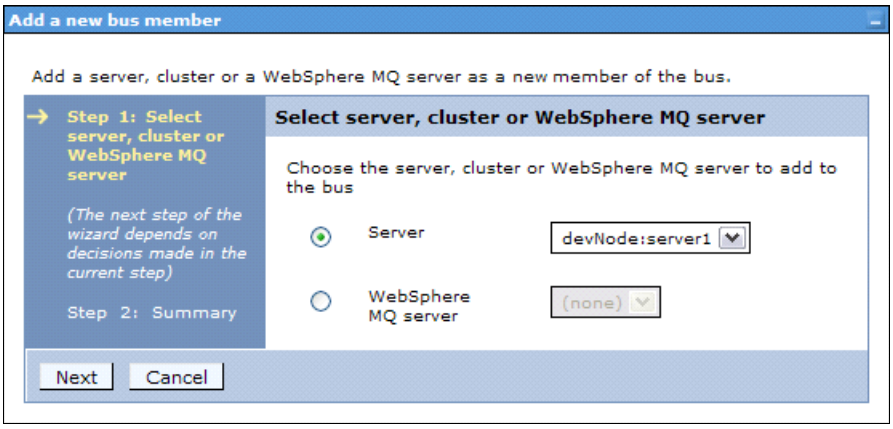


Figure 5-10 Administrative console: Add a new bus member

9. Select the message storage option for your SIBus. This option specifies how the messaging engine that is associated with your bus member persists the messages that it receives in order to recover lost data in the case of a server failure. For DictionaryApp V3, select **File store**, as shown in Figure 5-11. Click **Next.**

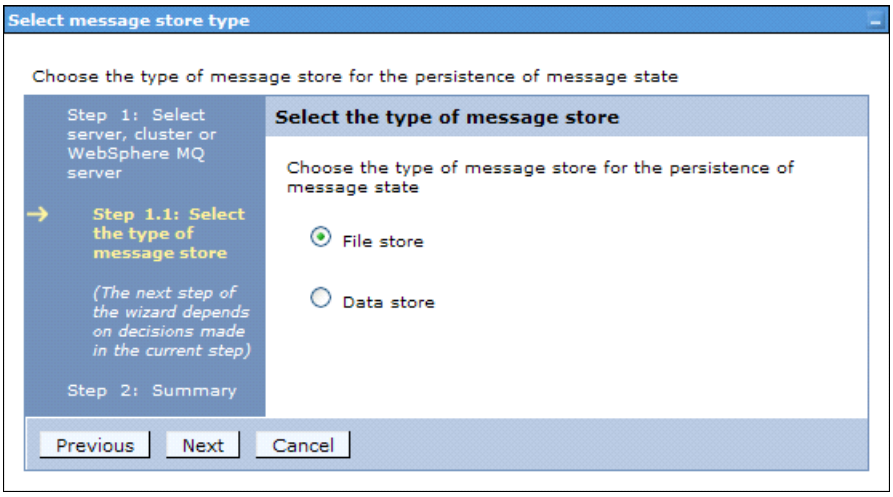


Figure 5-11 Administrative console: Add a new bus member: Select the type of message store

10. Specify the file storage options for your SIBus member. For DictionaryApp V3, accept the defaults, as shown in Figure 5-12. Click **Next**.

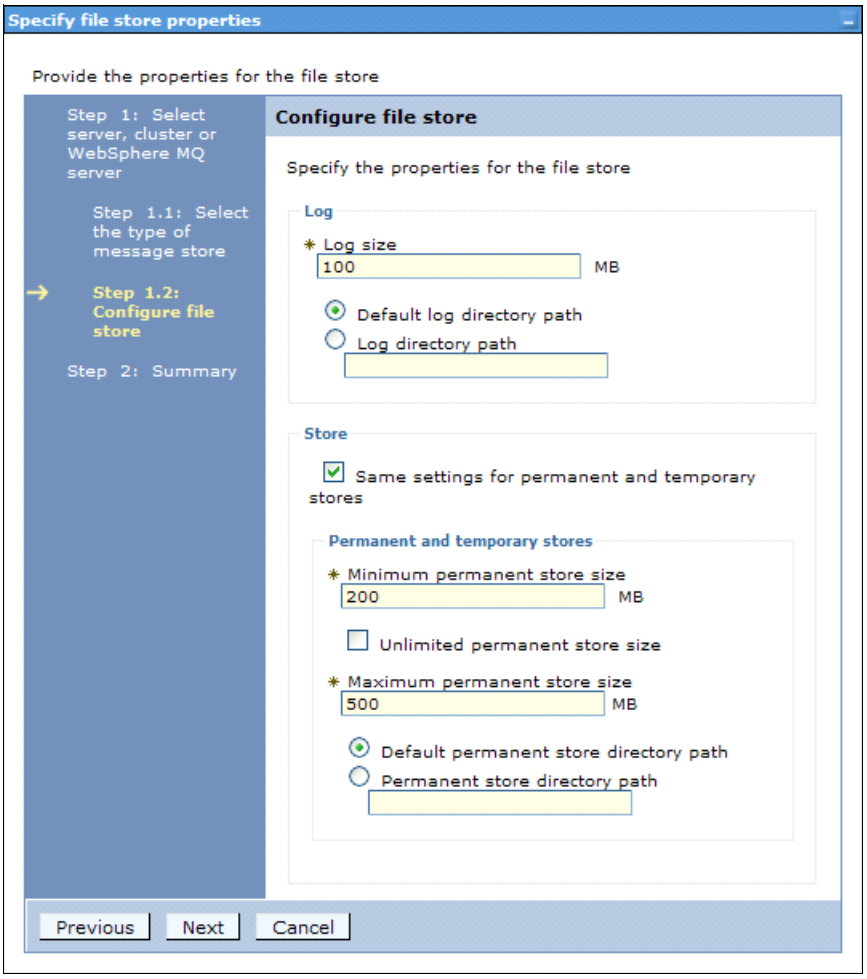


Figure 5-12 Administrative console: Add a new bus member: Configure file store

11. Specify storage parameters to improve the performance of messaging. For DictionaryApp V3, accept the defaults, as shown in Figure 5-13. Click **Next**.

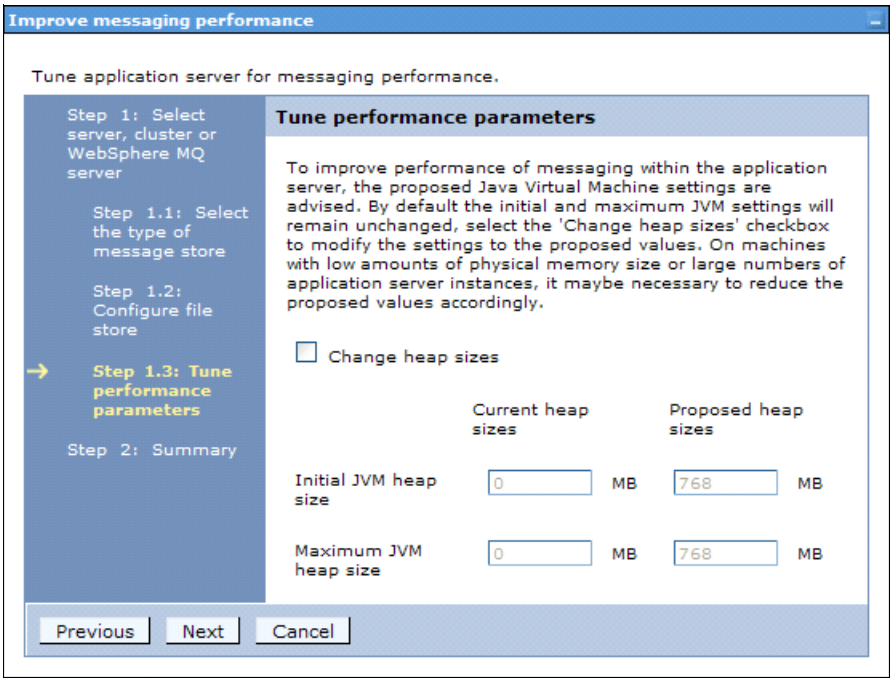


Figure 5-13 Administrative console: Add a new bus member: Tune performance parameters

12. In the Summary section, review your choices, and click **Finish** to add your server as a bus member, as shown in Figure 5-14.

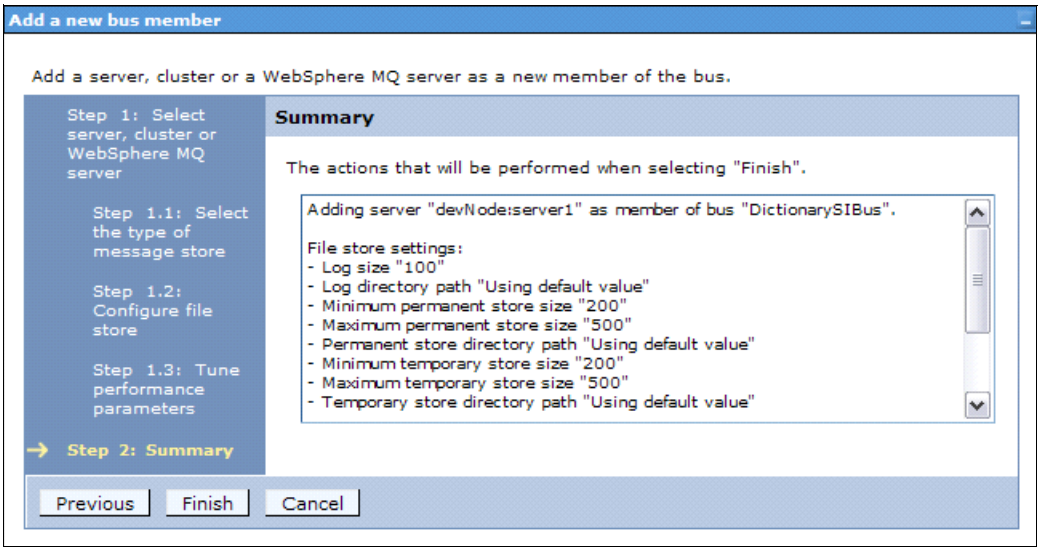


Figure 5-14 Administrative console: Add a new bus member: Summary

After creating the new bus member, you can get command assistance by clicking **View administrative scripting command for last action** in the help box. The output looks similar to Figure 5-15. Remember to return to save your changes after viewing the scripting commands.

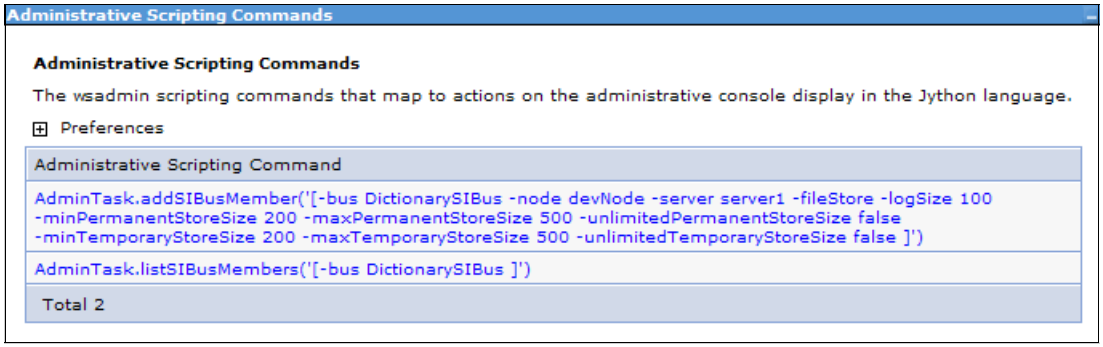


Figure 5-15 Administrative console: Administrative Scripting Commands

13. After you have added your server as a bus member, go back to the Buses page and select the SIBus that you have created. Under the Destination resources section, select **Destinations**. This option takes you to **Buses** → **DictionarySIBus** → **Destinations**, which contains a list of destinations supported by the SIBus, as shown in Figure 5-16. Click **New** to create a new destination.

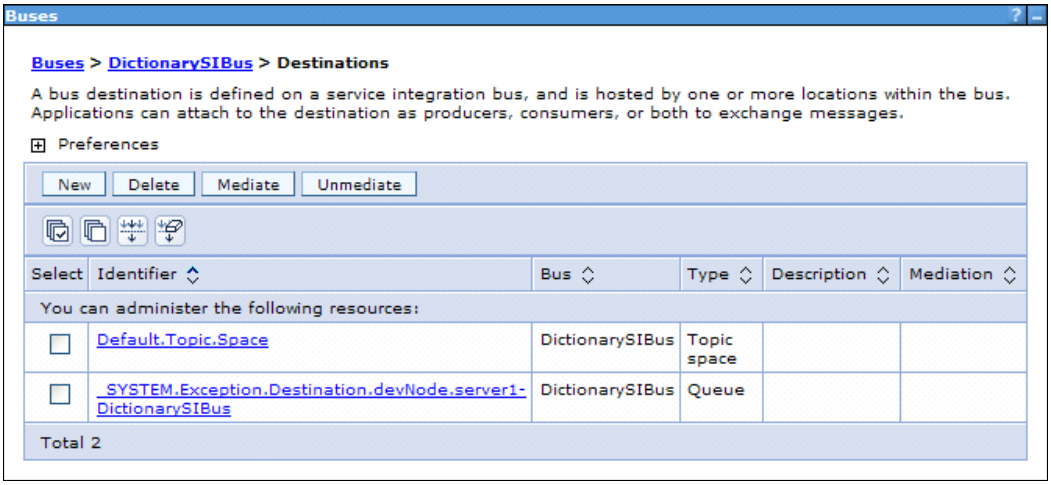


Figure 5-16 Administrative console: Destinations

14. Select the type of destination that you want to create. For DictionaryApp V3, because LookupLoggerBean.java subscribes to a JMS queue, select **Queue**, as shown in Figure 5-17. Click **Next**.

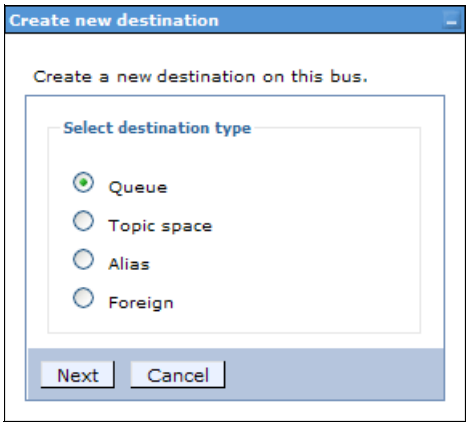


Figure 5-17 Administrative console: Create new destination

15. Assign a name to the destination and a description for the destination (optional). For DictionaryApp V3, enter `DictionarySIBQueue` for Identifier and leave the Description section blank, as shown in Figure 5-18. Click **Next**.

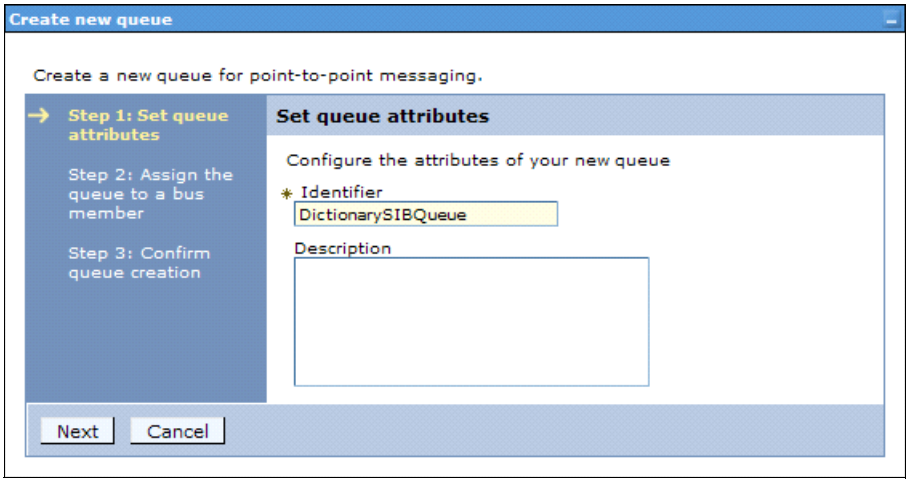


Figure 5-18 Administrative console: Set queue attributes

16. Select the Bus member with which the queue will be associated. All messages that are sent to this destination are processed and stored by the bus member. For DictionaryApp V3, select the member that includes your current server, as shown in Figure 5-19. Click **Next**.

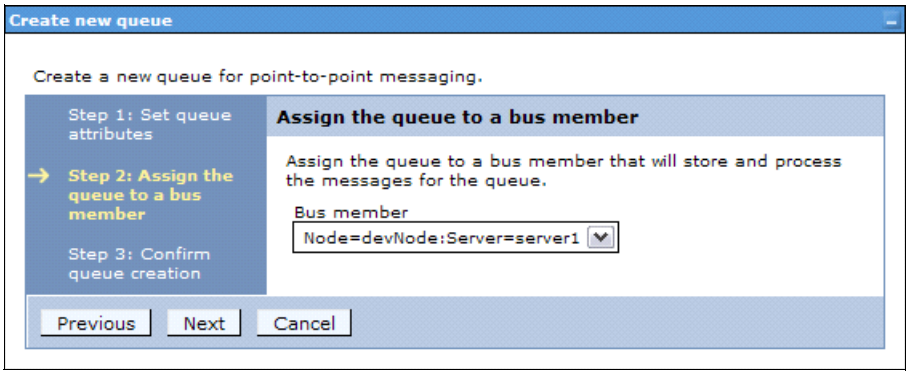


Figure 5-19 Administrative console: Assign the queue to a bus member

17. In the Confirm queue creation page, review your choices. Click **Finish** to create DictionarySIBQueue, as shown in Figure 5-20.

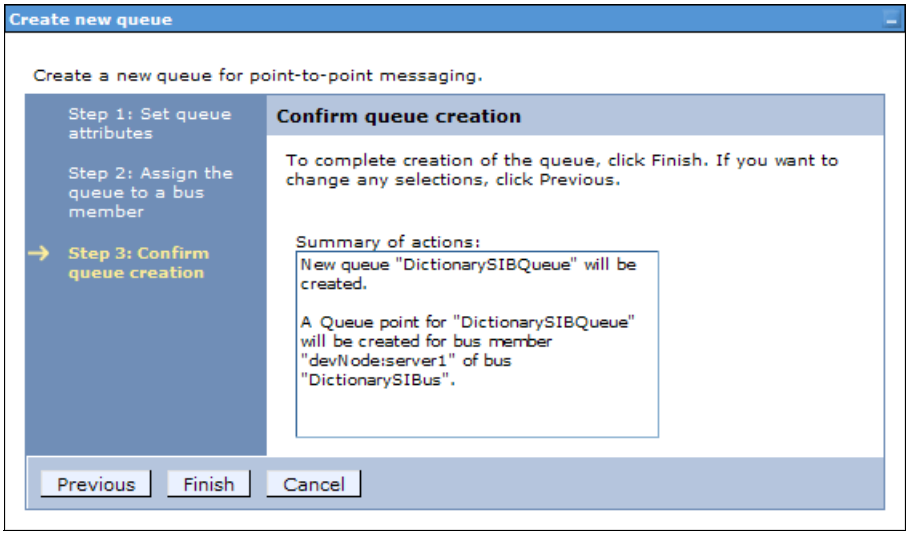


Figure 5-20 Administrative console: Confirm queue creation summary

After creating the new queue, you can get command assistance by clicking **View administrative scripting command for last action** in the help box. The output looks similar to Figure 5-21. Remember to return to save your changes after viewing the scripting commands.

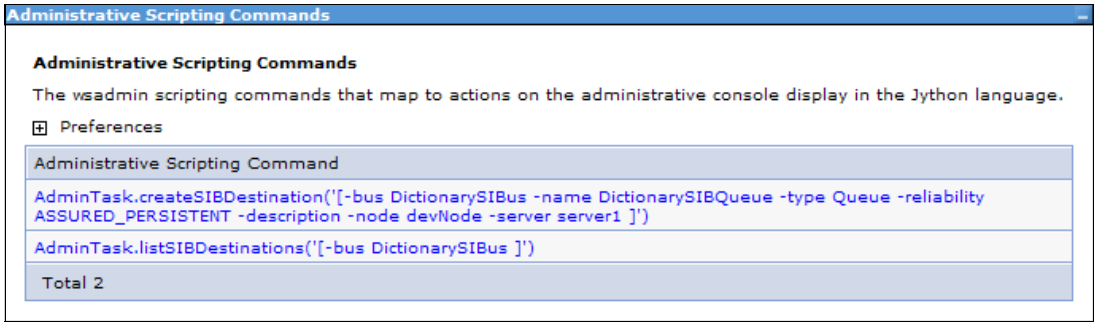


Figure 5-21 Administrative console: Administrative Scripting Commands

18. Go to the Buses page, and select **DictionarySIBus**. Under the Destination section, select **Destinations**. This option takes you back to **Buses** → **DictionarySIBus** → **Destinations**. Click **New** to create a new bus destination.
19. Select the type of destination that you want to create. For DictionaryApp V3, because DefineLoggerBean.java subscribes to a JMS topic, select **Topic Space**, as shown in Figure 5-22. Click **Next**.

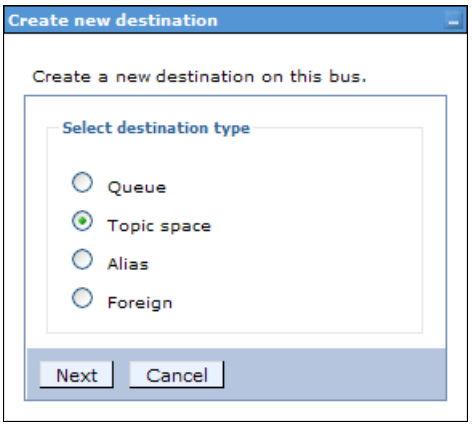


Figure 5-22 Administrative console: Create new destination

20. Assign a name to the destination and a description for the destination (optional). For DictionaryApp V3, enter `DictionarySIBTopic` for Identifier and leave the Description section blank, as shown in Figure 5-23. Click **Next**.

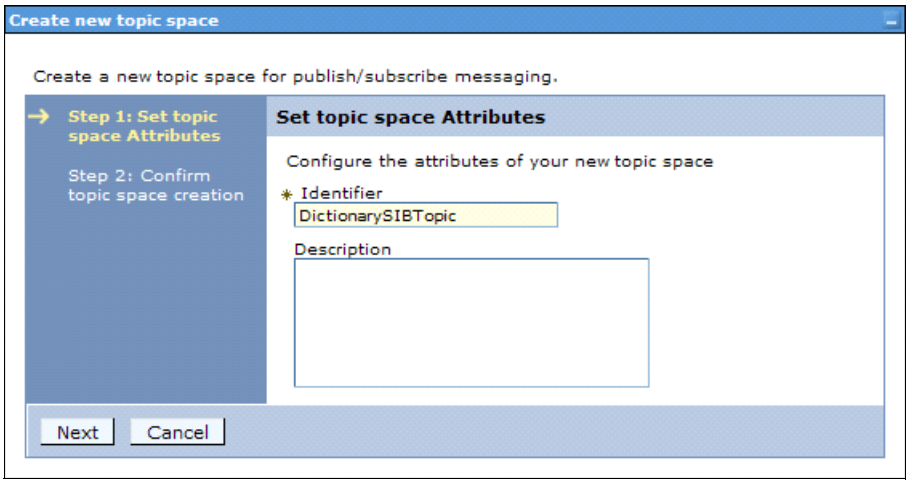


Figure 5-23 Administrative console: Set topic space Attributes

21. In the Confirm topic space creation page, review your selections, and click **Finish** to create the DictionarySIBTopic, as shown in Figure 5-24.

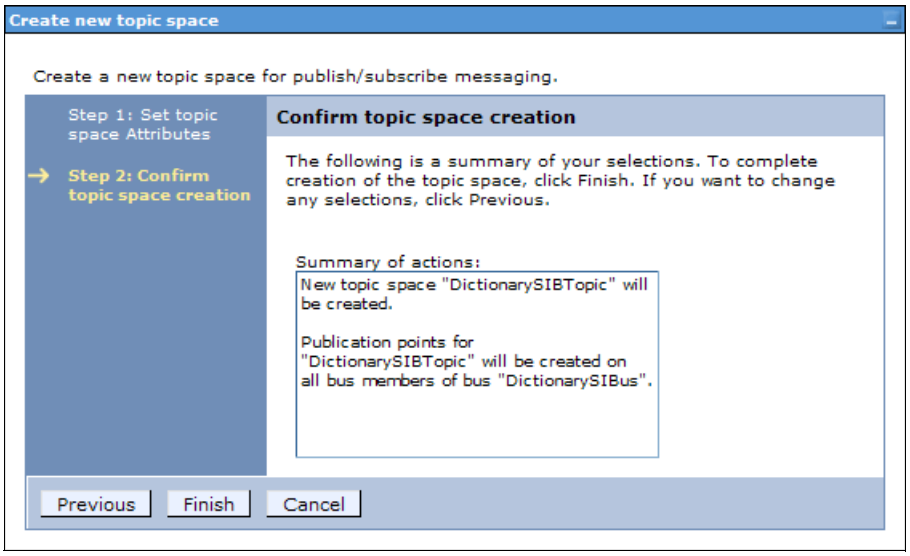


Figure 5-24 Administrative console: Create new topic space summary

After creating the topic space, you can get command assistance by clicking **View administrative scripting command for last action** in the help box. The output looks similar to Figure 5-25. Remember to return to save your changes after viewing the scripting commands.

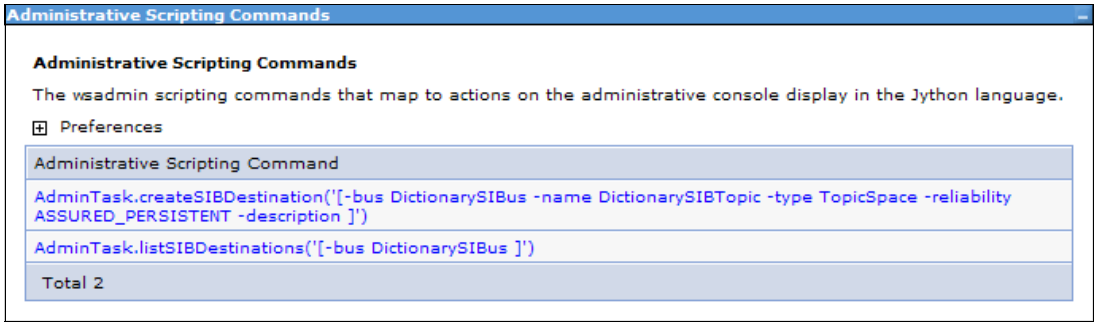


Figure 5-25 Administrative console: Administrative Scripting Commands

22. After configuring and saving the SIBus, restart the server. By including your server as a member of the newly created SIBus, WebSphere automatically creates a message engine for this member. This messaging engine cannot obtain the resources to begin processing messages until you restart the server. Additionally, if a newly created SIBus does not provide messaging services, it might be helpful to check the `SystemOut.log` and `startServer.log` files to troubleshoot the problem.

5.4.2 wsadmin scripting

To create an SIBus in **`wsadmin`**, use the following command:

```
AdminTask.createSIBus('-bus desired SIBus name -busSecurity FALSE');
```

The *`desired SIBus name`* defines the name of the SIBus to be created. For more options that are associated with this command, go to this website:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.pmc.express.doc/ref/rjj_cli_bus_create.html

For DictionaryApp V3, the script looks like this script:

```
AdminTask.createSIBus('-bus DictionarySIBus -busSecurity FALSE');
```

To add a member to a SIBus, use the following command:

```
AdminTask.addSIBusMember('-bus SIBus name -node node name -server server name
-fileStore');
```

The *`SIBus name`* defines the name of the SIBus to which the member is to be added. The *`server name`* and *`node name`* define the member to be added to the SIBus. For more options associated with this command, go to this website:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.pmc.express.doc/ref/rjj_cli_busm_add.html

For DictionaryApp V3, the script looks like this script:

```
AdminTask.addSIBusMember('-bus DictionarySIBus -node devNode -server server1
-fileStore');
```

To add a destination to a SIBus, use the following command:

```
AdminTask.createSIBDestination('-bus SIBus name -type type -name desired
Destination name -node node name -server server name');
```

The *SIBus name* defines the name of the SIBus to which the member is to be added. The *type* specifies the type of destination to be created. This type can be Queue or TopicSpace. The *desired Destination name* specifies the name for the created destination. The *server name* and *node name* define the member to be added to the SIBus. For more options associated with this command, go to this website:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.pmc.express.doc/ref/rjo_cli_dest_create.html

For DictionaryApp V3, the script looks like this script:

```
AdminTask.createSIBDestination('-bus DictionarySIBus -type TopicSpace -name
DictionarySIBTopic -node devNode -server server1')

AdminTask.createSIBDestination('-bus DictionarySIBus -type Queue -name
DictionarySIBQueue -node devNode -server server1')
```

5.5 Setting up the client (message sender) side of DictionaryApp V3

On the client side, DictionaryManager serves as the message sender with capabilities to send to both DictionarySIBQueue and DictionarySIBTopic.

You must configure the following components within the application server to provide access to the message engine within DictionarySIBus:

- JMS queue connection factory: to be named DictionaryQueueFactory
- JMS topic connection factory: to be named DictionaryFactory

You must configure these additional components to provide access to DictionarySIBQueue and DictionarySIBTopic:

- JMS queue: to be named DictionaryJMSQueue
- JMS topic: to be named DictionaryJMSTopic

We provide the instructions for creating the client side JMS resources in the following sections.

5.5.1 Administrative console

This section provides instructions for creating DictionaryQueueFactory, DictionaryJMSQueue, DictionaryTopicFactory, and DictionaryJMSTopic. This section includes the following procedures:

- JMS queue and connection factory
- JMS topic and connection factory

JMS queue and connection factory

Use the following procedure to configure the JMS queue and connection factory:

1. In the administrative console, click **Resources** → **JMS** → **Queue connection factories** in the navigation tree.
2. In the drop-down list box under the Scope section, select the preferred scope for the connection factory. For DictionaryApp V3, use the scope containing your server, as shown in Figure 5-26. Click **New**.

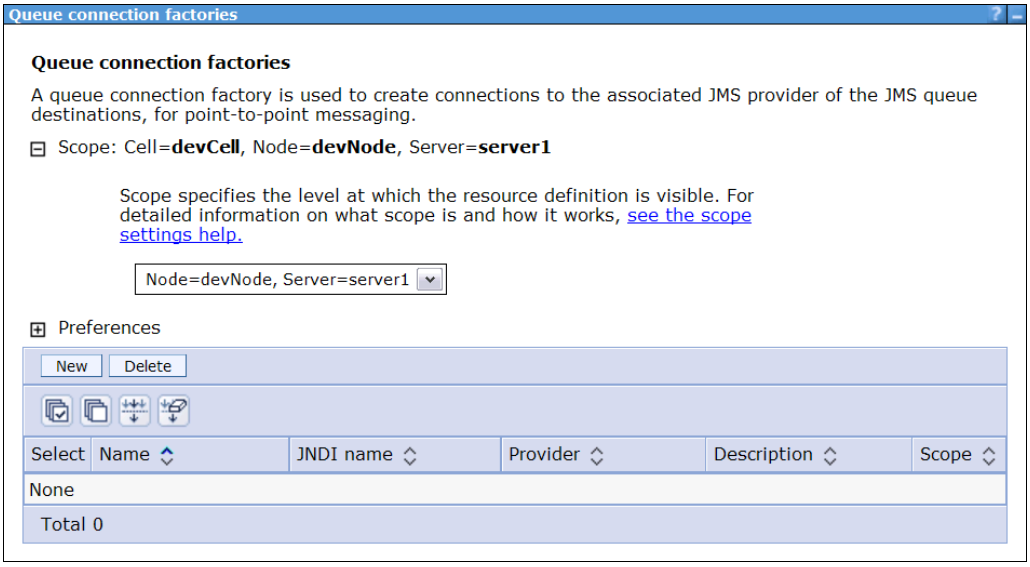


Figure 5-26 Administrative console: Queue connection factories

3. Select the messaging provider for the connection factory. For DictionaryApp V3, choose **Default messaging provider**, as shown in Figure 5-27. Click **OK**.

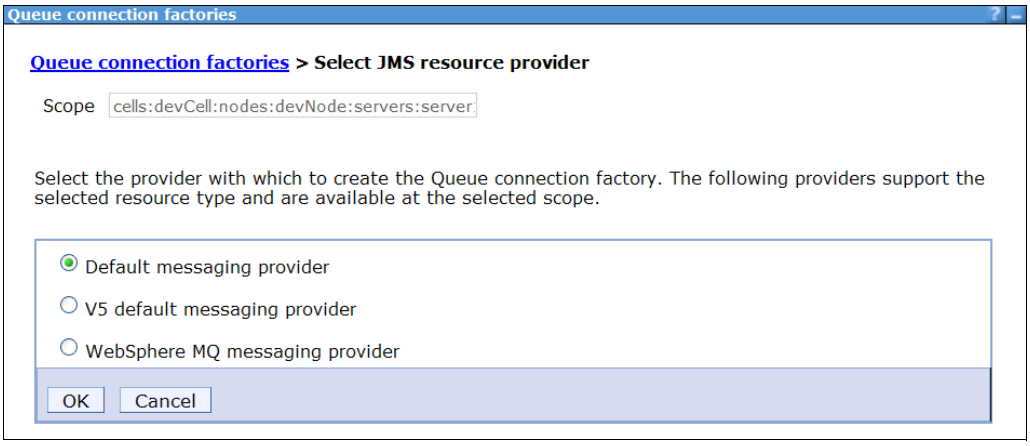


Figure 5-27 Administrative console: Select JMS resource provider

4. Configure the connection factory as necessary. For DictionaryApp V3, specify `DictionaryQueueFactory` as the Name, `jms/DictionaryQCF` as the JNDI name, and `DictionarySIBus` as the Bus name, as shown in Figure 5-28. Click **OK** to create the queue connection factory.

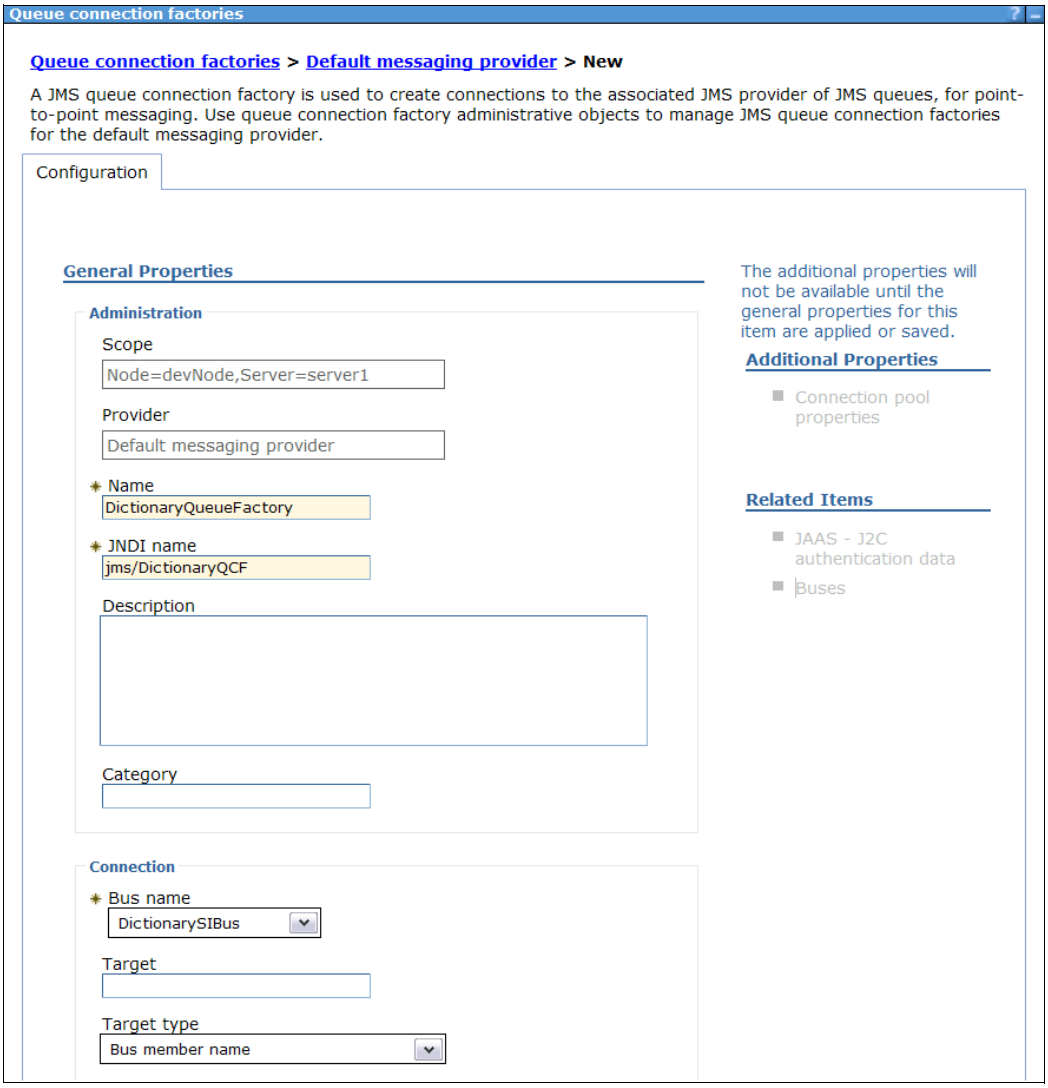


Figure 5-28 Administrative console: Default messaging provider

After creating the new queue connection factory, you can get command assistance by clicking **View administrative scripting command for last action** in the help box. The output looks similar to Figure 5-29. Remember to return to save your changes after viewing the scripting commands.

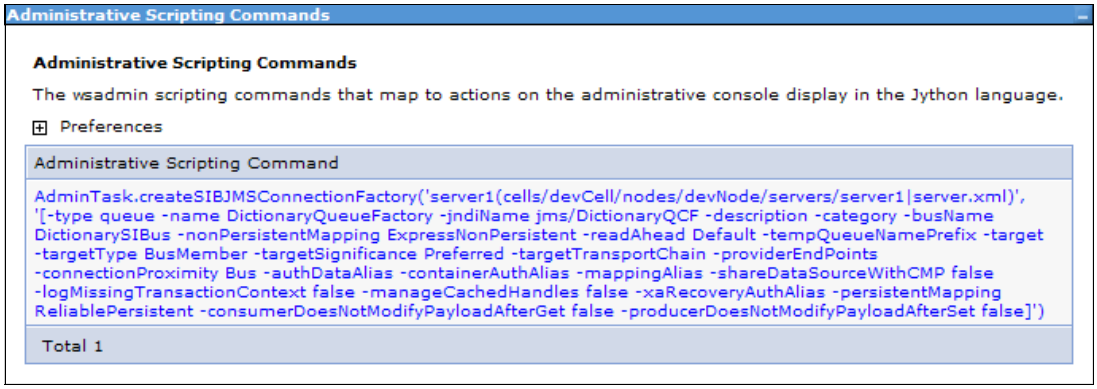


Figure 5-29 Administrative console: Administrative Scripting Commands

5. In the administrative console, click **Resources** → **JMS** → **Queues** in the navigation tree.
6. In the drop-down list box under Scope, select the appropriate scope for the queue. For DictionaryApp V3, use the scope containing your server, as shown in Figure 5-30. Click **New**.

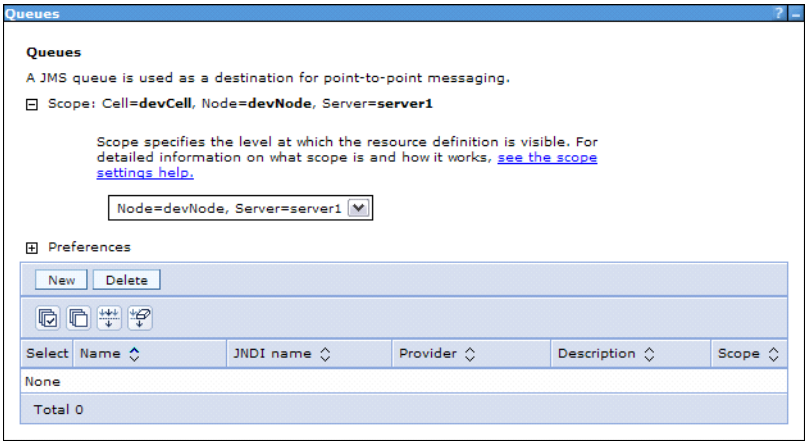


Figure 5-30 Administrative console: Queues

7. Select a JMS resource provider. For DictionaryApp V3, select **Default messaging provider**, as shown in Figure 5-31. Click **OK**.

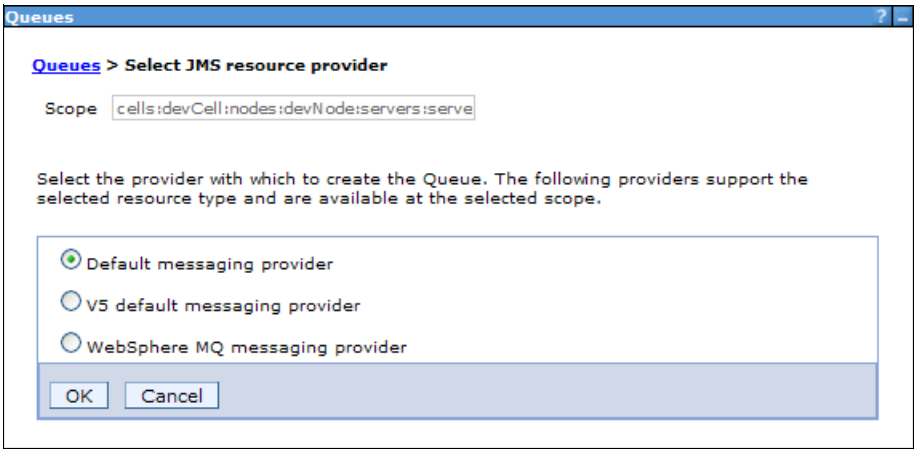


Figure 5-31 Administrative console: Select JMS resource provider

8. Configure the queue, as necessary. For DictionaryApp V3, specify `DictionaryJMSQueue` in the Name field, `jms/DictionaryQ` for the JNDI name, `DictionarySIBus` for the Bus name, and `DictionarySIBQueue` for the SIBus for Queue name, as shown in Figure 5-32. Click **OK**.

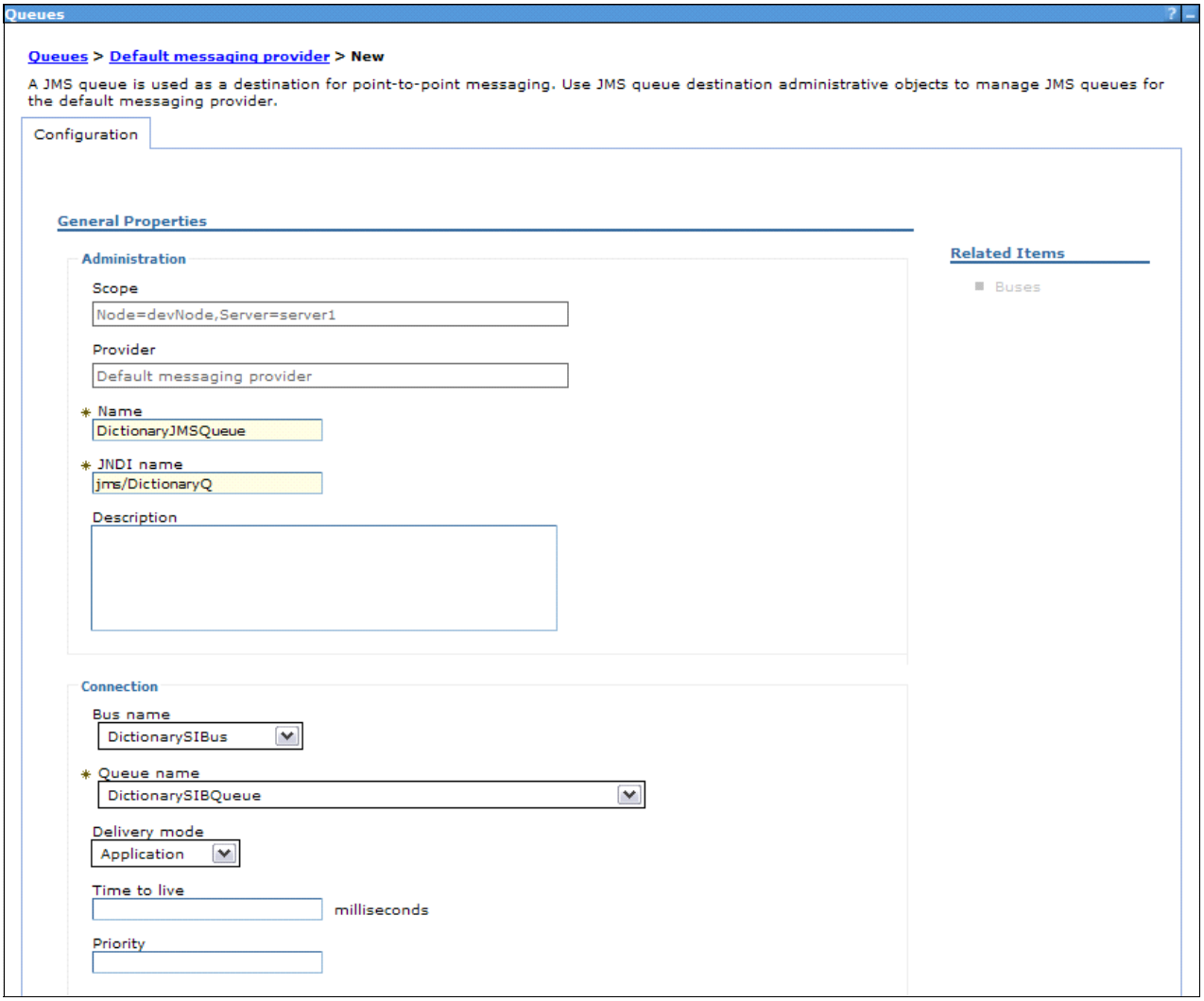


Figure 5-32 Administrative console: New queue

After creating the new queue, you can get command assistance by clicking **View administrative scripting command for last action** in the help box. The output looks similar to Figure 5-33. Remember to return to save your changes after viewing the scripting commands.

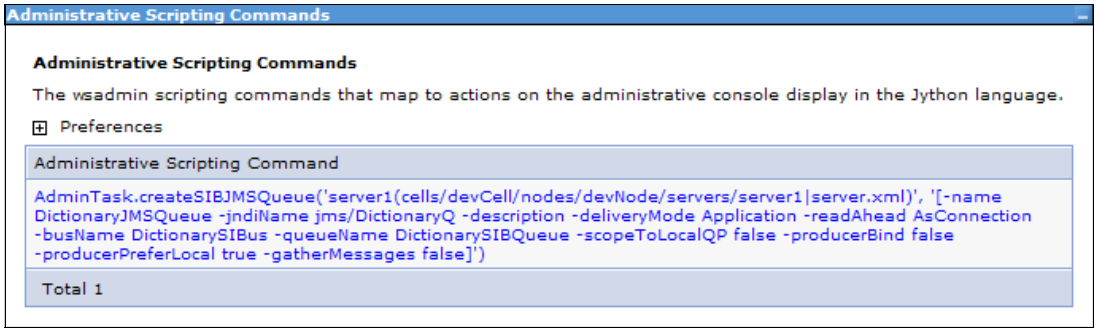


Figure 5-33 Administrative console: Administrative Scripting Commands

JMS topic and connection factory

Use the following procedure to configure the JMS topic and connection factory:

1. In the administrative console, click **Resources** → **JMS** → **Topic connection factories** in the navigation tree.
2. In the drop-down list box under Scope, select the preferred scope for the connection factory. For DictionaryApp V3, use the scope containing your server, as shown in Figure 5-34. Click **New**.

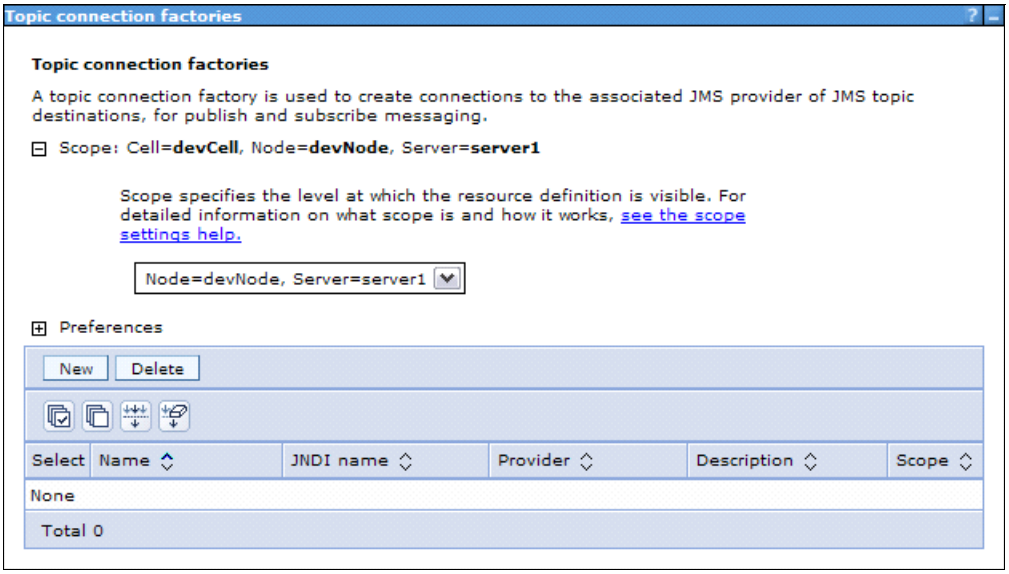


Figure 5-34 Administrative console: Topic connection factories

3. Select the messaging provider for your connection factory. For DictionaryApp V3, choose **Default messaging provider**, as shown in Figure 5-35. Click **OK.**

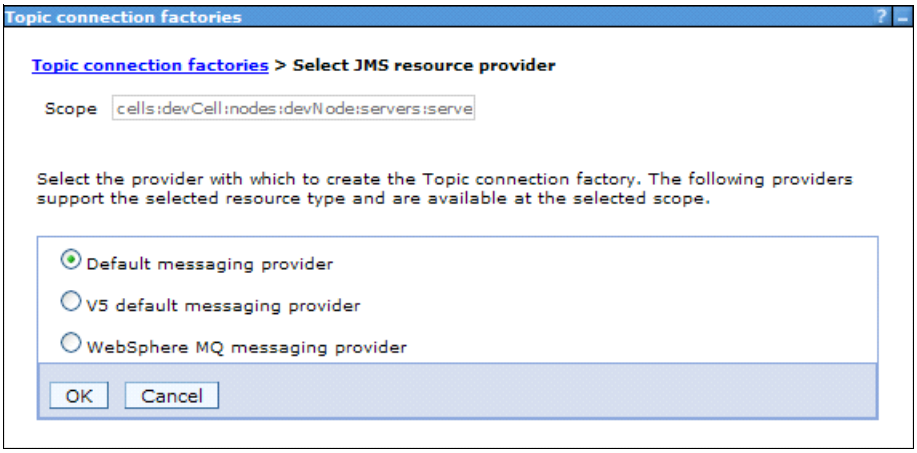


Figure 5-35 Administrative console: Select JMS resource provider

4. Configure the connection factory, as necessary. For DictionaryApp V3, specify `DictionaryTopicFactory` for Name, `jms/DictionaryTCF` for the JNDI name, and `DictionarySIBus` for the SIBus name, as shown in Figure 5-36. Click **OK.**

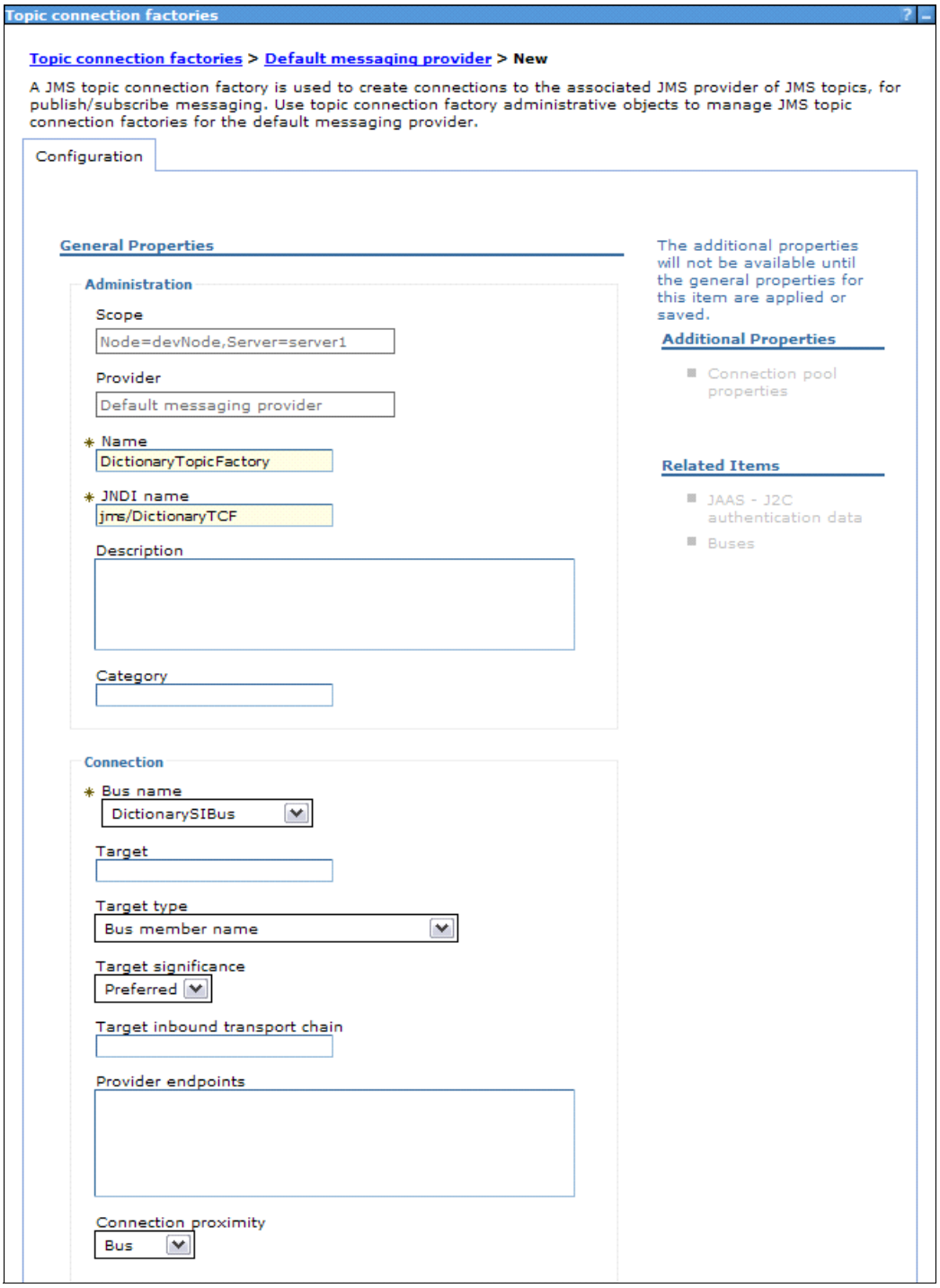


Figure 5-36 Administrative console: New topic connection factory

After creating the new topic connection, you can get command assistance by clicking **View administrative scripting command for last action** in the help box. The output looks similar to Figure 5-37. Remember to return to save your changes after viewing the scripting commands.

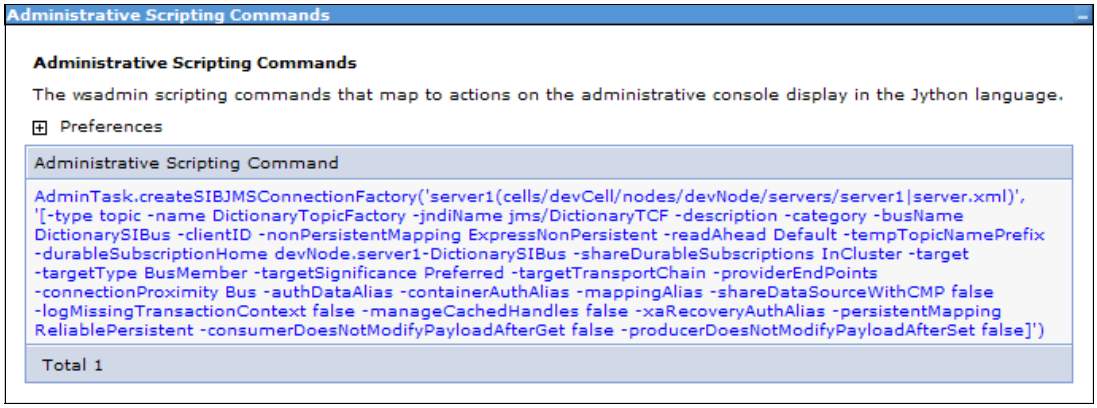


Figure 5-37 Administrative console: Administrative Scripting Commands

5. In the administrative console, click **Resources** → **JMS** → **Topics** in the navigation tree.
6. Select the appropriate scope for the topic. For DictionaryApp V3, use the scope containing `server1`, as shown in Figure 5-38. Click **New**.

Figure 5-38 Administrative console: Topics

7. Select the JMS provider. For DictionaryApp V3, select **Default messaging provider**, as shown in Figure 5-39. Click **OK**.

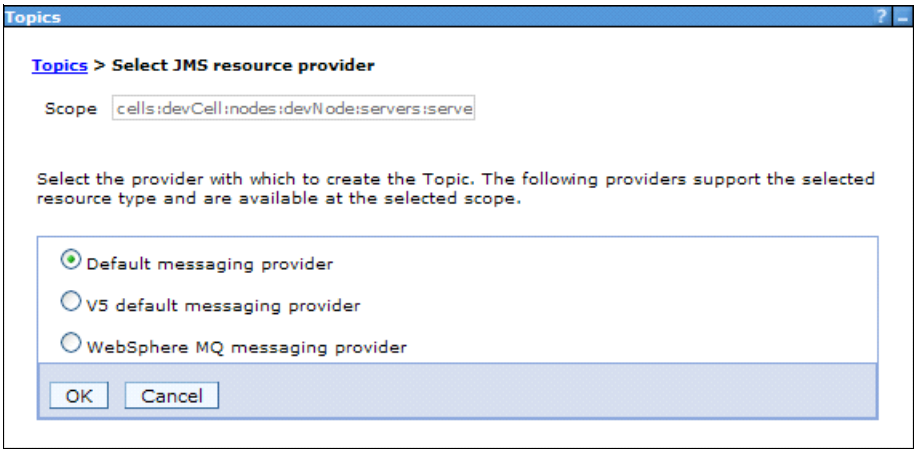


Figure 5-39 Administrative console: Select JMS resource provider

8. Configure the topic, as necessary. For DictionaryApp V3, specify `DictionaryJMSTopic` for Name, `jms/DictionaryT` for the JNDI name, `DictionarySIBus` for the SIBus name, and `DictionarySIBTopic` as the Topic space, as shown in Figure 5-40. Click **OK**.

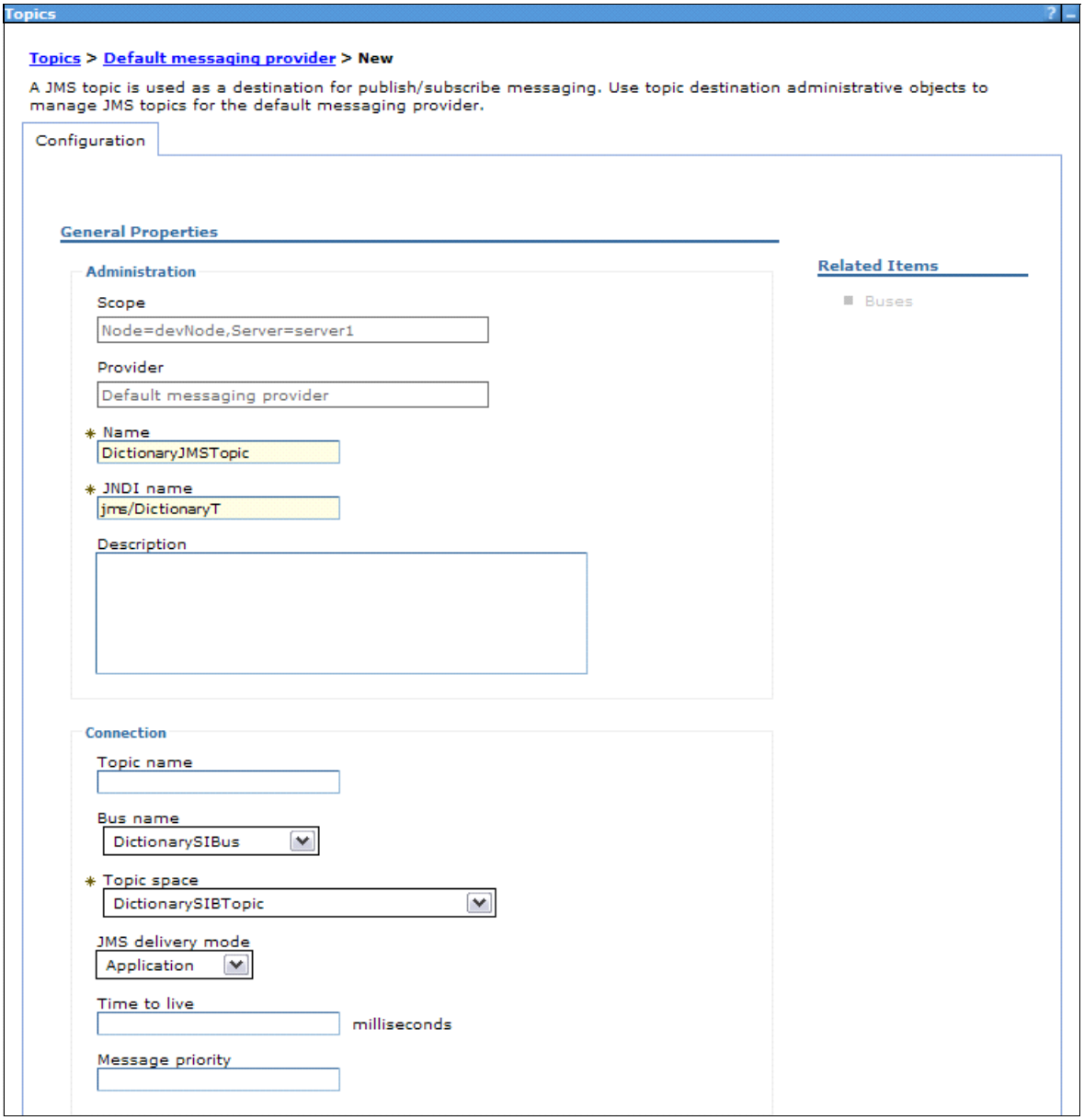


Figure 5-40 Administrative console: New topic

After creating the new topic, you can get command assistance by clicking **View administrative scripting command for last action** in the help box. The output looks similar to Figure 5-41. Remember to return to save your changes after viewing the scripting commands.

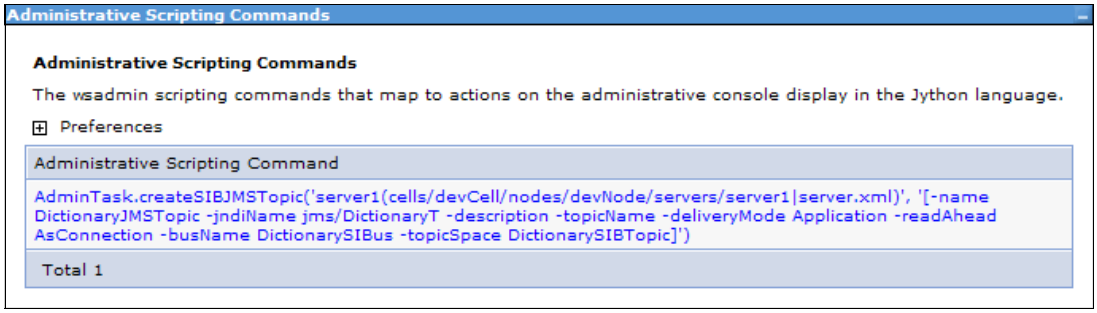


Figure 5-41 Administrative console: Administrative Scripting Command

5.5.2 wsadmin scripting

Before administering any of the commands for creating JMS resources, obtain a reference to the scope from which you want to create the resource by using the following command:

```
id = AdminConfig.getid("/Server:server name");
```

The identifier id serves as the reference to the server scope for configuring the JMS resources. If you use the default profile that was configured during installation, the script looks like this script:

```
id = AdminConfig.getid("/Server:server1");
```

To create a connection factory in **wsadmin**, use the following command:

```
AdminTask.createSIBJMSConnectionFactory(id, '-name Connection Factory name
-jndiName Connection Factory JNDI name -type type -busName SIBus name');
```

The *Connection Factory name* defines the name of the connection factory to create. The *Connection Factory JNDI name* specifies the JNDI name for the connection factory. The *type* specifies the type of connection factory to make, which can be either queue or topic. The *SIBus name* specifies the SIBus from which the connection factory obtains its connections. For more options associated with this command, go to this website:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.pmc.express.doc/ref/rjn_jmscf_create.html

For DictionaryApp V3, the script looks like this example:

```
AdminTask.createSIBJMSConnectionFactory(id, '-name DictionaryQueueFactory
-jndiName jms/DictionaryQCF -type queue -busName DictionarySIBus');

AdminTask.createSIBJMSConnectionFactory(id, '-name DictionaryTopicFactory
-jndiName jms/DictionaryTCF -type topic -busName DictionarySIBus');
```

To create a JMS queue in **wsadmin**, use the following command:

```
AdminTask.createSIBJMSQueue(id, '-name Queue name -jndiName Queue JNDI name
-busName SIBus name -queueName SIBus Destination name');
```

The *Queue name* defines the name of the JMS queue to create. The *Queue JNDI name* specifies the JNDI name for the JMS queue. The *SIBus Destination name* specifies the

SIBus destination with which the JMS queue is associated. For more options associated with this command, go to this website:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.pmc.express.doc/ref/rjn_jmsqueue_create.html

For DictionaryApp V3, the script looks like this example:

```
AdminTask.createSIBJMSQueue(id, '-name DictionaryJMSQueue -jndiName
jms/DictionaryQ -busName DictionarySIBus -queueName DictionarySIBQueue');
```

To create an JMS topic in **wsadmin**, use the following command:

```
AdminTask.createSIBJMSTopic(id, '-name <Topic name> -jndiName Topic JNDI name
-busName SIBusname -topicSpace SIBus Destination name');
```

The *Topic name* defines the name of the JMS topic to create. The *Topic JNDI name* specifies the JNDI name for the JMS topic. The *SIBus Destination name* specifies the SIBus destination with which the JMS topic will be associated. For more options that are associated with this command, go to this website:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.pmc.express.doc/ref/rjn_jmstopic_create.html

For DictionaryApp V3, the script looks like this example:

```
AdminTask.createSIBJMSTopic(id, '-name DictionaryJMSTopic -jndiName
jms/DictionaryT -busName DictionarySIBus -topicSpace DictionarySIBTopic');
```

5.6 Setting up the server (message receiver) side of DictionaryApp V3

DictionaryApp V3 contains both the message senders and message receivers. On the server side, the MDB, DefineLoggerBean, is the message receiver linked with the JMS queue (jms/DictionaryQ), and the MDB, LookupLoggerBean, is the message receiver linked with the JMS topic (jms/DictionaryT). Therefore, within the application server, you must configure activation specifications, which consist of a group of message configuration properties for MDBs to retrieve messages from their associated messaging destinations, for each MDB.

5.6.1 Administrative console

This section provides instructions to use the administrative console to configure DictionaryQueueSpec (the activation specifications for LookupLoggerBean) and DictionaryTopicSpec (the activation specifications for DefineLoggerBean). Follow these steps:

1. In the administrative console, click **Resources** → **JMS** → **Activation specifications** in the navigation tree.
2. Select the appropriate scope for the activation specifications. For DictionaryApp V3, select **server1**, as shown in Figure 5-42. Click **New.**

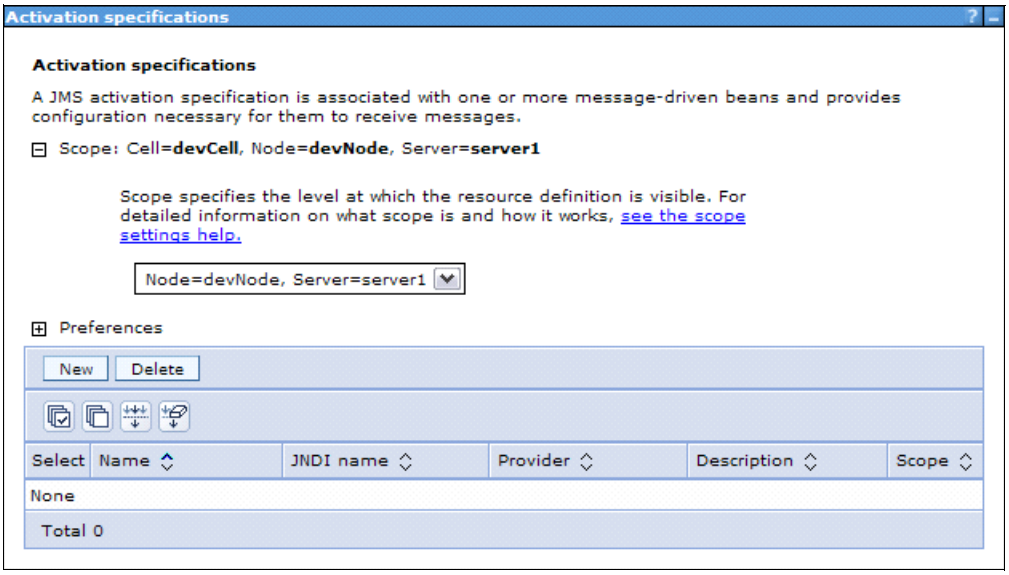


Figure 5-42 Administrative console: Activation specifications

3. Select a JMS provider. For DictionaryApp V3, select **Default messaging provider**, as shown in Figure 5-43. Click **OK.**

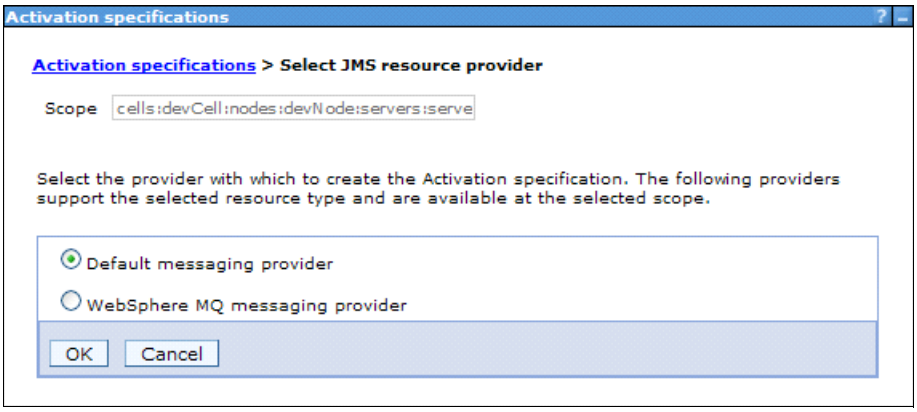


Figure 5-43 Administrative console: Select JMS resource provider

4. Configure the activation specifications, as necessary. For DictionaryApp V3, specify `DictionaryQueueSpec` for the Name, `jms/DictionaryQS` for the JNDI name, `Queue` for the Destination Type, `jms/DictionaryQ` for the Destination JNDI name, and `DictionarySIBus` for the Bus name, as shown in Figure 5-44. Click **OK**.

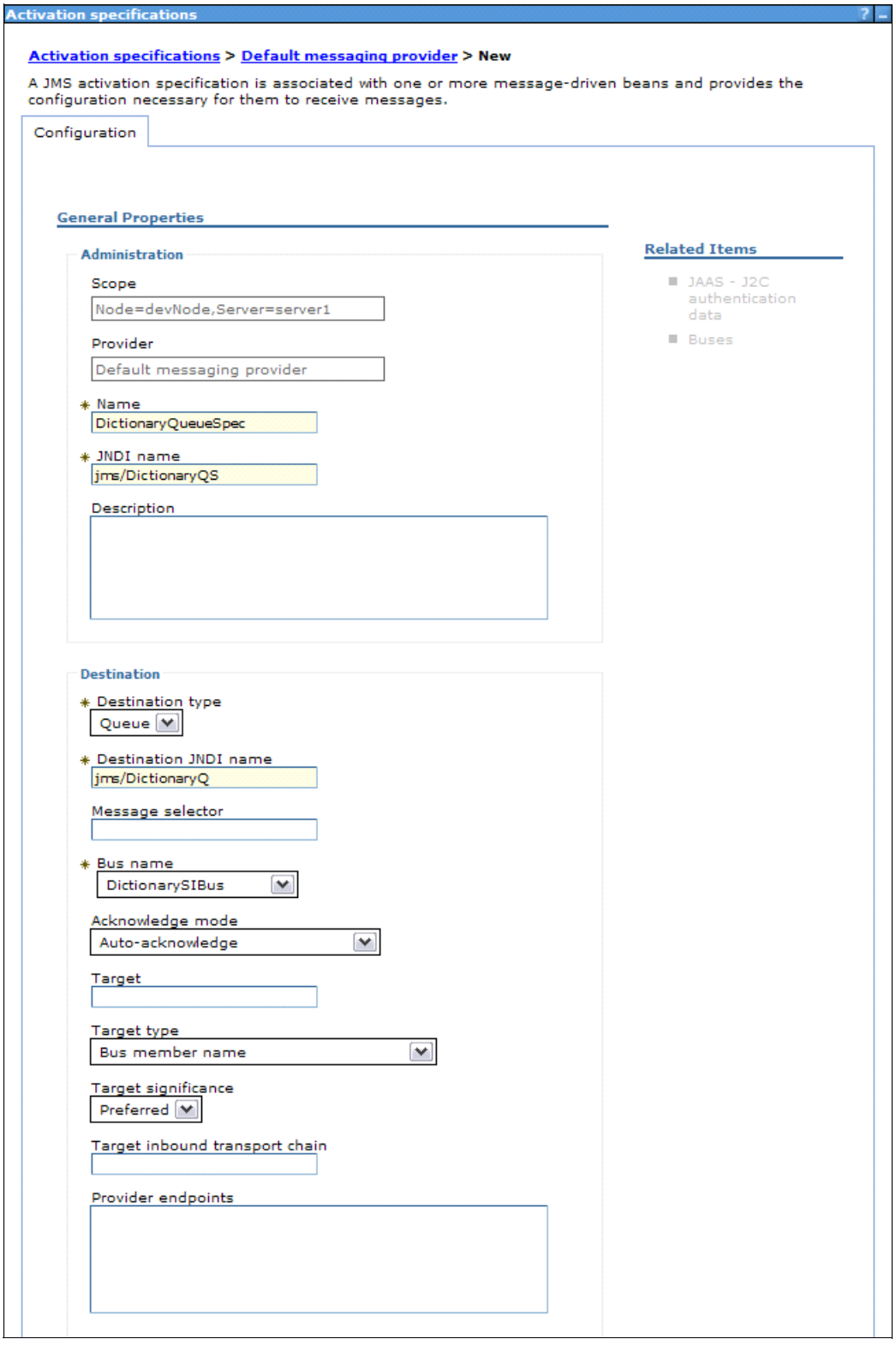


Figure 5-44 Administrative console: New activation specifications

After creating DictionaryQueueSpec, you can get command assistance by clicking **View administrative scripting command for last action** in the help box. The output looks similar to Figure 5-45. Remember to return to save your changes after viewing the scripting commands.

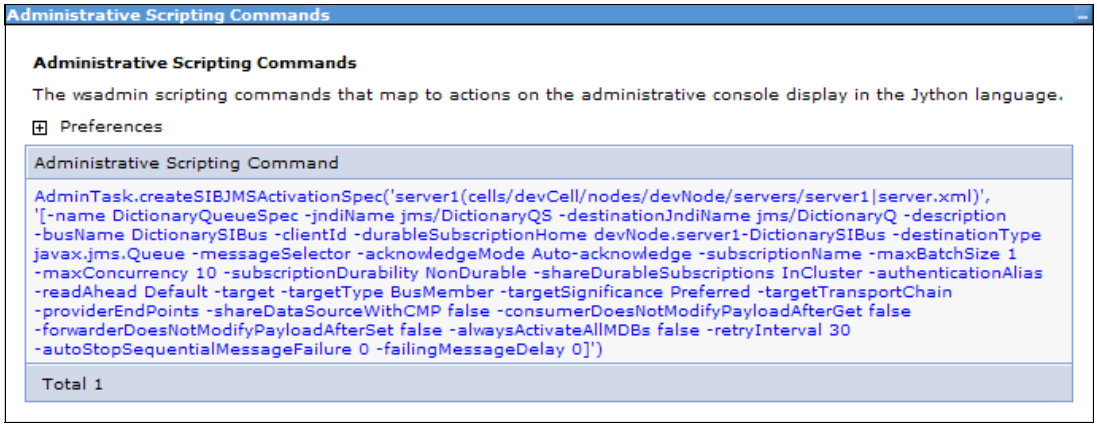


Figure 5-45 Administrative console: Administrative Scripting Commands

5. After creating DictionaryQueueSpec, create DictionaryTopicSpec. The instructions are similar. In the administrative console, click **Resources** → **JMS** → **Activation specifications** in the navigation tree.
6. Select the appropriate scope for the activation specifications. For DictionaryApp V3, select **server1**, as shown in Figure 5-46. Click **New.**

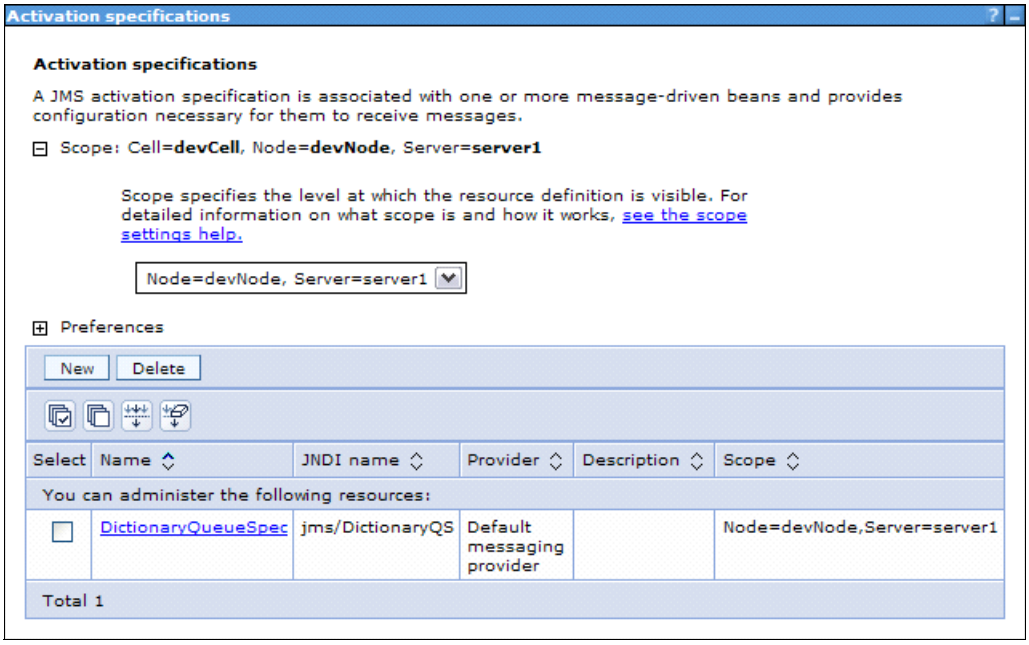


Figure 5-46 Administrative console: Activation specifications

7. Select a JMS provider. For DictionaryApp V3, select **Default messaging provider**, as shown in Figure 5-47. Click **OK**.

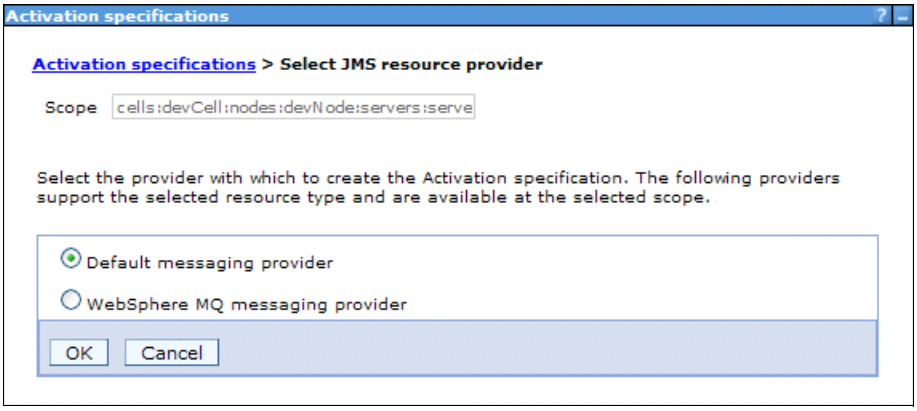


Figure 5-47 Administrative console: Select JMS resource provider

8. Configure the activation specifications, as necessary. For DictionaryApp V3, specify `DictionaryTopicSpec` as the Name, `jms/DictionaryTS` as the JNDI name, `Topic` as the Destination type, `jms/DictionaryT` as the Destination JNDI name, and `DictionarySIBus` as the Bus name, as shown in Figure 5-48. Click **OK**.

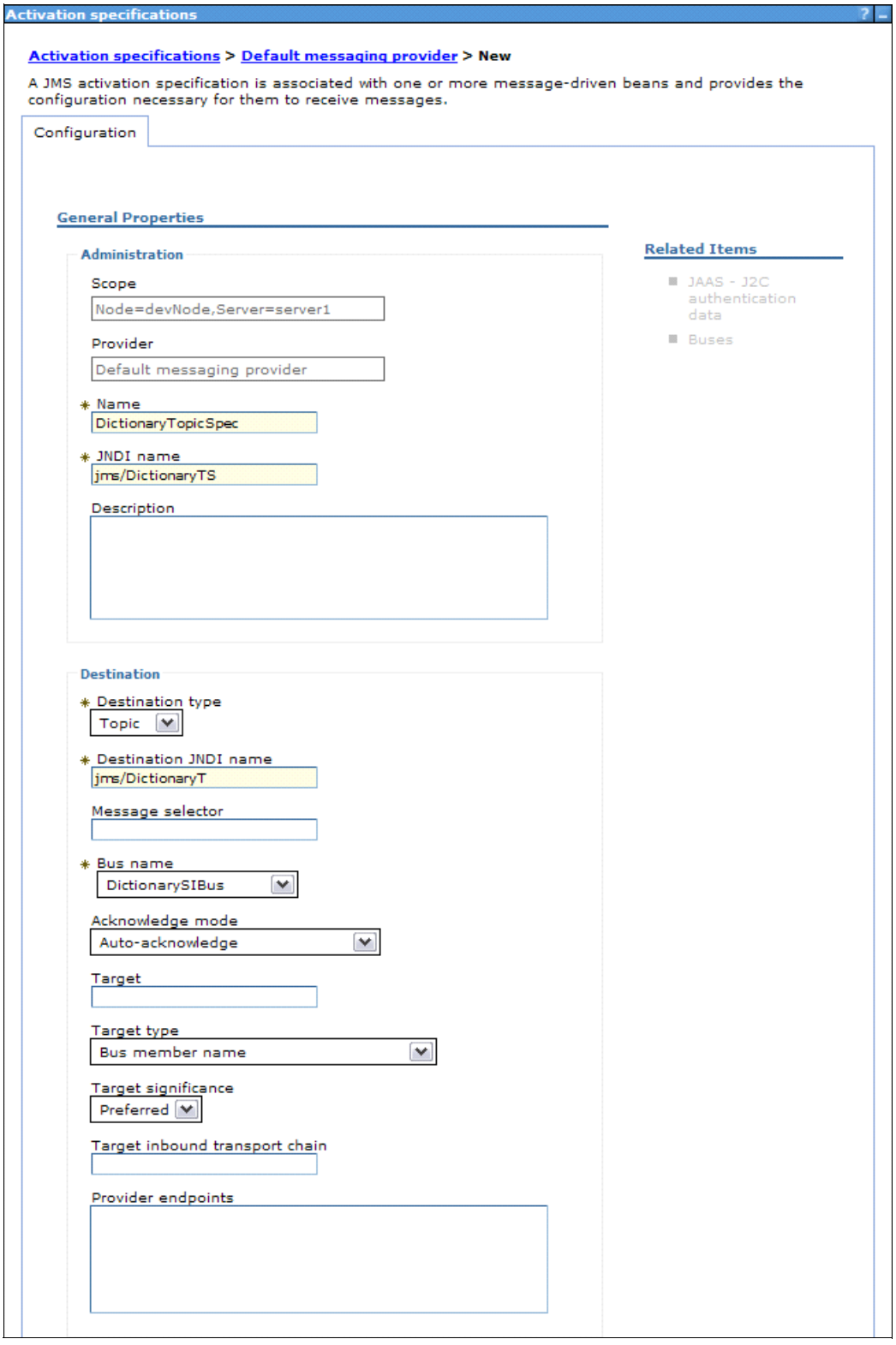


Figure 5-48 Administrative console: New Activation specifications

After creating DictionaryTopicSpec, you can get command assistance by clicking **View administrative scripting command for last action** in the help box. The output looks similar to Figure 5-49. Remember to return to save your changes after viewing the scripting commands.

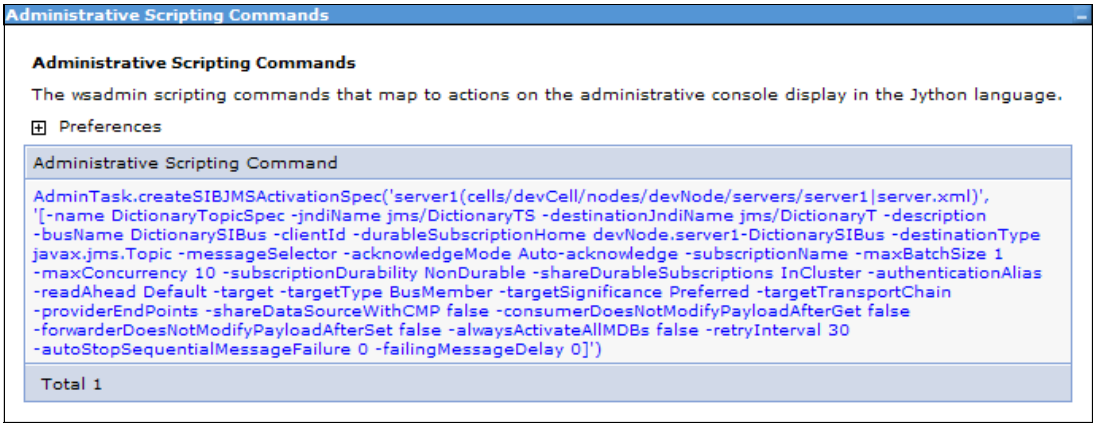


Figure 5-49 Administrative console: Administrative Scripting Commands

5.6.2 wsadmin scripting

Before administering any of the commands to create JMS resources, obtain a reference to the scope from which you want to create the resource using the following command:

```
id = AdminConfig.getid("/Server:server name");
```

The identifier id serves as the reference to the server scope for configuring the JMS resources. If you use the default profile that is configured during installation, the script looks like the following example:

```
id = AdminConfig.getid("/Server:server1");
```

To create JMS activation specifications in **wsadmin**, use the following command:

```
AdminTask.createSIBJMSActivationSpec(id, '-name Activation Specifications name
-jndiName Activation Specifications JNDI name -destinationType Destination Type
-destinationJndiName Destination JNDI name -busName SIBus name);
```

The *Activation Specifications name* defines the name of the activation specifications to be created. The *Activation Specifications JNDI name* specifies the JNDI name for the activation specifications. The *Destination Type* specifies the destination type for which the activation specifications will be created: queue or topic. The *Destination JNDI name* specifies the JMS destination with which the activation specifications will be associated. The *SIBus name* specifies the SIBus to which connections will be made by the activation specifications. For more options that are associated with this command, go to this website:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.pmc.express.doc/ref/rjn_jmsas_create.html

For DictionaryApp V3, the script looks like this example:

```
AdminTask.createSIBJMSActivationSpec(id, '-name DictionaryQueueSpec -jndiName
jms/DictionaryQS -destinationType Queue -destinationJndiName jms/DictionaryQ
-busName DictionarySIBus');
AdminTask.createSIBJMSActivationSpec(id, '-name DictionaryTopicSpec -jndiName
jms/DictionaryTS -destinationType Topic -destinationJndiName jms/DictionaryT
-busName DictionarySIBus');
```

5.7 Redeploying DictionaryApp V3

This section tells you how to redeploy DictionaryApp V3 by using the administrative console or **wsadmin**.

5.7.1 Administrative console

Using the administrative console to deploy DictionaryApp V3 is similar to deploying the previous versions, except for three additional steps that are required for the application server to map JMS resource references to the JMS resources:

1. After the “Map modules to servers” section, the application server prompts you to bind message listener specifications to MDBs. Specify how to bind each MDB in your application to the JMS destinations. For DictionaryApp V3, select **Activation Specification** for LookupLoggerBean and DefineLoggerBean. For LookupLoggerBean, specify `jms/DictionaryQS` as the Target Resource JNDI Name and `jms/DictionaryQ` as the Destination JNDI name. For DefineLoggerBean, specify `jms/DictionaryTS` as the Target Resource JNDI Name and `jms/DictionaryT` as the Destination JNDI name, as shown in Figure 5-50. Click **Next**.

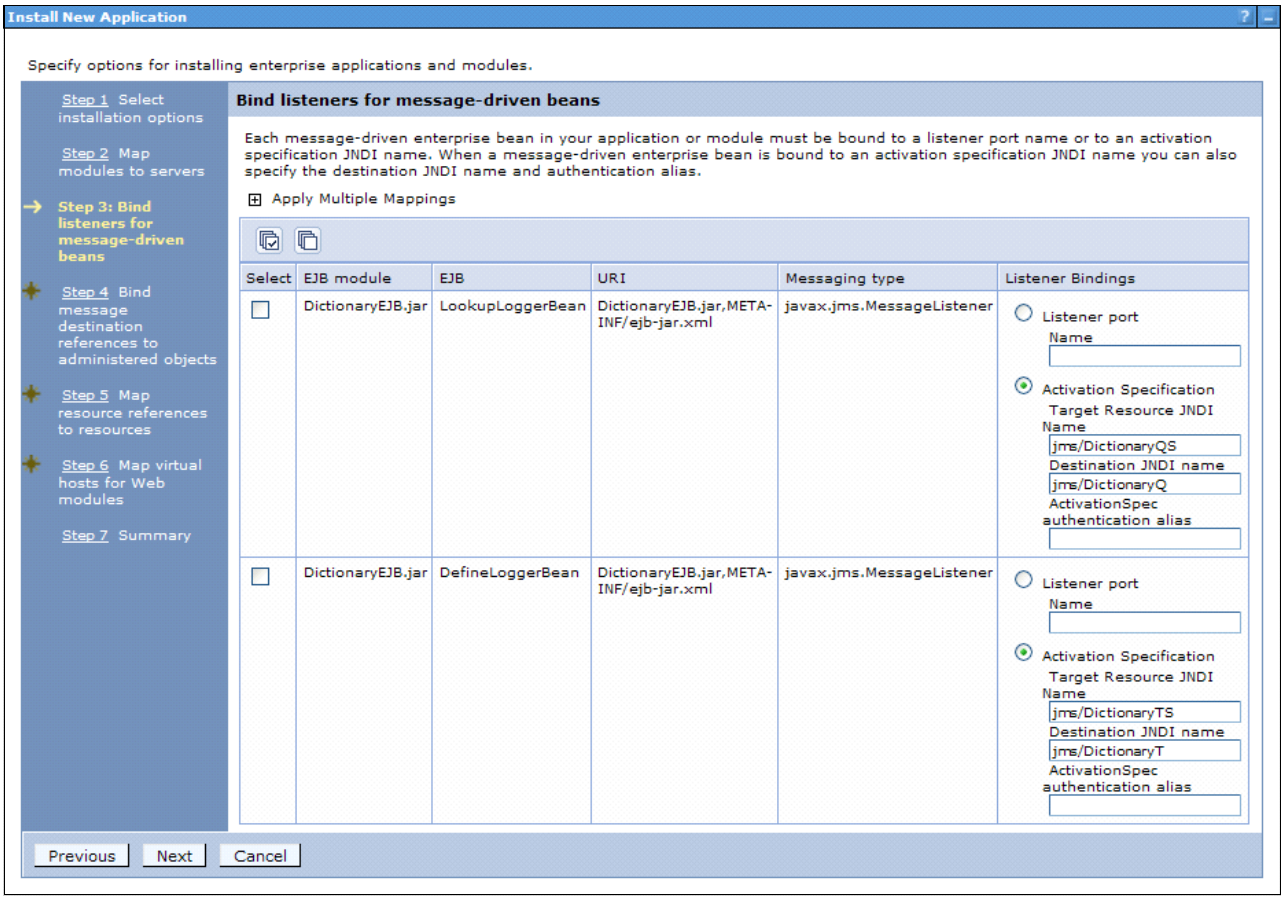


Figure 5-50 Administrative console: Bind listeners for message-driven beans

2. The application server prompts you to resolve the JMS destination resources referenced in your application. For DictionaryApp V3, specify `jms/DictionaryQ` and `jms/DictionaryT`, as shown in Figure 5-51. The field name under Message destination object shows the name under which the resource is referenced by the application. Therefore, it is not confusing if an EJB references multiple destinations. Click **Next**.

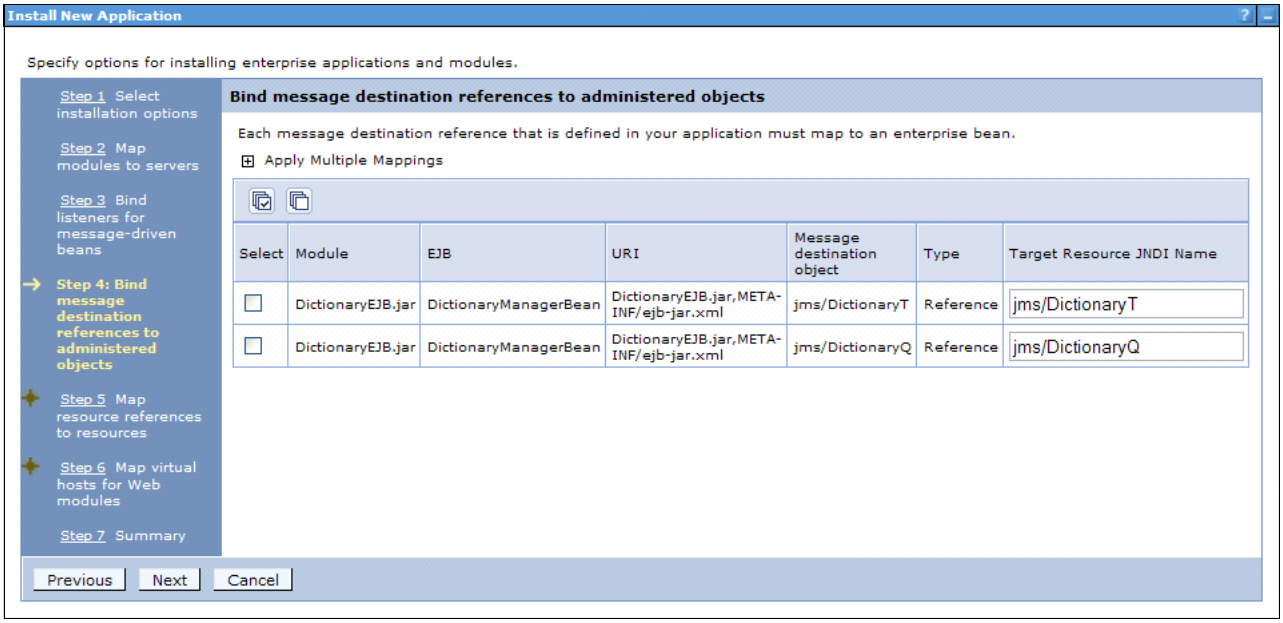


Figure 5-51 Administrative console: Bind message destination references to administered objects

3. The application server takes you to the Map resource references to resources section. You have been to this section before when deploying DictionaryApp V2, but because DictionaryApp V3 references ConnectionFactories in `DictionaryManagerBean.java`, the application server prompts you to resolve it at this point. For DictionaryApp V3, specify `jms/DictionaryQCF` and `jms/DictionaryTCF`. Click **Next**.

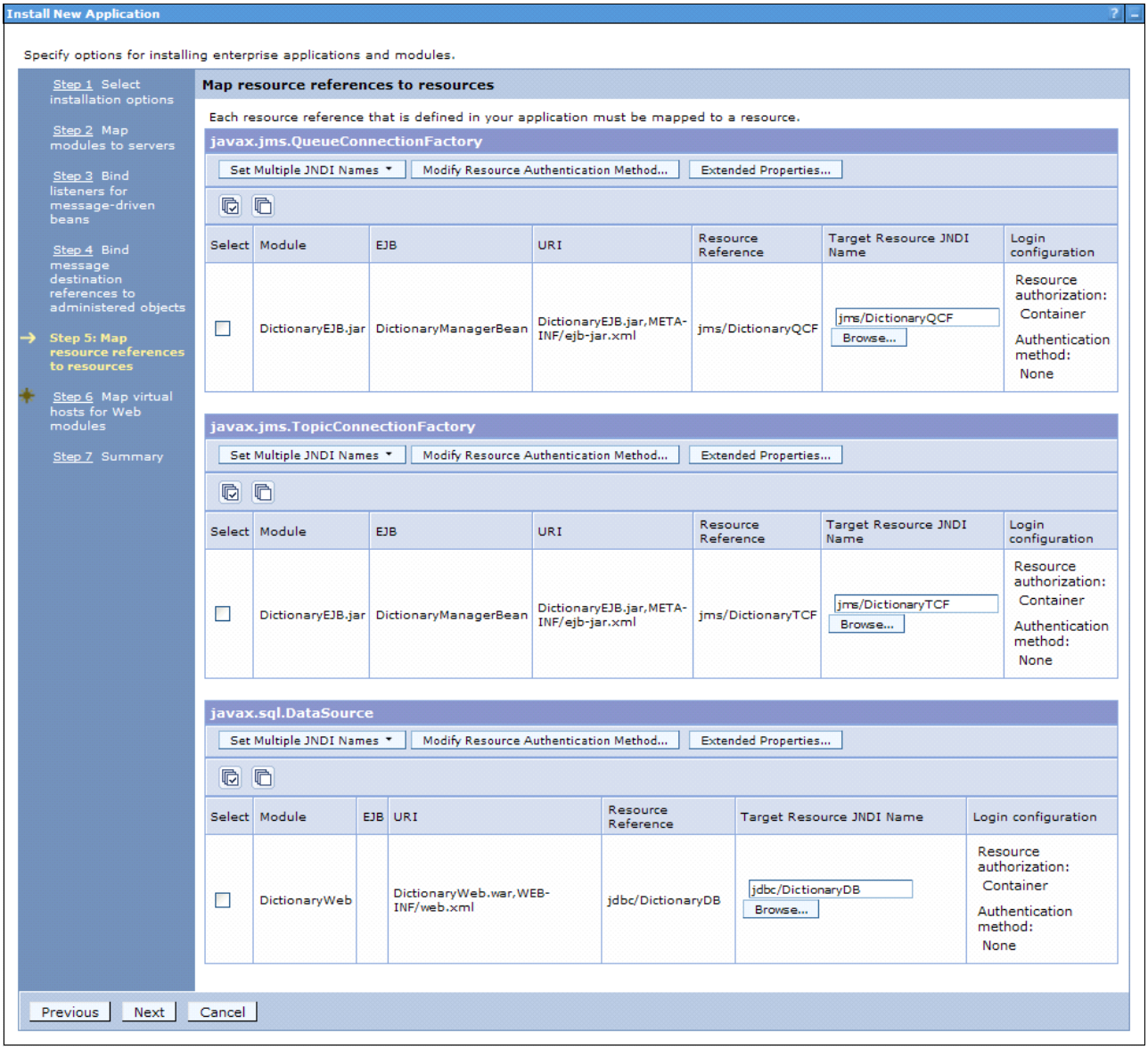


Figure 5-52 Administrative console: Map resource references to resources

The remaining sections of the installation look similar to previous installations.

5.7.2 wsadmin scripting

Example 5-6 is the script for redeploying DictionaryApp V3.

Example 5-6 wsadmin script for redeploying DictionaryApp V3

```
AdminApp.install('C:\DictionaryApp\Version 3\DictionaryApp.ear', '[-appname
DictionaryApp -MapResRefToEJB [[ DictionaryEJB.jar DictionaryManagerBean
DictionaryEJB.jar,META-INF/ejb-jar.xml jms/DictionaryQCF
javax.jms.QueueConnectionFactory jms/DictionaryQCF "" "" "" ][ DictionaryEJB.jar
DictionaryManagerBean DictionaryEJB.jar,META-INF/ejb-jar.xml jms/DictionaryTCF
javax.jms.TopicConnectionFactory jms/DictionaryTCF "" "" "" ][ DictionaryWeb ""
DictionaryWeb.war,WEB-INF/web.xml jdbc/DictionaryDB javax.sql.DataSource
jdbc/DictionaryDB "" "" "" ]] -MapWebModToVH [[ DictionaryWeb
DictionaryWeb.war,WEB-INF/web.xml default_host ]] -MapMessageDestinationRefToEJB
[[ DictionaryEJB.jar DictionaryManagerBean DictionaryEJB.jar,META-INF/ejb-jar.xml
jms/DictionaryT jms/DictionaryT ][ DictionaryEJB.jar DictionaryManagerBean
DictionaryEJB.jar,META-INF/ejb-jar.xml jms/DictionaryQ jms/DictionaryQ ]]
-BindJndiForEJBMessageBinding [[ DictionaryEJB.jar LookupLoggerBean
DictionaryEJB.jar,META-INF/ejb-jar.xml "" jms/DictionaryQS jms/DictionaryQ "" ][
DictionaryEJB.jar DefineLoggerBean DictionaryEJB.jar,META-INF/ejb-jar.xml ""
jms/DictionaryTS jms/DictionaryT "" ]]]' )
AdminConfig.save();
appManager = AdminControl.queryNames('type=ApplicationManager,process=server1,*);
AdminControl.invoke(appManager, 'startApplication','DictionaryApp');
```

The `-MapMessageDestinationRefToEJB` options allow you to map message destination references that are defined within EJBs to JMS destinations in the application server. You must provide the necessary parameters, which define the mapping, afterward in the list format. These installation options correspond with step 2 on page 163 in the administrative console installation. These parameters are necessary:

EJB Module	The name of the EJB module that contains the EJB that references the JMS destination
EJB	The name of the EJB, which references the JMS destination
URI	Module URI for the *`EJB Module`*
Message Destination Object	The name of the destination that is defined by the application
Target Resource JNDI Name	The JNDI name that is defined in the application server for the JMS destination

The `-BindJndiForEJBMessageBinding` options allow you to bind JMS activation specifications to MDBs. You must provide the necessary parameters, which define the mapping, afterward in the list format. These installation options correspond with step 1 on page 162 in the administrative console installation. These parameters are necessary:

EJB Module	The name of the EJB module that contains the MDB to be bound to the JMS activation specifications.
EJB	The name of the MDB.
URI	Module URI for the *`EJB Module`*.
Listener Port	Port on which the MDB listens. This field is left null, because DictionaryApp V3 uses activation specifications.
JNDI name	The JNDI name that is defined in the application server for the JMS activation specification.

Destination JNDI name The JNDI name that is defined in the application server for the JMS destination.

ActivationSpec Authentication Alias

Used to allow access the activation specifications without having to expose username and passwords. Left as null since DictionaryApp V3 does not use an authentication alias.

5.8 Accessing DictionaryApp V3

Accessing DictionaryApp V3 is similar to the previous versions of DictionaryApp. The URL to access DictionaryApp V3 is similar to this website:

```
http://localhost:9080/DictionaryApp/DictionaryServlet
```

DictionaryApp V3 looks similar to Figure 5-53.

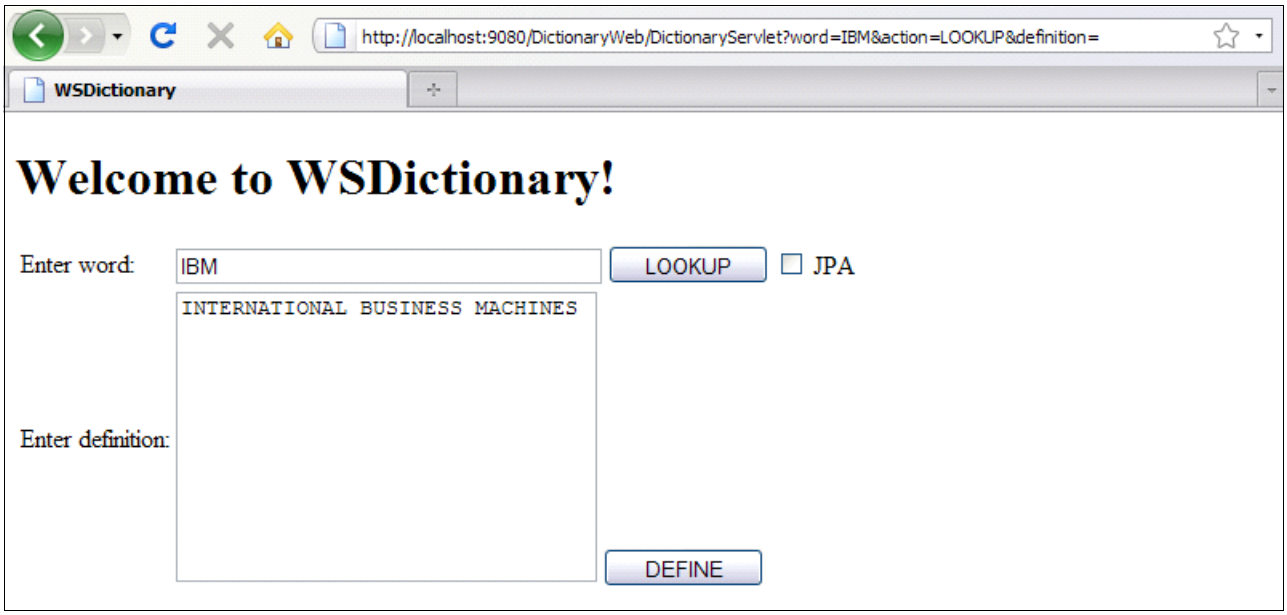


Figure 5-53 DictionaryApp V3

After looking up and defining a few words, you can check your `SystemOut.log` for the logs that are produced by the MDBs. The log has entries similar to Example 5-7.

Example 5-7 Accessing DictionaryApp V3 log file

```
[8/17/10 16:58:37:785 CDT] 0000002e SystemOutVO Tue Aug 17 16:58:37 CDT 2010
:LOOKUP(IBM : INTERNATIONAL BUSINESS MACHINES)
```

6

Incorporating RESTful web services into DictionaryApp V4

In this chapter, we discuss how to deploy applications as web services on WebSphere Application Server. You can implement web services in many ways, but it is beyond the scope of this document to cover all of the capabilities of web services. We explore how to implement the Representational State Transfer or RESTful architecture, which is one style of web services, in DictionaryApp V4. DictionaryApp V4 allows clients to access information in DictionaryDatabase and to obtain results in simple text format instead of HTML. Therefore, you can write applications on the client side that implement a server-independent user interface (UI) as the front end for the functionalities that are provided by DictionaryApp V4. We provide you with a client-side application for this purpose.

Sample material: See Appendix C, “Additional material” on page 235 for information about downloading the sample material that is used in this chapter.

© Copyright IBM Corp. 2010. All rights reserved.

6.1 Setting up Web 2.0 Feature Pack for WebSphere Application Server 7.0

To use the JAX-RS application programming interface (API) in WebSphere Application Server, you must obtain the necessary jar files for the IBM implementation of JAX-RS and configure the application server web container to manage RESTful resources. WebSphere Application Server for Developers V7 does not include these features. You can obtain them through installing the Web 2.0 Feature Pack for WebSphere Application Server 7.0.

6.2 Downloading Web 2.0 Feature Pack for WebSphere Application Server 7.0

Like WebSphere Application Server for Developers V7, the Web 2.0 Feature Pack is no charge for developers who want to incorporate service-oriented architecture into their web service applications. Use the following procedure to download the Web 2.0 Feature Pack:

1. Go to the following URL, as shown in Figure 6-1:

   http://www.ibm.com/software/webservers/appserv/was/featurepacks/web20/

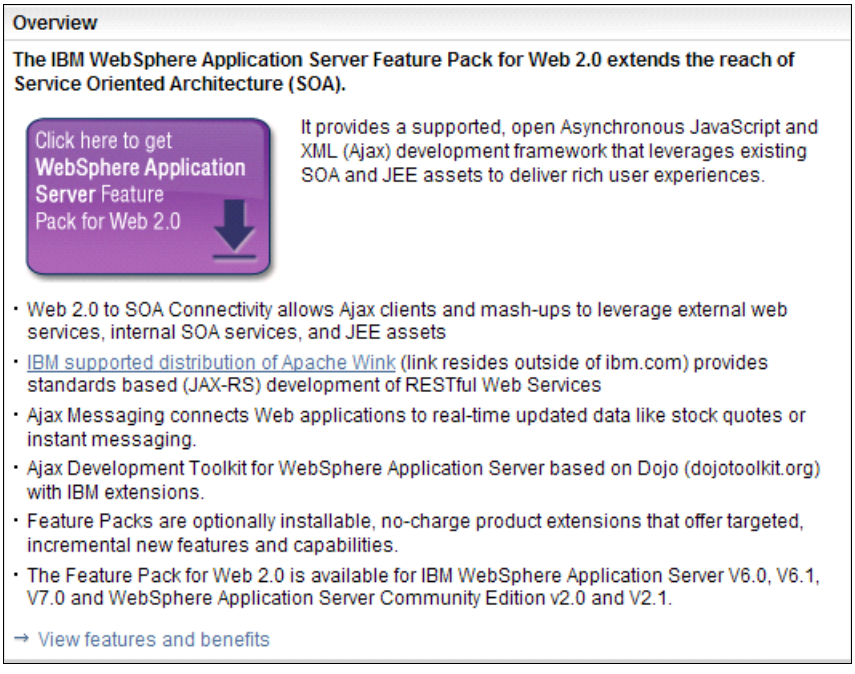


Figure 6-1 WebSphere Application Server Feature Pack download

2. Select **Click Here to get WebSphere Application Server Feature Pack for Web 2.0**.

3. Log in with your IBM account (you have already registered for an IBM account when you downloaded the WebSphere Application Server), as shown in Figure 6-2.

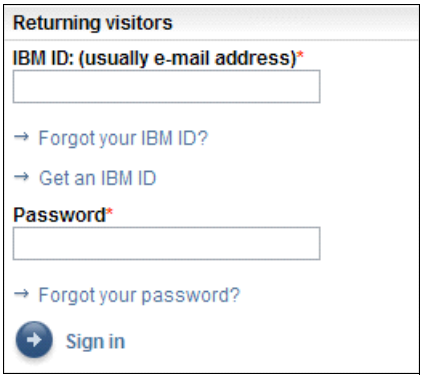


Figure 6-2 IBM sign in

4. Under Offering, select **WebSphere Application Server Version 6.0.2, 6.1 and V7.0.0**, as shown in Figure 6-3. Click **Continue**.

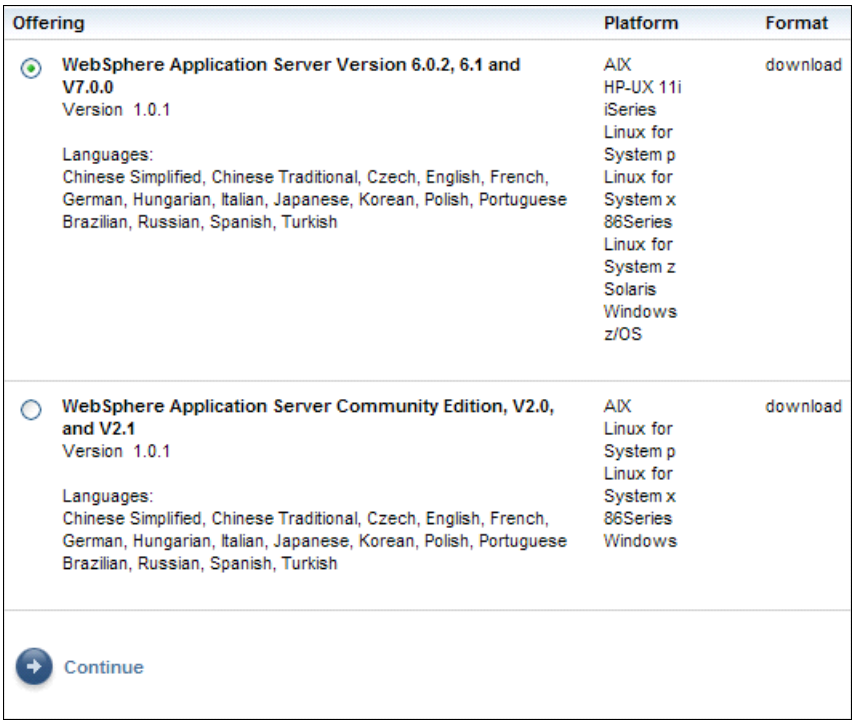


Figure 6-3 IBM WebSphere Application Server Feature Pack for Web 2.0 offerings

5. Complete the licensing agreement form on the next page as appropriate, as shown in Figure 6-4. Click **I confirm**.

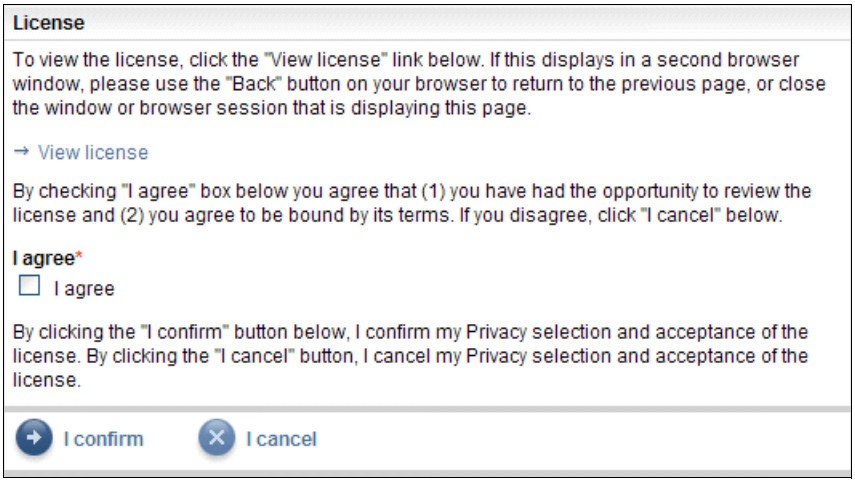


Figure 6-4 IBM WebSphere Application Server Feature Pack for Web 2.0 license

6. On the Download using Download Director window, select the `.zip` file version, as shown in Figure 6-5. A Download Director window opens, where the File attribute is the file path to which your selected package will be downloaded. Alternatively, you can select the HTTP, which allows you to download the `.zip` file directly through your browser.

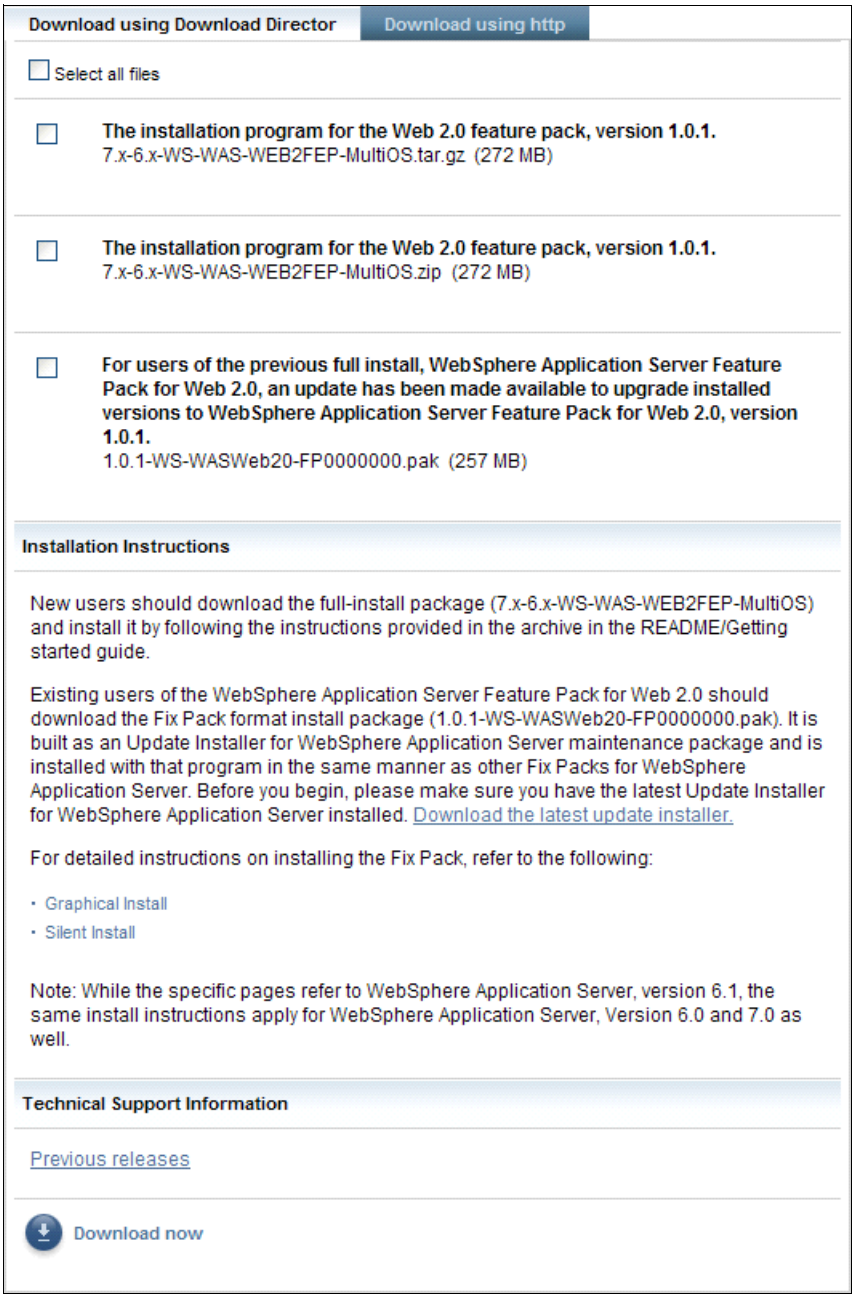


Figure 6-5 IBM WebSphere Application Server Feature Pack for Web 2.0 download

6.2.1 Installing Web 2.0 Feature Pack for WebSphere Application Server 7.0

After you have downloaded the Web 2.0 `.zip` archive, use the following procedure to run the setup wizard in the download package to install the feature pack:

1. Stop your server. If you used the same profile construction template that was used in the examples in this book, the server name is `server1`.
2. Extract the downloaded package.
3. Open a command prompt, and navigate to the folder where the extracted files were saved.
4. Use the following command to install the package:

   `install.exe -is:javahome` *`install root`*`\java`

   Where *`install root`* is the directory where WebSphere is installed.
5. An installation wizard displays. Follow the steps provided by the installation wizard to complete the installation.
6. Run *`profile root`*`/bin/versionInfo.bat`, and verify that the feature pack is installed on top of the application server, as shown in Example 6-1.

Example 6-1 Feature pack installation verification

```
Installed Product
--------------------------------------------------------------------------------

Name                        IBM WebSphere Application Server
Version                     7.0.0.0
ID                          BASE
Build Level                 r0835.03
Build Date                  8/31/08

Installed Product
--------------------------------------------------------------------------------

Name                        Web 2.0 Feature Pack
Version                     1.0.1.0
ID                          WEB2FEP
Build Level                 web21012.5
Build Date                  3/15/10

--------------------------------------------------------------------------------
```

6.3 Updating DictionaryApp V4 source files

For DictionaryApp V4, you expose the LOOKUP and DEFINE functionalities of DictionaryApp V4 as a simple RESTful web service. This task requires a RESTful resource, `DictionaryService.java`, and a REST servlet, `DictionaryApplication.java`. `DictionaryService.java` implements LOOKUP and DEFINE through calling the business methods in DictionaryJPAAdapter. `DictionaryApplication.java` serves as the service endpoint for all RESTful resources in DictionaryApp V4.

Figure 6-6 illustrates the overall architecture of DictionaryApp V4.

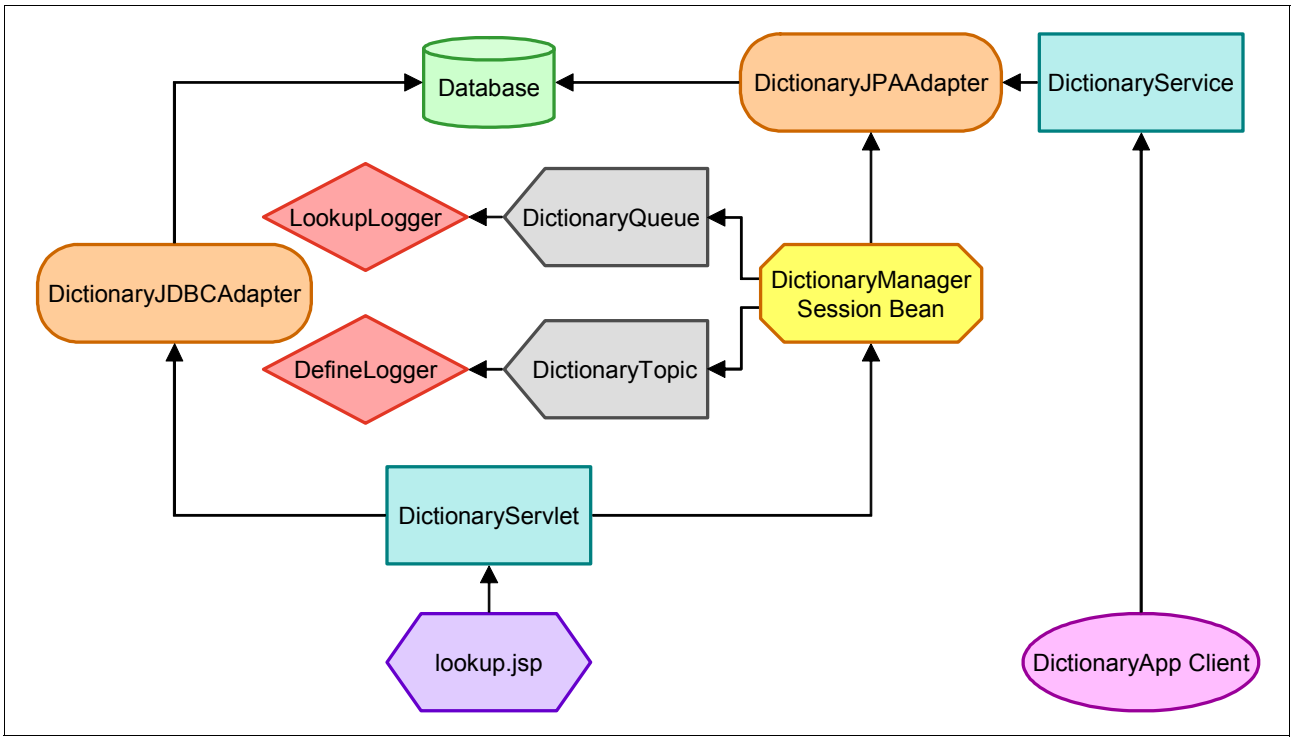


Figure 6-6 DictionaryApp V4

The following files are additions in DictionaryApp V4 to the existing source files in DictionaryApp V3:

- `DictionaryApplication.java`
- `DictionaryService.java`
- `web.xml`

6.3.1 DictionaryApplication.java

`DictionaryApplication.java` is the service endpoint for `DictionaryService.java`. Example 6-2 shows the text of `DictionaryApplication.java`.

Example 6-2 DictionaryApplication.java

```
package com.ibm.dictionaryapp.rest;

import java.util.HashSet;
import java.util.Set;

import javax.ws.rs.core.Application;

public class DictionaryApplication extends Application {
      public Set<Class<?>> getClasses() {
         Set<Class<?>> classes = new HashSet<Class<?>>();
         classes.add(DictionaryService.class);
         return classes;
      }
}
```

6.3.2 DictionaryService.java

`DictionaryService.java` is the resource representation of the LOOKUP and DEFINE functionalities as web services for DictionaryApp V4. Example 6-3 shows the text of `DictionaryService.java`.

Example 6-3 DictionaryService.java

```
package com.ibm.dictionaryapp.rest;

import javax.ws.rs.GET;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

import com.ibm.dictionaryapp.database.DictionaryJPAAdapter;
import com.ibm.dictionaryapp.database.Entry;

@Path("/Dictionary")
public class DictionaryService {
   private static DictionaryJPAAdapter jpaadapter = new DictionaryJPAAdapter();

      @GET
      @Produces(MediaType.TEXT_PLAIN)
      @Path("/lookup/{word}")
      public Response getDictionary(@PathParam("word") String word) {
         return Response.ok(jpaadapter.lookup(word).toString()).build();
      }

      @PUT
      @Produces(MediaType.TEXT_PLAIN)
      @Path("/define/{entry}")
      public Response define(@PathParam("entry") Entry entry) {
         jpaadapter.define(entry);
         return Response.ok(entry.getWord() + " has been defined.").build();
      }
}
```

6.3.3 web.xml

The updated `web.xml` file includes `DictionaryApplication.java` as a functional servlet. Example 6-4 shows the text of `web.xml`.

Example 6-4 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>DictionaryApp</display-name>
```

```
  <servlet>
    <description></description>
    <display-name>DictionaryServlet</display-name>
    <servlet-name>DictionaryServlet</servlet-name>
    <servlet-class>com.ibm.dictionaryapp.servlet.DictionaryServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>DictionaryRestService</servlet-name>
    <servlet-class>com.ibm.websphere.jaxrs.server.IBMRestServlet</servlet-class>
    <init-param>
      <param-name>javax.ws.rs.Application</param-name>
      <param-value>com.ibm.dictionaryapp.rest.DictionaryApplication</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>DictionaryServlet</servlet-name>
    <url-pattern>/DictionaryServlet</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>DictionaryRestService</servlet-name>
    <url-pattern>/DictionaryService/*</url-pattern>
  </servlet-mapping>
</web-app>
```

All of the necessary JARs for the IBM implementation of JAX-RS are in the directory *install root*\web2fep\optionalLibraries\jaxrs. Be sure to include them in the DictionaryWeb.war\WEB INF\lib folder in your EAR file when you prepare the application, so that WebSphere Application Server can reference them during deployment.

6.4 Redeploying DictionaryApp V4

Deploying DictionaryApp V4 is similar to previous versions of DictionaryApp.

6.5 Accessing DictionaryApp V4

Use the following URL to access DictionaryApp V4:

```
http://localhost:9080/DictionaryWeb/DictionaryService/Dictionary/lookup/IBM
```

This URL displays text similar to Figure 6-7 in your browser.

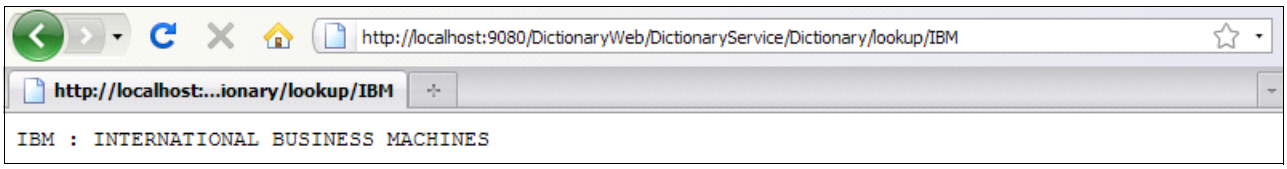


Figure 6-7 DictionaryApp V4

With only a web browser, you can only access the LOOKUP functionality of DictionaryApp V4, because it is the only method available with the GET request. Because the Entry object is serialized through the toString method, you only receive the String representation of the

dictionary entry in the HTTP response to your request. Your browser typically handles this situation by displaying the text as it is. In order to access DEFINE, you have to access DictionaryApp V4 through a client that can send both GET and PUT requests.

6.6 Creating a web service client for DictionaryApp V4

In section 6.3, “Updating DictionaryApp V4 source files” on page 172, we demonstrated how to access one of the methods in the `DictionaryService.java` resource, but you can enhance the functionality and UI by accessing the data through an actual web service client for DictionaryApp V4. Example 6-5 shows a listing of a simple client that accesses both functions of DictionaryApp V4.

Writing your own client: Because DictionaryApp V4 conforms to the RESTful architecture, it is possible to access DictionaryApp V4 through a client that uses HTTP. You might want to write your own client to get an understanding of how web services run on WebSphere Application Server (possibly even in a non-Java-based language).

Example 6-5 DictionaryApp V4 web service client

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

import javax.swing.JOptionPane;

import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.client.utils.URIUtils;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.wink.common.internal.uri.UriEncoder;

public class DictionaryClient {
   public static void main(String[] a) throws Exception {
      HttpClient client = new DefaultHttpClient();
      boolean quit = false;
      while (!quit) {
         String function = JOptionPane.showInputDialog(null,
               "FUNCTION? (LOOKUP/DEFINE/QUIT)");
         if (function.toUpperCase().equals("DEFINE")) {
            String word = UriEncoder.encodeQuery(JOptionPane
                  .showInputDialog(null, "WORD"), true);
            String definition = UriEncoder.encodeQuery(JOptionPane
                  .showInputDialog(null, "DEFINITION"), true);
            HttpPut request = new HttpPut(URIUtils.createURI("http",
                  "localhost", 9080,
                  "/DictionaryWeb/DictionaryService/Dictionary/define/"
                        + word + "%20:%20" + definition, null, null));
            JOptionPane.showMessageDialog(null, client.execute(request)
                  .getStatusLine());
         } else if (function.toUpperCase().equals("LOOKUP")) {
            String word = UriEncoder.encodeQuery(JOptionPane
                  .showInputDialog(null, "WORD"), true);
            HttpGet request = new HttpGet(URIUtils.createURI("http",
```

```
                "localhost", 9080,
                "/DictionaryWeb/DictionaryService/Dictionary/lookup/"
                      + word, null, null));
            String entrystring = new BufferedReader(new InputStreamReader(
                  client.execute(request).getEntity().getContent()))
                  .readLine();
            JOptionPane.showMessageDialog(null, entrystring);
         } else if (function.toUpperCase().equals("QUIT")) {
            quit = true;
         } else {
            JOptionPane.showMessageDialog(null, "UNKNOWN FUNCTION");
         }
      }
   }
}
```

DictionaryClient references several Apache HTTP libraries. These JAR files were included in the Web 2.0 Feature Pack. You can access these JAR files in the following directory:

install root\web2fep\optionalLibraries\jaxrs

You can locate the DictionaryClient in the Web Services Client directory that is included in the sample applications. To run the client, include the directory of DictionaryClient and all the jar files in the `jaxrs` directory in the class path, and run DictionaryClient as the major program. Example 6-6 shows this action if we are currently in the Web Services Client directory.

Example 6-6 Running the DictionaryClient

```
set WAS_HOME=<install root>
    set JAXRS=%WAS_HOME%\web2fep\optionalLibraries\jaxrs
    set
CP=.;%JAXRS%\commons-codec-1.3.jar;%JAXRS%\commons-lang-2.3.jar;%JAXRS%\httpclient
-4.0.jar;%JAXRS%\httpcore-4.0.jar;%JAXRS%\ibm-wink-jaxrs-1.0.0.jar;%JAXRS%\jcl-ove
r-slf4j-1.5.8.jar;%JAXRS%\jsr311-api-1.0.jar;%JAXRS%\slf4j-api-1.5.8.jar;%JAXRS%\s
lf4j-jdk14-1.5.8.jar
   %WAS_HOME%\java\bin\java.exe -cp %CP% DictionaryClient
```

Figure 6-8 through Figure 6-10 on page 178 show what the DictionaryApp V4 web services client looks like when run on your local Java runtime environment (JRE).

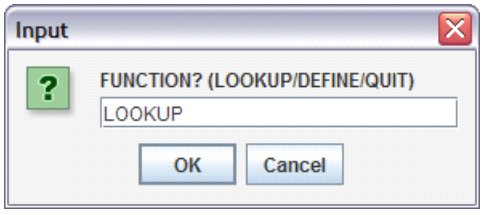


Figure 6-8 DictionaryApp V4 Web Services Client

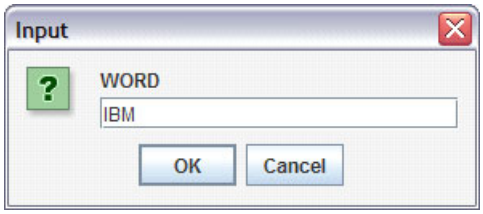


Figure 6-9 DictionaryApp V4 Web Services Client

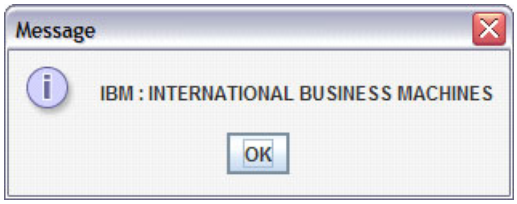


Figure 6-10 DictionaryApp V4 Web Services Client

7

Including WebSphere-specific bindings files for DictionaryApp V5

During application deployment, WebSphere generates two types of files: `deployment.xml` and the WebSphere-specific bindings files, which are based on the deployment options specified by the user. To shorten and simplify the deployment process, WebSphere allows developers to deploy applications with the WebSphere-specific bindings files already included in them. During deployment, the application server can extract the specifications from the bindings files to apply them automatically. This approach eliminates the need for users to specify application server-specific bindings information during deployment through both the administrative console and **`wsadmin`**.

Sample material: See Appendix C, “Additional material” on page 235 for information about downloading the sample material that is used in this chapter.

© Copyright IBM Corp. 2010. All rights reserved.

7.1 DictionaryApp V5 bindings files

We work with two major bindings files in this document. The `ibm-web-bnd.xml` file specifies the deployment bindings and options for web modules, such as context roots and security. The `ibm-ejb-jar-bnd.xml` file specifies the deployment bindings and options for Enterprise JavaBeans (EJB) modules, such as Java Naming and Directory Interface (JNDI) names and message bean bindings. You can look in previously installed versions of DictionaryApp in the configuration repository to see how WebSphere stored the previously specified deployment options in the bindings files. We provide you with the necessary bindings files for DictionaryApp V5.

If you want to generate your own bindings files, see Appendix A, “Development tools reference” on page 219 for how to import the XML schema for the WebSphere-specific bindings files into Eclipse to assist with development.

Note: The WebSphere bindings files provided here are only compatible with Java Enterprise Edition V1.5 and later. The schema is located in the *`install_root`*`/properties/schemas` directory. Look for `ibm-web-bnd_1_0.xsd` and `ibm-ejb-jar-bnd_1_0.xsd`.

7.2 ibm-web-bnd.xml for DictionaryWeb

The `ibm-web-bnd.xml` file is the bindings file for DictionaryWeb. Example 7-1 shows the text of the `ibm-web-bnd.xml` file.

Example 7-1 ibm-web-bnd.xml for DictionaryWeb

```
<?xml version="1.0" encoding="UTF-8"?>
<web-bnd xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://websphere.ibm.com/xml/ns/javaee"
    xsi:schemaLocation="http://websphere.ibm.com/xml/ns/javaee
http://websphere.ibm.com/xml/ns/javaee/ibm-web-bnd_1_0.xsd" version="1.0">
  <virtual-host name="default_host"/>
  <resource-ref name="jdbc/DictionaryDB" binding-name="DictionaryDB"/>
</web-bnd>
```

This file is included in the DictionaryApp V5 EAR file in the following directory:

`DictionaryWeb.war\WEB-INF`

The `<resource-ref>` tag contains the information that is required by the application server to perform the resource mapping between the DataSource object in DictionaryApp V5 and DictionaryDatasource. The `<virtual-host>` tag contains the information that is required by the application server to map `default_host` as the virtual host for DictionaryApp V5.

7.3 ibm-web-bnd.xml for DictionaryEJB

The `ibm-web-bnd.xml` file is the bindings file for DictionaryEJB. Example 7-2 shows the text of `ibm-web-bnd.xml`.

Example 7-2 ibm-web-bnd.xml for DictionaryEJB

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar-bnd xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://websphere.ibm.com/xml/ns/javaee"
    xsi:schemaLocation="http://websphere.ibm.com/xml/ns/javaee
http://websphere.ibm.com/xml/ns/javaee/ibm-ejb-jar-bnd_1_0.xsd" version="1.0">
  <session name="DictionaryManagerBean">
    <resource-ref name="jms/DictionaryTCF" binding-name="jms/DictionaryTCF"/>
    <resource-ref name="jms/DictionaryQCF" binding-name="jms/DictionaryQCF"/>
    <message-destination-ref binding-name="jms/DictionaryQ"
name="jms/DictionaryQ"/>
    <message-destination-ref binding-name="jms/DictionaryT"
name="jms/DictionaryT"/>
  </session>
  <message-driven name="StatisticsBean">
    <jca-adapter activation-spec-binding-name="jms/DictionaryTS"
destination-binding-name="jms/DictionaryT"/>
  </message-driven>
  <message-driven name="LoggerBean">
    <jca-adapter activation-spec-binding-name="jms/DictionaryQS"
destination-binding-name="jms/DictionaryQ"/>
  </message-driven>
</ejb-jar-bnd>
```

This file is included in the DictionaryApp V5 EAR file in the following directory:

`DictionaryEJB.jar\META-INF`

The `<resource-ref>` tag contains the information that is required by the application server to perform the resource mapping between the connection factory and destination resources in DictionaryManagerBean.java. The `<message-driven>` tag contains the information that is required by the application server to bind each message-driven bean (MDB) to its activation specifications.

7.4 Redeploying DictionaryApp V5

After including these files in DictionaryApp V5, the deployment process is simplified in both the administrative console and **`wsadmin`**.

7.4.1 Administrative console

In the administrative console, you see that the only options that are required by the application server are the options for mapping application targets and class loaders. Figure 7-1 shows the number of steps that must be taken to install DictionaryApp V5.

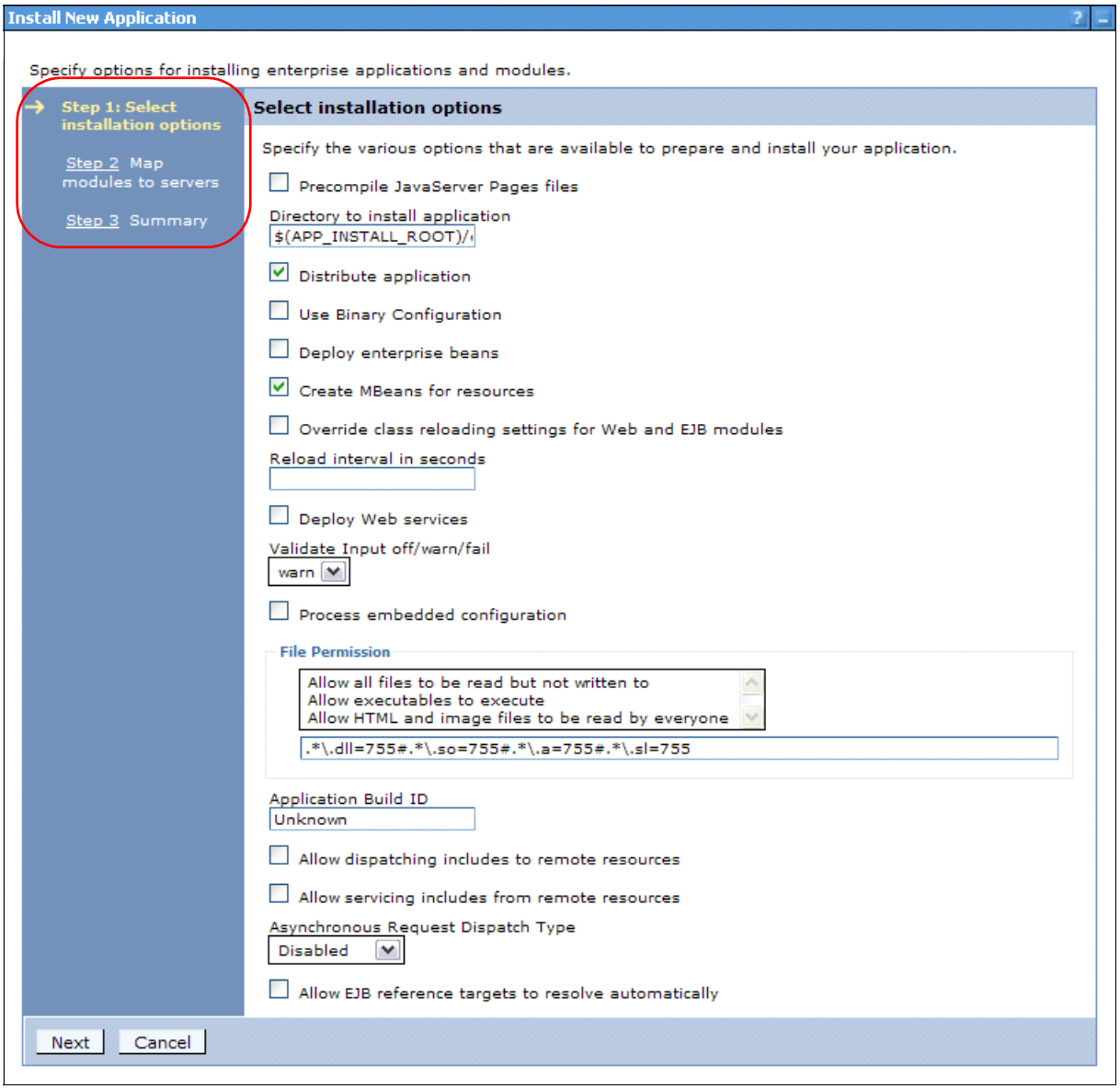


Figure 7-1 Administrative console: Install New Application fast path

7.4.2 wsadmin scripting

In **wsadmin**, you do not need to provide the application bindings options for DictionaryApp V5. To install an application with the bindings files already included, use the following command:

```
AdminApp.install('application file path', '[-appname application name]')
```

The install script for DictionaryApp V5 looks similar to Example 7-3.

Example 7-3 wsadmin install script

```
AdminApp.install('C:/DictionaryApp/version
5/DictionaryApp.ear', '[-appname DictionaryApp]');
AdminConfig.save();
appManager =AdminControl.queryNames('type=ApplicationManager,process=server1,*);
AdminControl.invoke(appManager,'startApplication','DictionaryApp');
```

8

Debugging

Debugging is integral to application design and development. Depending on the nature of your problem and the scale of your application, WebSphere offers many tools to help you to identify and fix errors in your application.

For specific application problems, WebSphere allows you to attach a remote debugger to the Java virtual machine (JVM) running in your server to set breakpoints and to closely examine a specific application thread and the memory components of that thread. Also, WebSphere can output other information concerning the JVM to help developers debug core problems and enhance the performance in their applications.

The application server can produce three types of output at the user's command:

- Verbose garbage collection
- Java core dump
- Java heap dump

© Copyright IBM Corp. 2010. All rights reserved.

8.1 Enabling WebSphere debug mode

Remote debugging in Java is the process of attaching a debugger from another environment onto a running JVM. This mode of debugging allows you to set up breakpoints and debug applications step-by-step on a remote JVM. WebSphere Application Server debug mode allows you to attach any debugger of your choice to the JVM that is running in your current process.

Use the following procedure to enable WebSphere Application Server debug mode:

1. From the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** in the navigation tree.
2. Select **server1** from the list of Application servers, as shown in Figure 3.

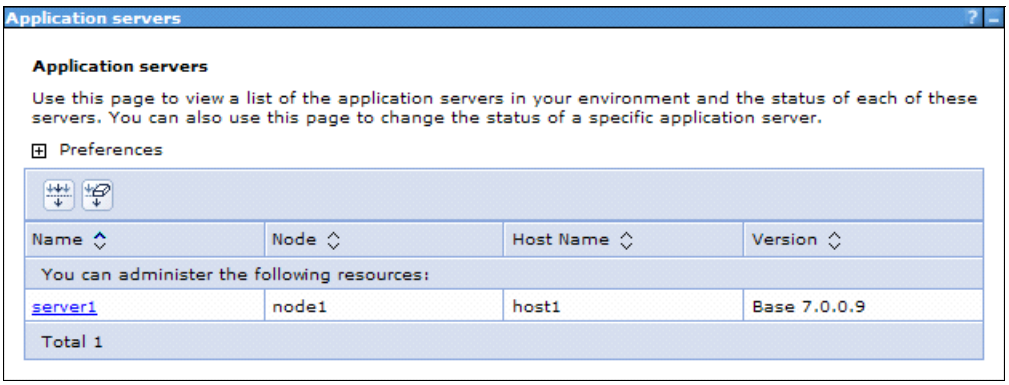


Figure 8-1 Administrative console: Application servers

3. In the Additional Properties section on the lower-right side of the page, select **Debugging service**, as shown in Figure 8-2.

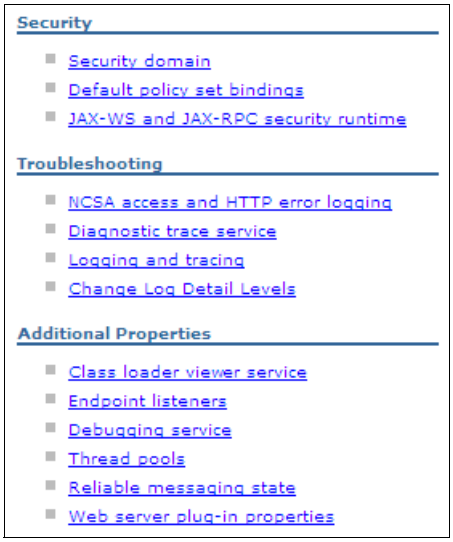


Figure 8-2 Administrative console: Debugging service

4. Configure your process for debugging purposes. To enable debug mode on server startup, select **Enable service at server startup**, as shown in Figure 8-3.

   You can also set up class filters to choose what classes the debugging process ignores. Selecting a package and clicking **Add** signal the application server to not stop in classes of that package in step-by-step debugging. Click **OK**.

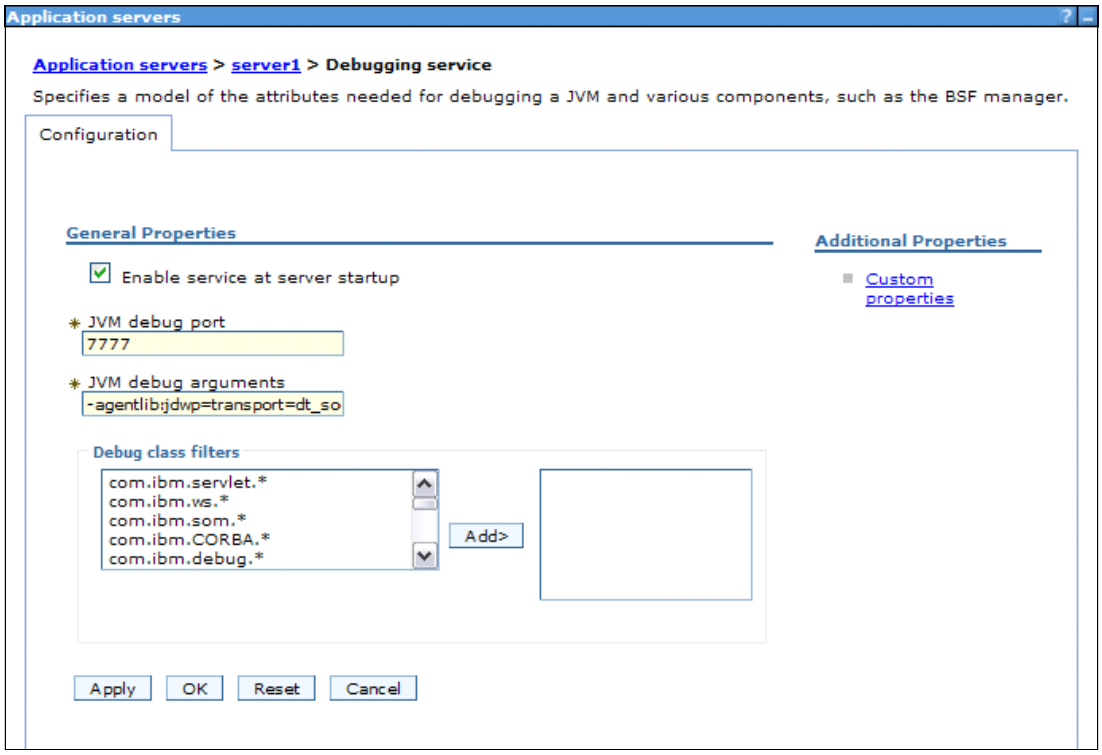


Figure 8-3 Administrative console: Application servers: Debugging service

5. Save and restart the server to enable the configuration changes that you have made.

Afterward, you can attach a debugger to your process JVM through specifying your server debug port (port 7777, if you use the default profile that is configured during installation). We document a tutorial for attaching the Eclipse debugger to the application server in Appendix A, “Development tools reference” on page 219.

8.2 Enabling verbose garbage collection

Verbose garbage collection automatically logs information concerning the JVM heap during every cycle of garbage collection. In most cases, enabling verbose garbage collection has a negligible impact on your server performance. Therefore, we suggest that you always enable this option, because it allows you to analyze every cycle of garbage collection to optimize performance and detect memory leaks.

Use the following procedure to enable verbose garbage collection:

1. From the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** in the navigation tree.
2. Select **server1** from the list of Application servers, as shown in Figure 8-4.

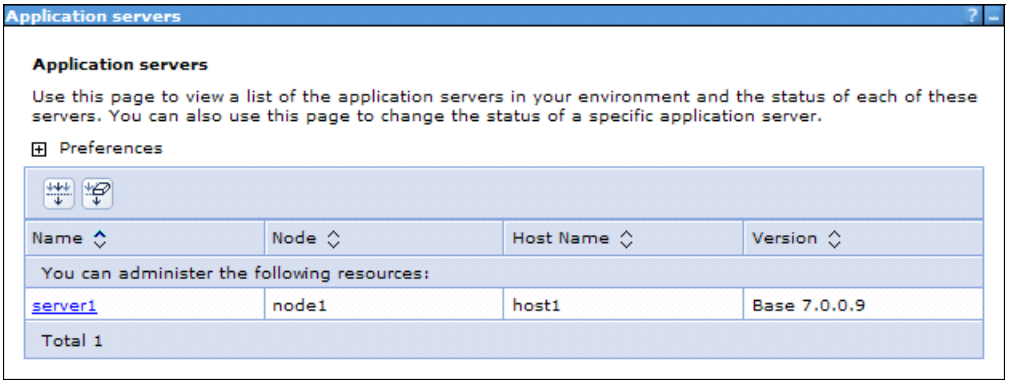


Figure 8-4 Administrative console: Application servers

3. In the Server Infrastructure section, click **Java and Process Management** to expand its contents, as shown in Figure 8-5. Select **Process definition**.

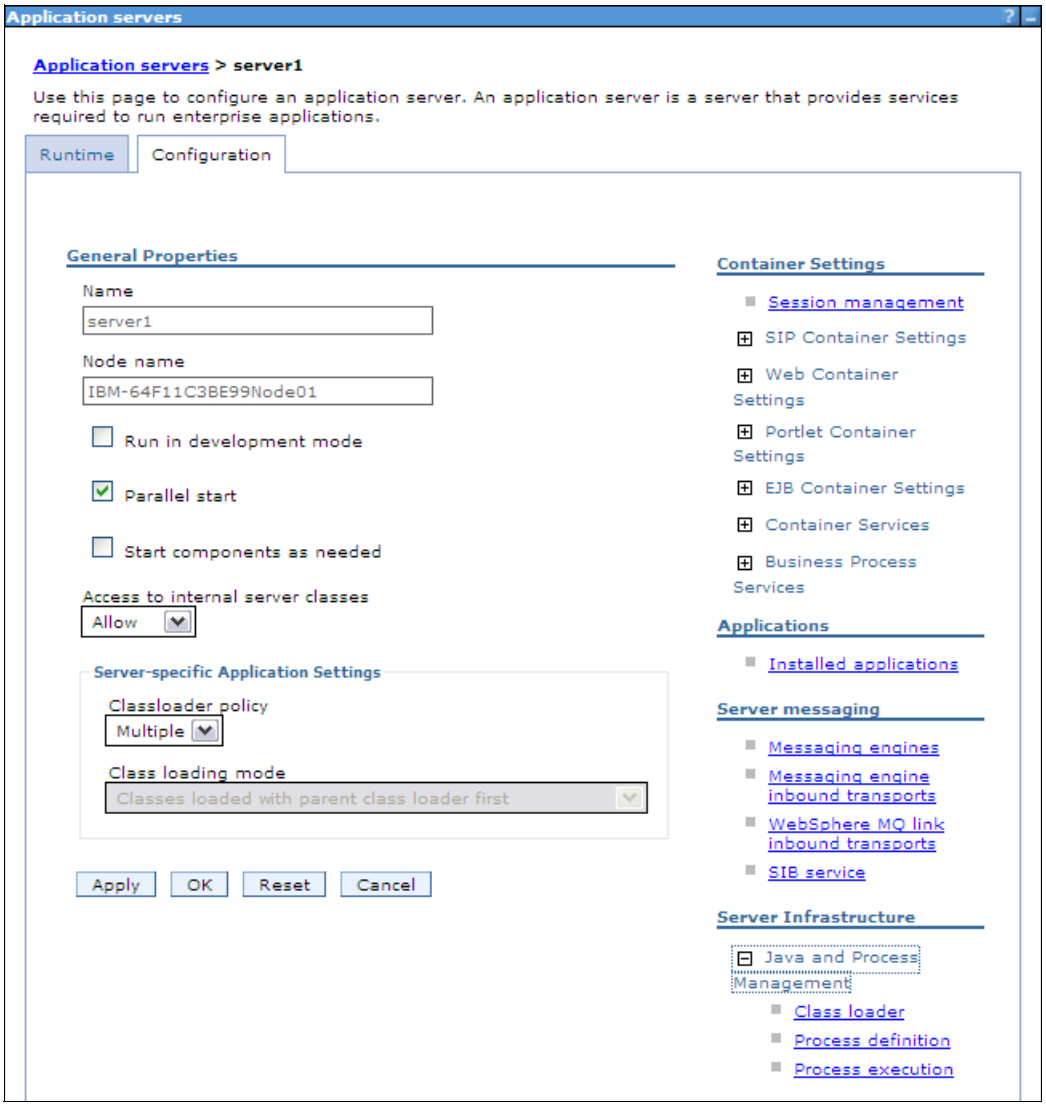


Figure 8-5 Administrative console: Server Infrastructure

4. In Additional Properties, select **Java Virtual Machine**, as shown in Figure 8-6.

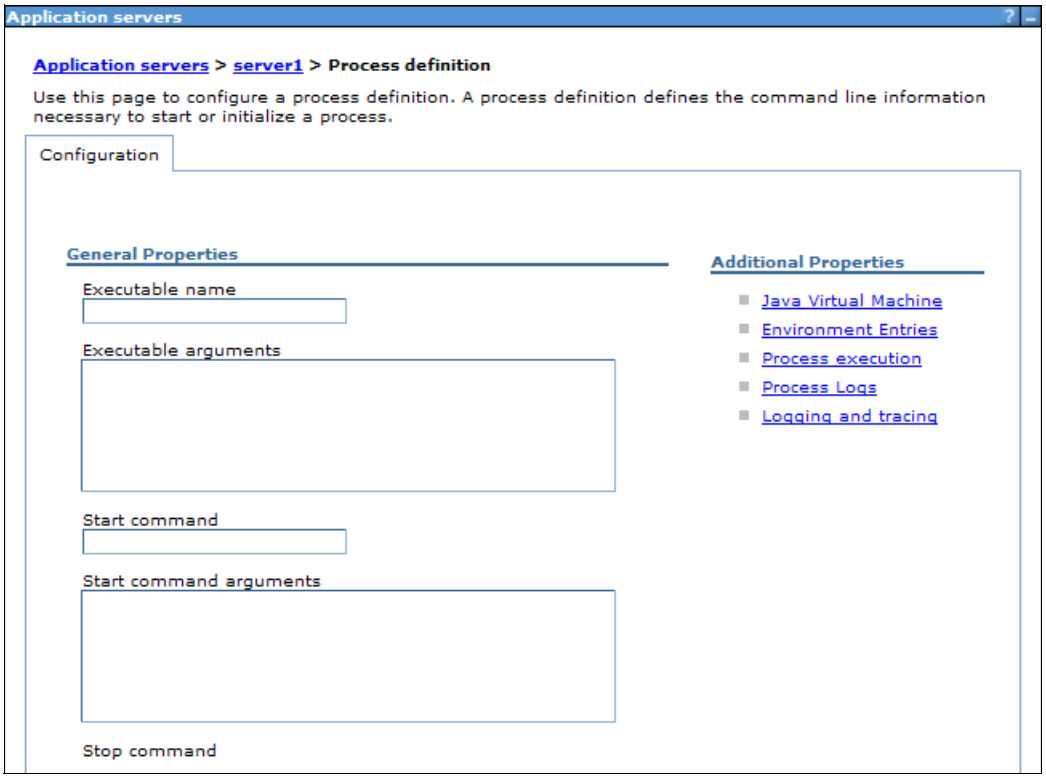


Figure 8-6 Administrative console: Process Definition

5. Configure the JVM process that is associated with your server. To enable verbose garbage collection, select **Verbose garbage collection**, as shown in Figure 8-7.

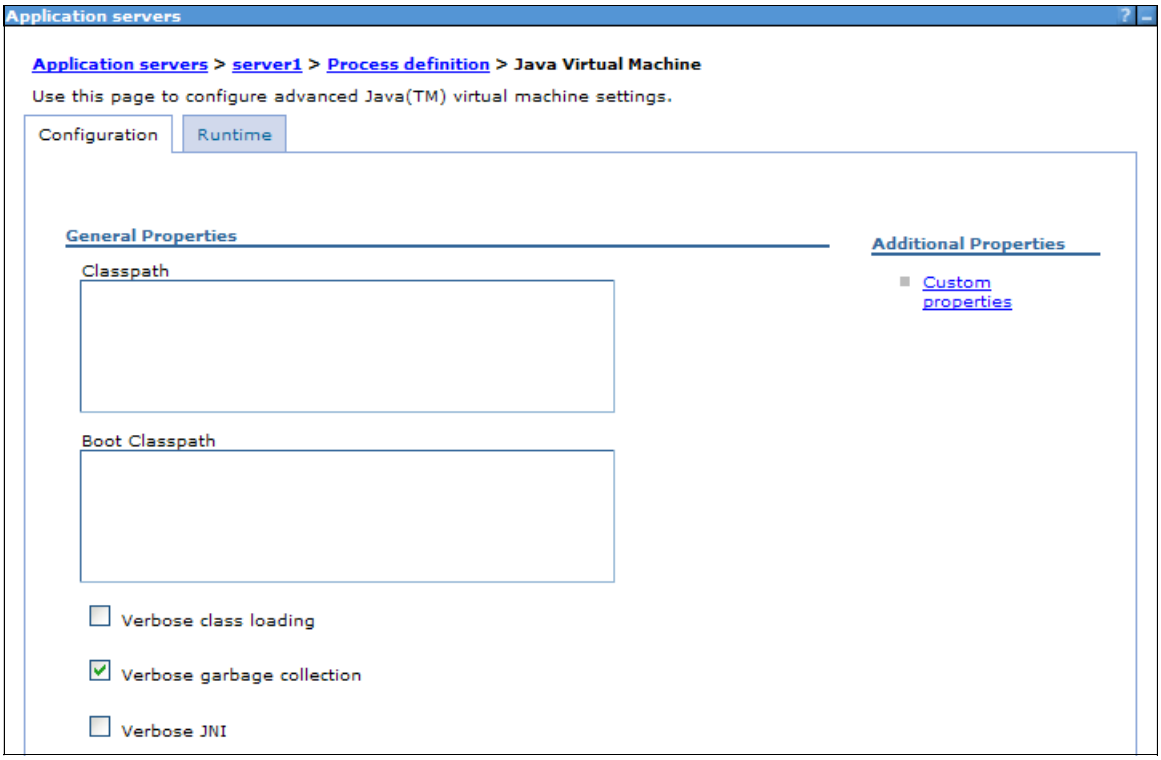


Figure 8-7 Administrative console: Java Virtual Machine

6. Save and restart the server. Afterward, check the `native_stderr.log` file for new entries detailing information concerning each garbage collection cycle.

You can obtain the documentation for verbose garbage collection data output in the `native_stderr.log` file at this website:

`http://www.ibm.com/developerworks/java/jdk/diagnosis/index.html`

For large applications and extremely long run cycles, it might not be practical to analyze the whole log in its raw form. IBM provides tools to help you analyze the data and produce meaningful and easily interpreted output. We describe these tools in 8.5, “IBM Support Assistant” on page 197.

8.3 Generating a Java heap dump

A *Java heap dump* is an IBM software development kit (SDK)-generated data file containing a snapshot of the current memory state of the application server JVM. A Java heap dump is generated automatically when the JVM runs out of memory. The user can request a Java heap dump through one of two ways:

- wsadmin scripting
- Process signaling

8.3.1 wsadmin scripting

To issue a command to generate a heap dump in **wsadmin**, you must first obtain a reference to the message bean (MBean) that is associated with the JVM running in your process using the command that is shown in Example 8-1.

Example 8-1 wsadmin generating a Java heap dump

```
jvm = AdminControl.queryNames("WebSphere:type=JVM,process=server1,node=devNode,*")
AdminControl.invoke(jvm, 'generateHeapDump')
```

Afterward, use the following command to induce the Java heap dump:

```
AdminControl.invoke(jvm, 'generateHeapDump');
```

8.3.2 Process signaling

Use the following procedure to induce Java heap dump through process signaling:

1. From the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** in the navigation tree.
2. Select **server1** from the list of Application servers, as shown in Figure 8-8.

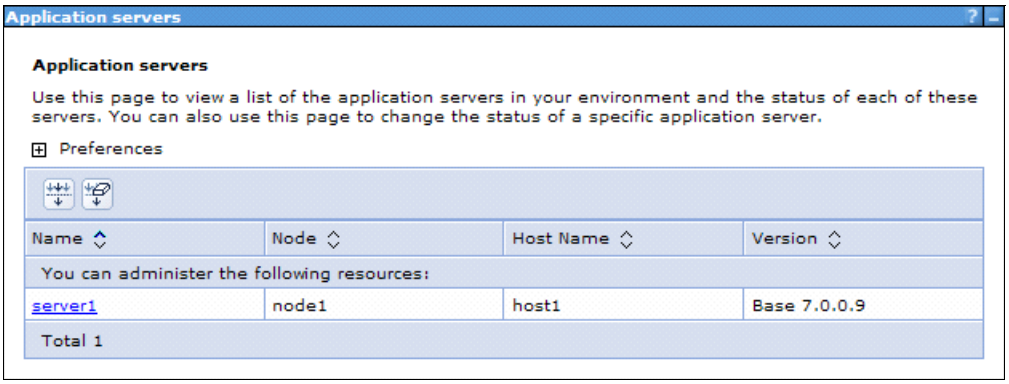


Figure 8-8 Administrative console: Application servers

3. In the Server Infrastructure section, click **Java and Process Management** to expand its contents, as shown in Figure 8-9. Select **Process definition**.

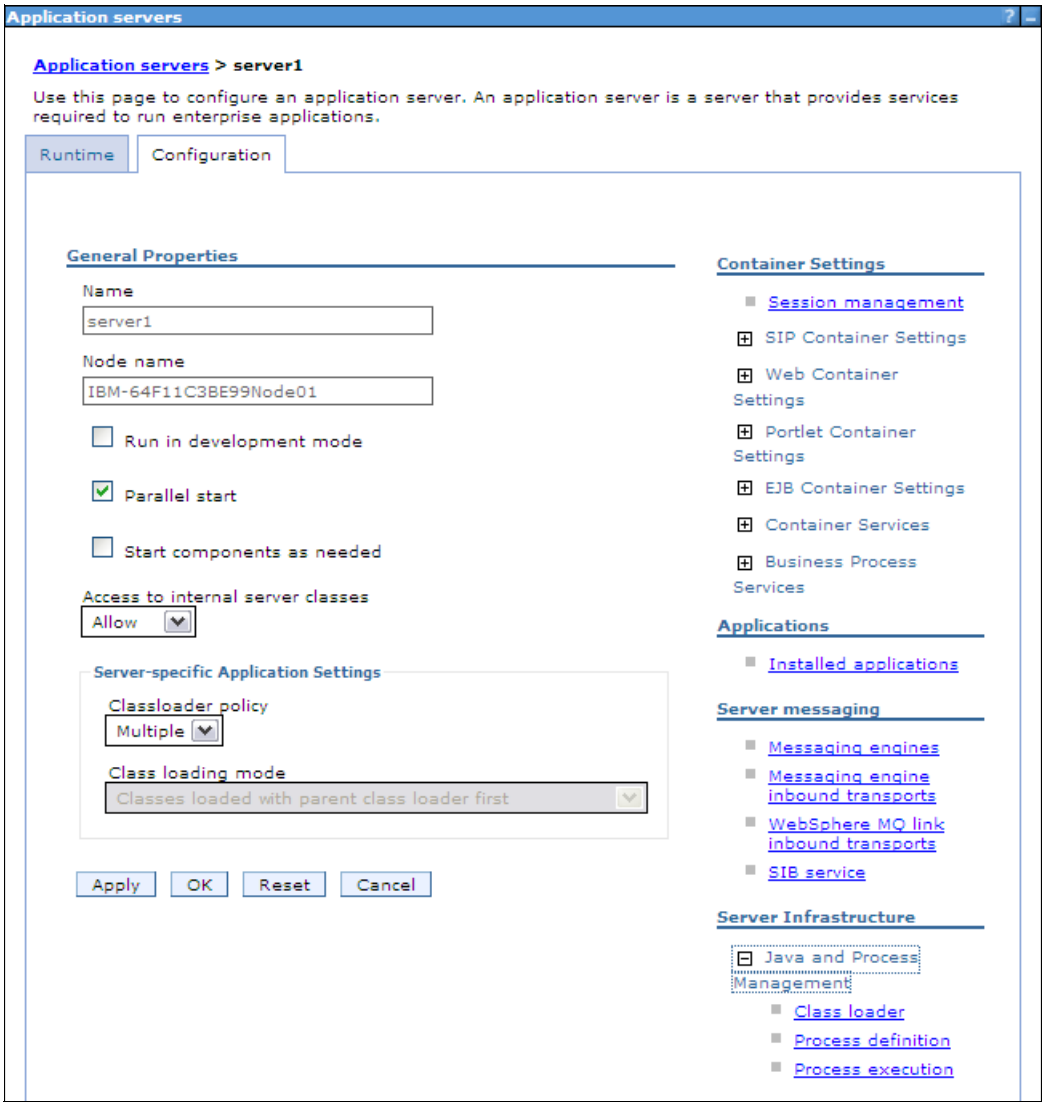


Figure 8-9 Administrative console: Server Infrastructure

4. In Additional Properties, select **Java Virtual Machine**, as shown in Figure 8-10.

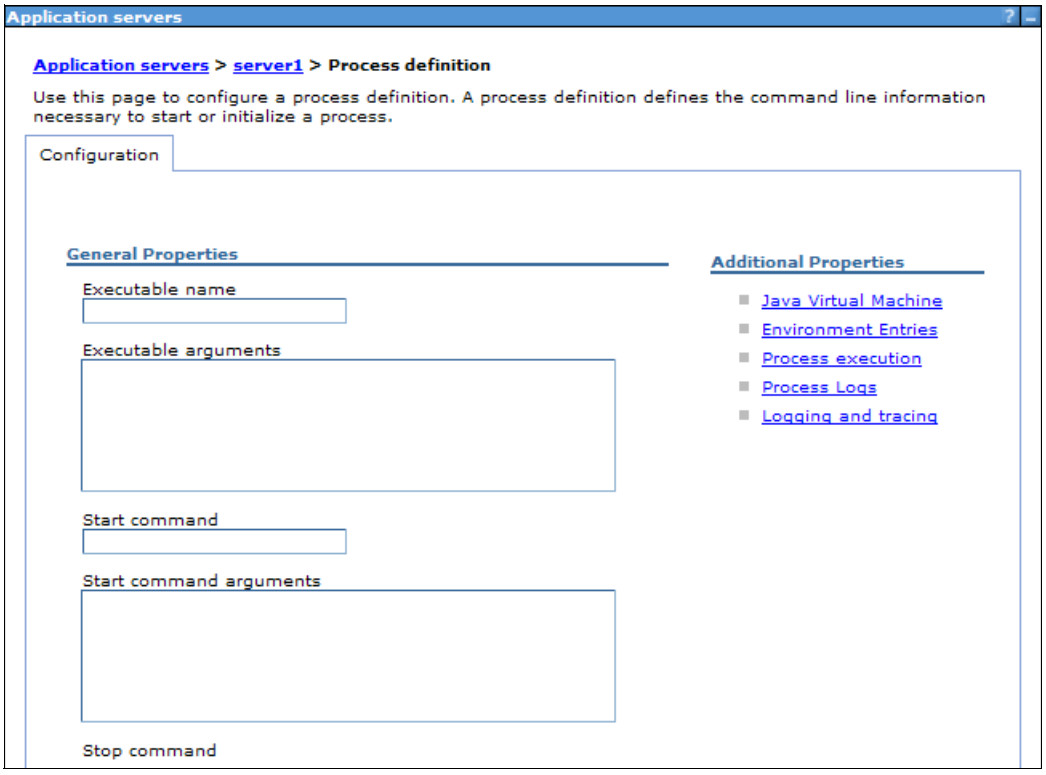


Figure 8-10 Administrative console: Process definition

5. To enable heap dump generation, you must specify JVM arguments during server startup. All IBM Java SDKs can generate heap dumps. To enable them by user request, set the Generic JVM arguments field to the string that is shown in Figure 8-11. Click **OK**.

Figure 8-11 Administrative console: Generic JVM arguments

6. Because you have just made a configuration change, changes do not take effect until the configuration repository is read during server startup, so you must save and stop the server.
7. Set the Microsoft Windows environment variable *IBM_HEAPDUMP* to any value, which signals the application server to generate heap dumps on user command.

   You can set Microsoft Windows environment variables in the command prompt using the following command:

   `set environment variable=value`

Use this command to set *IBM_HEAPDUMP*:

```
set IBM_HEAPDUMP=TRUE
```

This change only lasts through the duration of your command-line session. Alternatively, you can make this change permanent by right-clicking **My Computer** → **Properties** → **Advanced** → **Environment Variables** and creating *IBM_HEAPDUMP* as an environment variable.

8. Generate a start server script for your process using the following command:

   install root\profiles*profile name*\bin\startServer.bat -script *server name*

   This command generates a batch file that starts the server. When running this script, it allows you to maintain control of the process after the server has started. Thus, it allows you to send signals easily to the process while it is running.

9. Start the server again using the generated script at a command prompt:

   install root\profiles*profile name*\bin\start_*server name*.bat

   The generated script must have the string `start_` concatenated with the *server name* that is specified in the previous command. If you use the default profile that was configured during installation, the command looks like this example:

   C:\WAS\profiles*profile name*\bin\start_server1.bat

10. After the server has started, send a break signal to the server process. In Microsoft Windows, you press Break in the window, which references the process. (For certain computers, you press Fn+Pause). Afterward, the command prompt outputs a confirmation of the request, which is similar to Example 8-2.

Example 8-2 wsadmin JVMDUMP

```
JVMDUMP032I JVM requested Heap dump using
'C:\WAS\profiles\AppSrv01\bin\heapdump
.20100802.103715.1576.0003.phd' in response to an
event
JVMDUMP010I Heap dump written to
C:\WAS\profiles\AppSrv01\bin\heapdump.20100802.
103715.1576.0003.phd
```

The core dump is generated in the *install root*\profiles*profile name* directory.

This file is not readily readable. Do not analyze this file in its raw form. As with verbose garbage collection output, IBM provides tools to help you analyze heap dumps. We describe these tools in 8.5, “IBM Support Assistant” on page 197.

8.4 Generating Java core dump

The Java core dump is a IBM SDK-generated data file, which contains information pertaining to the threads and monitors in the JVM. Just as the Java heap dump is a snapshot of the process JVM memory, the Java core dump is a snapshot of the threads running on the JVM. This data file is generated automatically every time that a server crashes. Or, a user can issue a command to generate this data file.

A user can request the generation of a Java core dump in one of the following ways:

- wsadmin scripting
- Process signaling

8.4.1 wsadmin scripting

To issue a command to generate a core dump in **wsadmin**, you must first obtain a reference to the MBean that is associated with the JVM running in your process using the command shown in Example 8-3.

Example 8-3 wsadmin generating a Java core dump

```
jvm = AdminControl.queryNames("WebSphere:type=JVM,process=server1,node=devNode,*")
AdminControl.invoke(jvm, 'dumpThreads')
```

Afterward, use the following command to start the Java core dump:

```
AdminControl.invoke(jvm, 'dumpThreads');
```

8.4.2 Process signaling

Also, you can configure WebSphere to generate a Java core dump whenever the operating system (OS) sends a break signal to the server process. This process is the same procedure as generating the Java heap dump through signaling, except that the application server generates a Java core dump, by default, without configuration.

Use the following procedure to generate a Java core dump through process signaling:

1. Generate a start server script for your process using the following command:

   install root\profiles*profile name*\bin\startServer.bat -script *server name*

   This command generates a batch file that starts the server. When running this script, it allows you to maintain control of the process after the server has started. Thus, this command allows you to send process signals to the process easily.

2. Start the server again using the generated script in a command prompt:

   install root\profiles*profile name*\bin\start_*server name*.bat

   The generated script must have the string `start_` concatenated with the *server name* that is specified in the previous command. If you use the default profile that was configured during installation, the command looks like this example:

   ```
   C:\WAS\profiles\profile name\bin\start_server1.bat
   ```

3. After the server has started, send a break signal to the server process. In Microsoft Windows, you press Break in the window that references the process. (For certain computers, you press Fn+Pause). Afterward, the command prompt outputs a confirmation of the request that is similar to Example 8-4.

   Example 8-4 wsadmin JVMDUMP

   ```
   JVMDUMP032I JVM requested Java dump using
   'C:\WAS\profiles\AppSrv01\bin\javacore
   .20100804.164922.2440.0002.txt' in response to an
   event
   JVMDUMP010I Java dump written to
   C:\WAS\profiles\AppSrv01\bin\javacore.20100804.
   164922.2440.0002.txt
   ```

   The heap dump is generated in the *install root*\profiles*profile name* directory.

Unlike the Java heap dump, this file is in text format, and you can analyze it in its raw form. You can obtain documentation for the Java core dump at the following website:

http://www.ibm.com/developerworks/java/jdk/diagnosis/index.html

For large applications, it might not be practical to analyze the entire core file in its raw form. IBM provides tools to process the data to produce meaningful and easily interpreted output. We describe these tools in 8.5, “IBM Support Assistant” on page 197.

8.5 IBM Support Assistant

IBM Support Assistant (ISA) is a workbench that works with other IBM software to help users with problem determination. For WebSphere Application Server, ISA analyzes the application server through the generated data to provide a helpful visual representation of the JVM state.

For this book, we describe three popular tools that are used for analyzing the three forms of generated JVM data in the previous sections:

- Garbage Collection and Memory Visualizer
- Memory Analyzer
- IBM Thread and Monitor Dump Analyzer for Java

8.5.1 Downloading IBM Support Assistant

IBM Support Assistant (ISA) is a no-charge tool that is available to help developers diagnose errors in IBM products. You can download it from the IBM website. Use the following procedure for downloading ISA:

1. Go to the following website, as shown in Figure 8-12. Click **Download**.

   http://www.ibm.com/software/support/isa

Figure 8-12 IBM Support Assistant

2. Click **Download** in the IBM Support Assistant Workbench section, as shown in Figure 8-13.

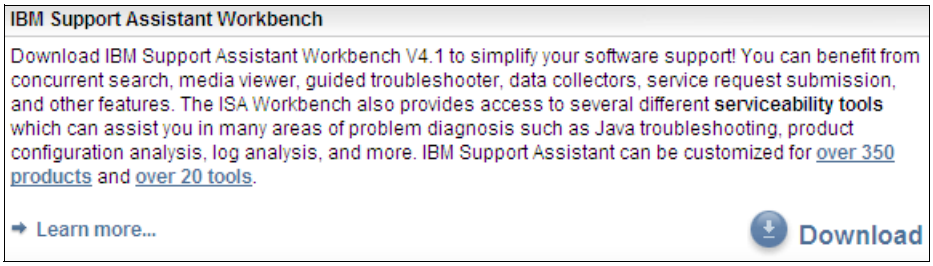


Figure 8-13 IBM Support Assistant download

3. Log in with your IBM ID and password, as shown in Figure 8-14. Click the Register here link if you do not have an IBM user ID, and register for an IBM ID before continuing with the instructions.

Figure 8-14 IBM Support Assistant: Returning visitors Sign in

4. Complete the License form, as appropriate, as shown in Figure 8-15, and click **I confirm.**

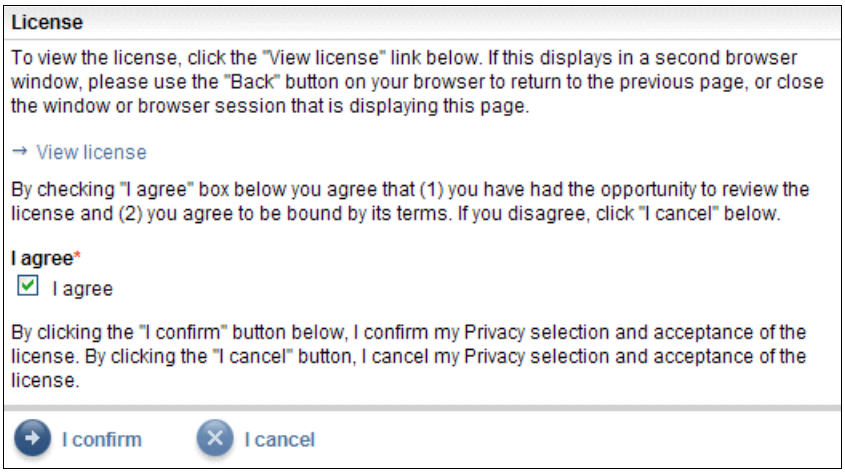


Figure 8-15 IBM Support Assistant: License agreement

5. Select **isa.wb.410-win32.zip** to download, as shown in Figure 8-16. A Download Director window opens, where the File attribute is the file path to which your selected package is downloaded. Click **Download now**.

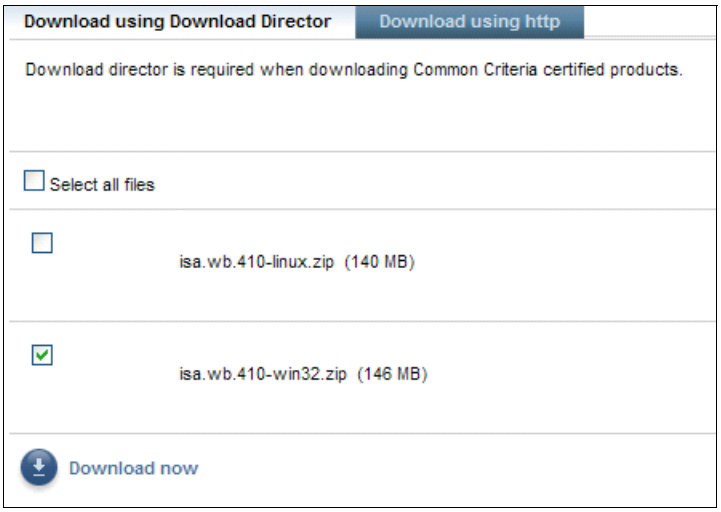


Figure 8-16 IBM Support Assistant: Download Director

8.5.2 Installing IBM Support Assistant

After downloading the ISA compressed archive file, you can run the setup wizard in the download package to install ISA. Use the following procedure to run the ISA setup wizard:

1. Extract the downloaded package.
2. In the folder that contains the extracted files, navigate to the **WAS** folder.
3. Click **Setupwin32.exe**.
4. Follow the steps that are provided by the install wizard.

8.5.3 Downloading the tools for data analysis

ISA is a workstation software product that hosts multiple tools that are provided by IBM for problem determination. You can download and install these tools through the ISA Add-ons manager. Use the following procedure to download these tools:

1. Access ISA by going to **Start** → **All programs** → **IBM Support Assistant** → **IBM Support Assistant Workbench 4.1**, as shown in Figure 8-17.

Figure 8-17 Accessing IBM Support Assistant

2. When ISA starts, follow the startup instructions for checking for updates.

3. Click **Update** → **Find New** → **Tools Add-ons** to open the download manager wizard, as shown in Figure 8-18.

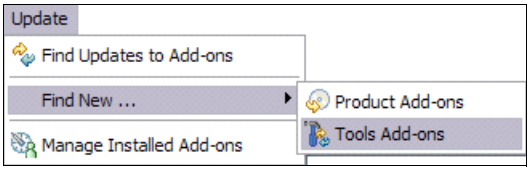


Figure 8-18 IBM Support Assistant: Downloading new tools

4. You can select the tools that you want to download. For this exercise, expand **JVM-based Tools** and select **IBM Monitoring and Diagnostic Tools for Java - Garbage Collection and Memory Visualizer**, **IBM Monitoring and Diagnostic Tools for Java - Memory Analyzer**, and **IBM Thread and Monitor Dump Analyzer for Java**, as shown in Figure 8-19. Click **Next**.

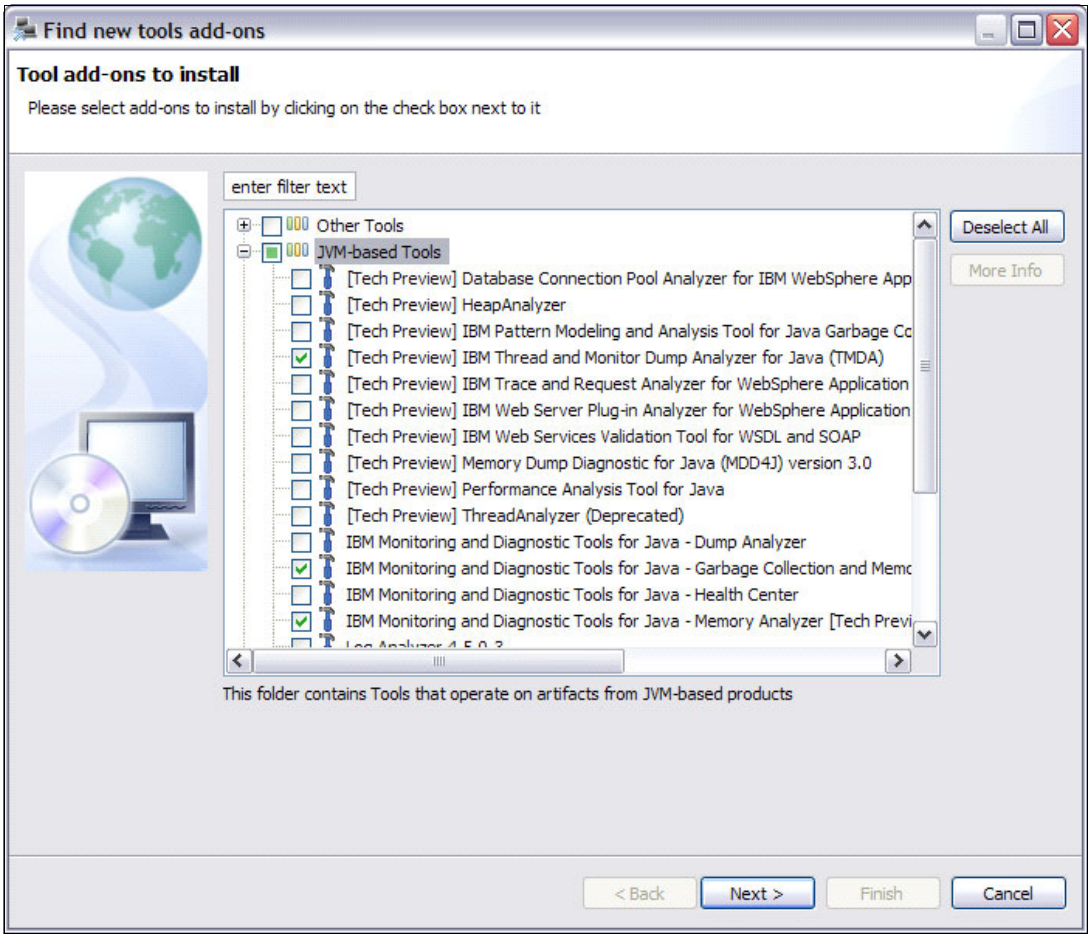


Figure 8-19 IBM Support Assistant: Downloading manager menu

5. Accept the terms of agreement, as shown in Figure 8-20. Click **Next**.

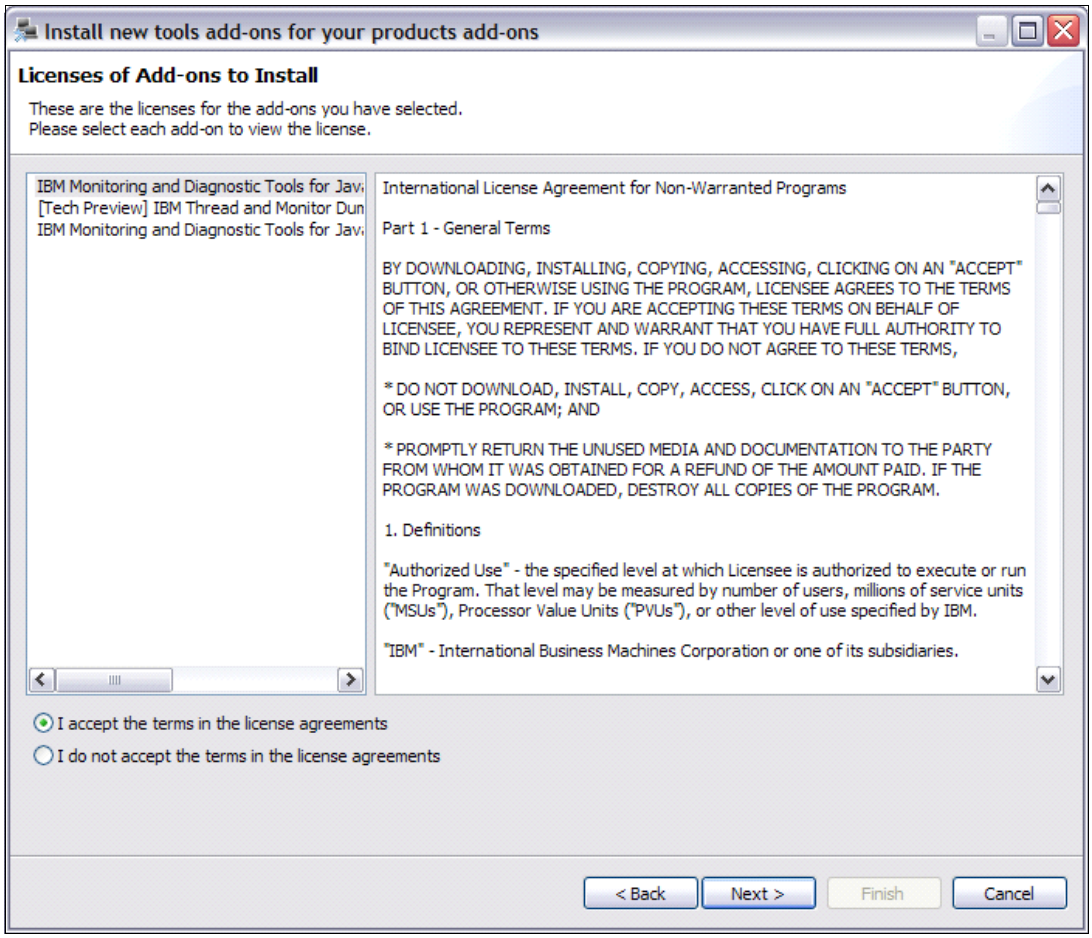


Figure 8-20 IBM Support Assistant: Licensing terms

6. Review your selections, and click **Finish** to download the selected tools, as shown in Figure 8-21.

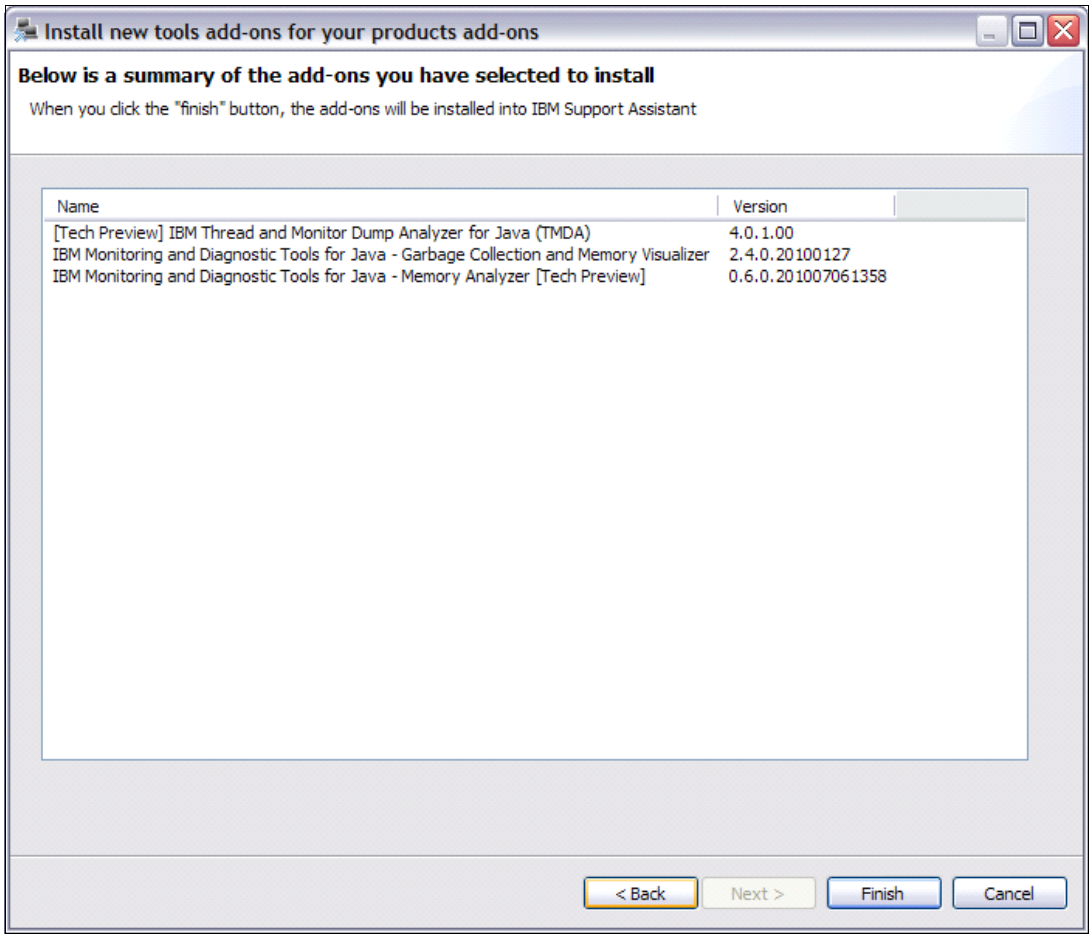


Figure 8-21 IBM Support Assistant: Tools installation summary

7. Afterward, ISA prompts you to restart the program to update the new settings. Restart ISA so that the new tools are installed.

8.6 Using Garbage Collection and Memory Visualizer

You use the Garbage Collection and Memory Visualizer tool to analyze verbose garbage collection data. The tool organizes the collected data and aggregates it as a graph of memory usage. This tool is especially useful for identifying memory leaks in the application server.

Important: Before you use the Garbage Collection and Memory Visualizer tool, make sure that you have enabled verbose garbage collection in the application server and have the verbose garbage collection output in the `native_stderr.log` file.

Use the following procedure to learn how to use the Garbage Collection and Memory Visualizer:

1. In the Welcome to IBM Support Assistant Workbench home page, select **Analyze Problem**, as shown in Figure 8-22.

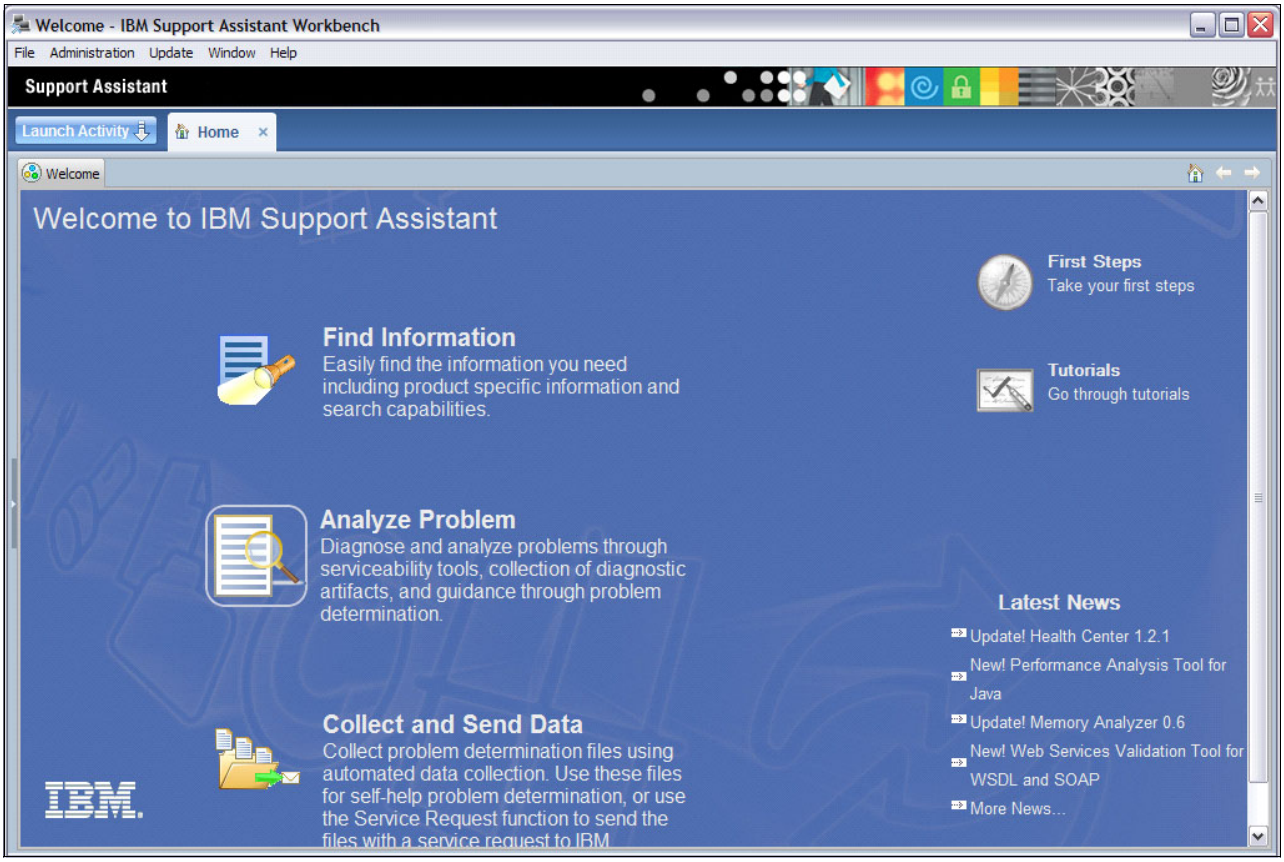


Figure 8-22 IBM Support Assistant Workbench home page

2. From the list of tools, select **IBM Monitoring and Diagnostic Tools for Java - Garbage Collection and Memory Visualizer**, as shown in Figure 8-23.

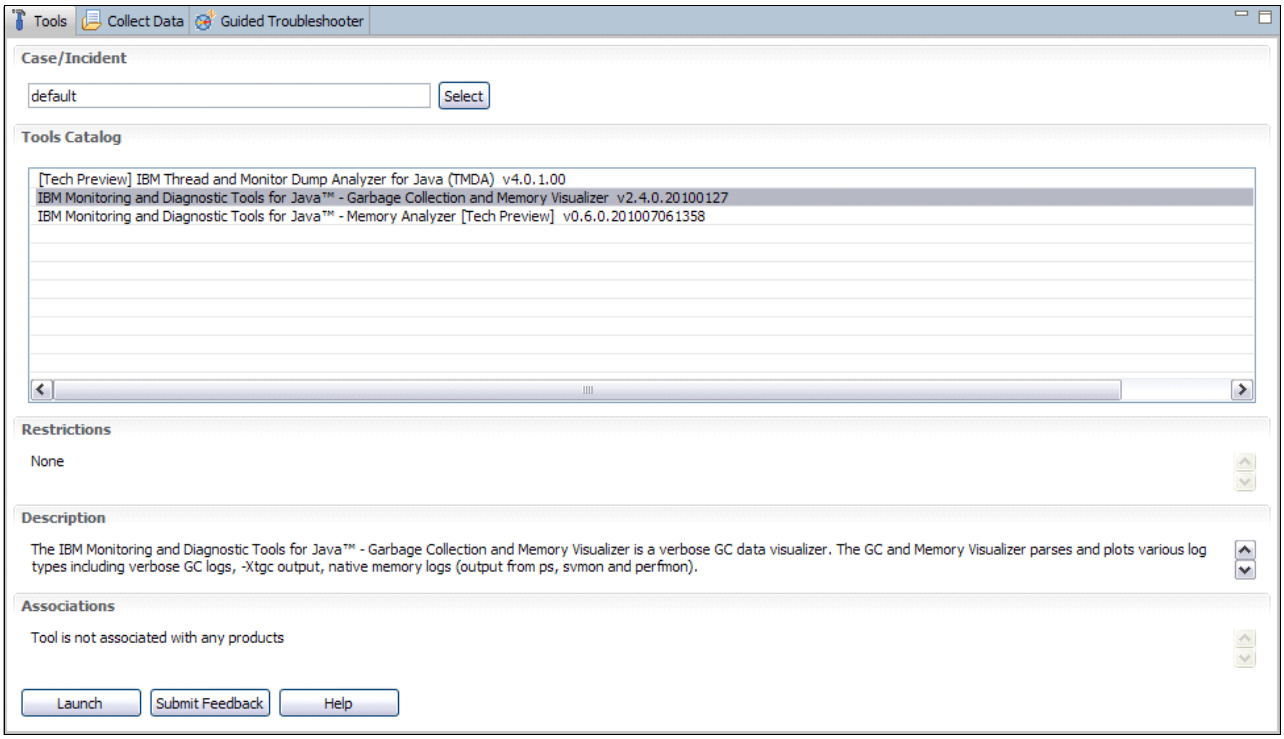


Figure 8-23 IBM Support Assistant Tools Catalog page

3. An input dialog opens, prompting for the Log name to analyze, as shown in Figure 8-24. Click **Browse**, and navigate to the `native_stderr.log` in the Remote Artifact Browse tab. Click **OK** to confirm your selection, and click **Next** to begin analyzing the garbage collection data.

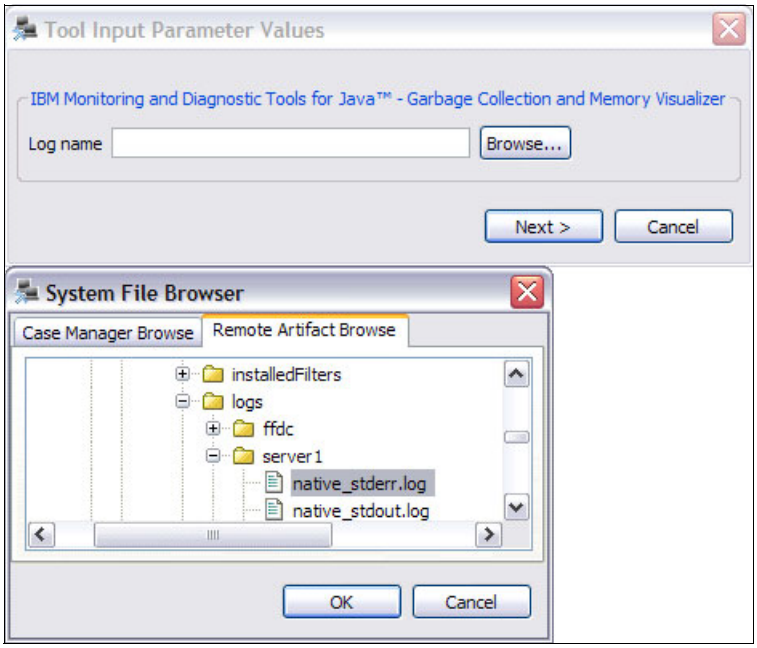


Figure 8-24 Garbage collection visualizer: Choose file to analyze

4. A new tab opens that contains the analysis of the garbage collection log, as shown in Figure 8-25. In the center of the tab, a graph shows the JVM heap memory usage as a function of time. The green line represents the amount of heap that is used by the application server, the red line shows the maximum heap size supported, and the blue line demarcates when the JVM has been reset. If a memory leak exists within your application, you can see an upward slope in the green line after long periods of time.

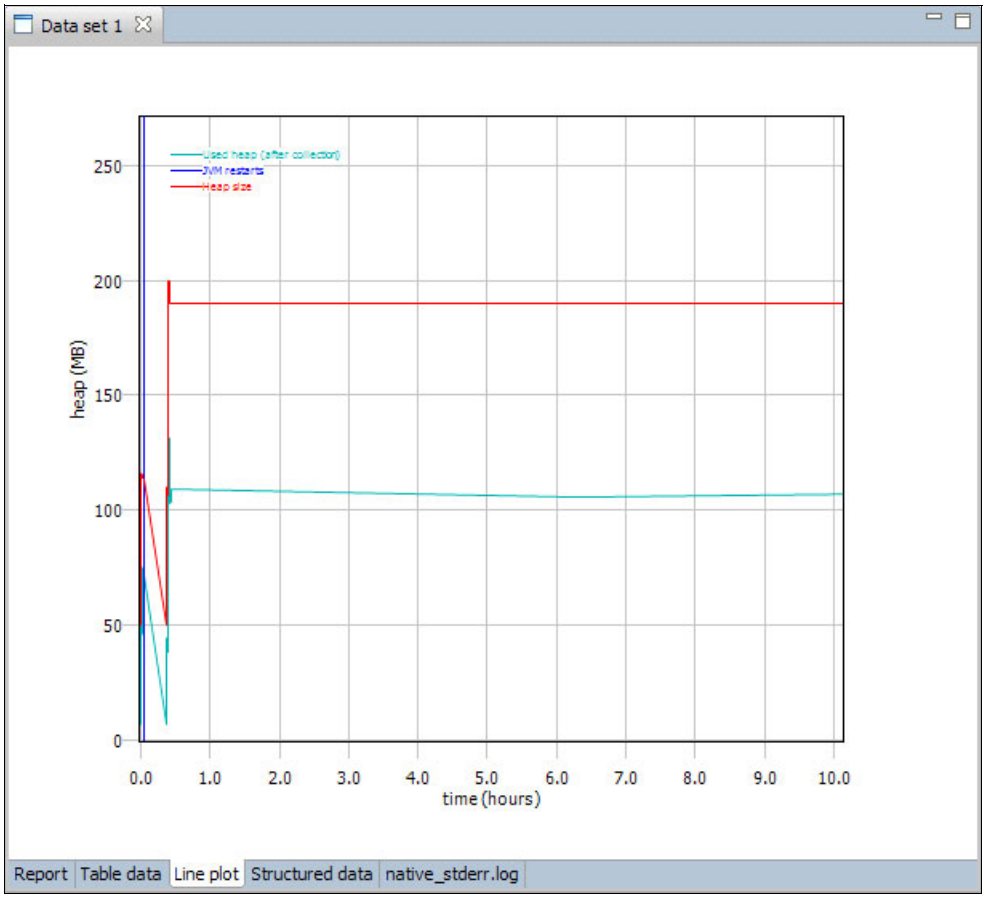


Figure 8-25 Garbage Collection Visualizer graph

Many other options are available in the Garbage Collection and Memory Visualizer that we do not describe here. If you want a more in-depth tutorial on this ISA tool, visit the following website:

http://www.ibm.com/developerworks/java/library/j-ibmtools2/#N100AF

8.7 Using Memory Analyzer

After viewing the application server JVM heap usage trend, it is useful to analyze the distribution of the memory. To see how the memory is partitioned, you can use ISA to analyze the heap dump of a given process. The Memory Analyzer is designed to profile the JVM state at any one instance to produce meaningful output regarding how memory is used, down to the level of every object instance. This tool is especially useful for determining the root cause of memory leaks and identifying memory-inefficient sectors of your application.

Important: Before you use the Memory Analyzer, make sure that you have generated the Java heap dump for analysis, as explained in 8.3, “Generating a Java heap dump” on page 191.

Use the following procedure to learn how to use the Memory Analyzer:

1. In the IBM Support Assistant Workbench home page, select **Analyze Problem**, as shown in Figure 8-26.

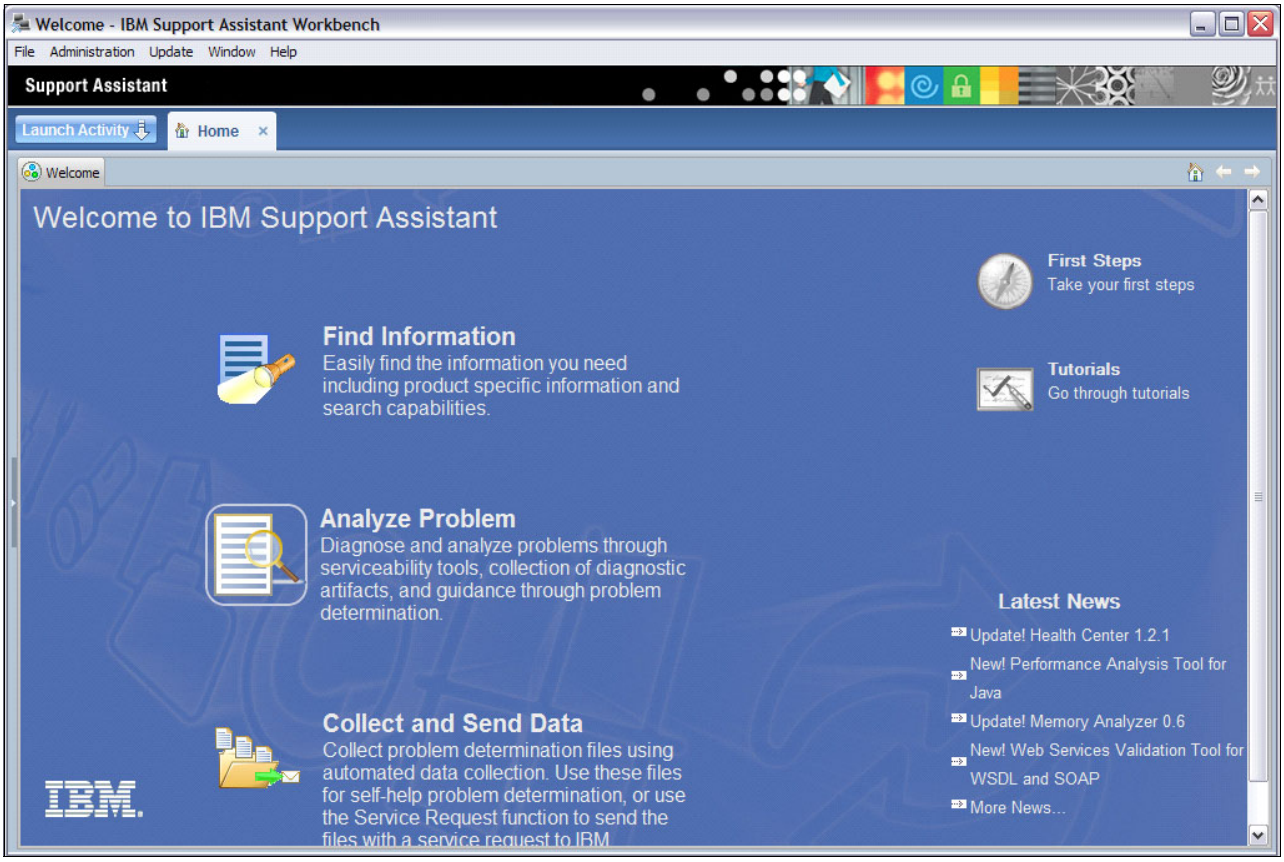


Figure 8-26 IBM Support Assistant home page

2. From the list of tools, select **IBM Monitoring and Diagnostic Tools for Java - Memory Analyzer**, as shown in Figure 8-27.

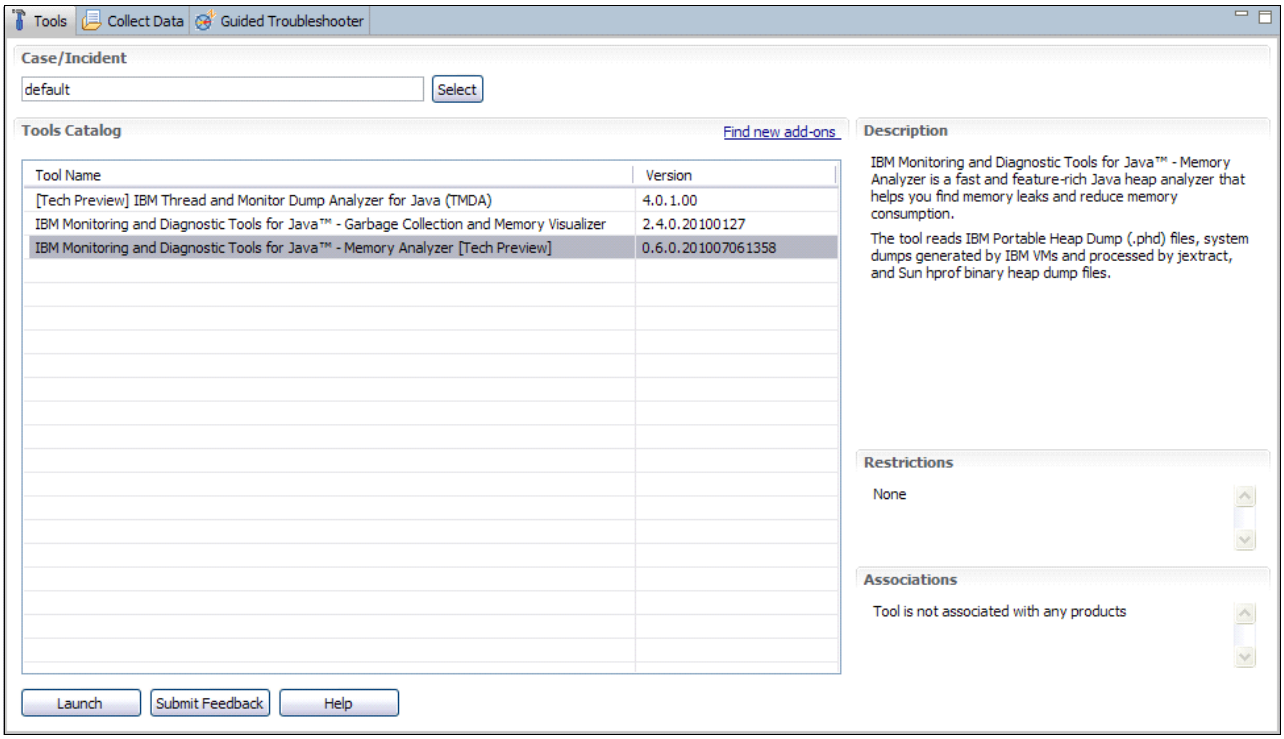


Figure 8-27 IBM Support Assistant: Tools Catalog page

3. An input dialog opens, prompting for the file name to analyze, as shown in Figure 8-28. Click **Browse**, and navigate to the Java heap dump file in the Remote Artifact Browse tab. Click **OK** afterward to confirm your selection, and click **Next** to begin analyzing the Java heap dump file.

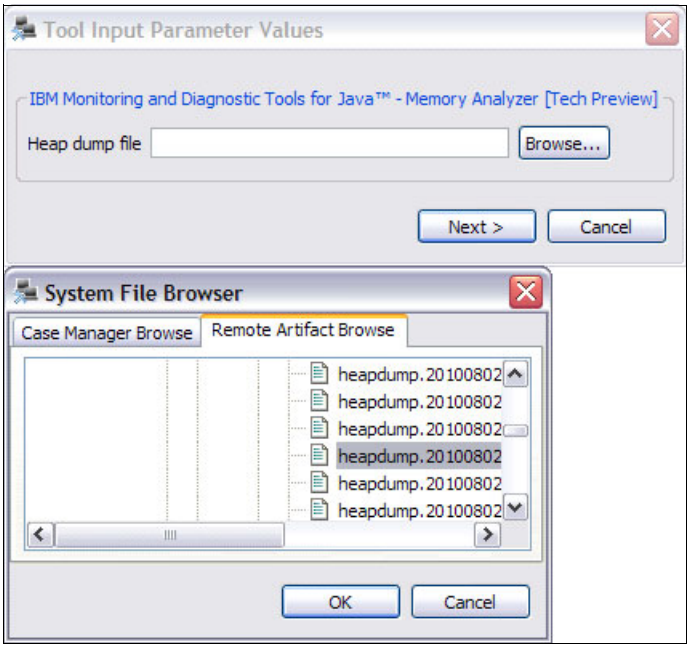


Figure 8-28 Memory Analyzer: Choose file to analyze

4. The IBM Monitor and Diagnostic Tools tab appears. Inside this tab, you can check the heap retention of each object in the server JVM. In the upper-left section, click the **Open Dominator Tree for Entire Heap** icon, as shown in Figure 8-29.

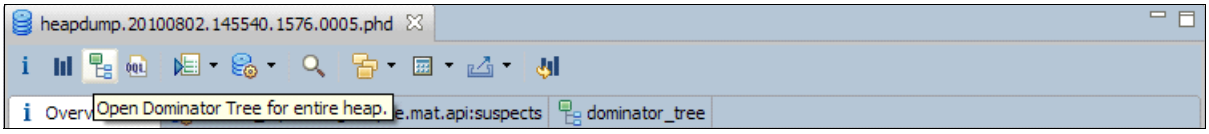


Figure 8-29 Memory Analyzer menu options

This icon displays a list of all of the objects in the JVM at the time of the snapshot, as shown in Figure 8-30.

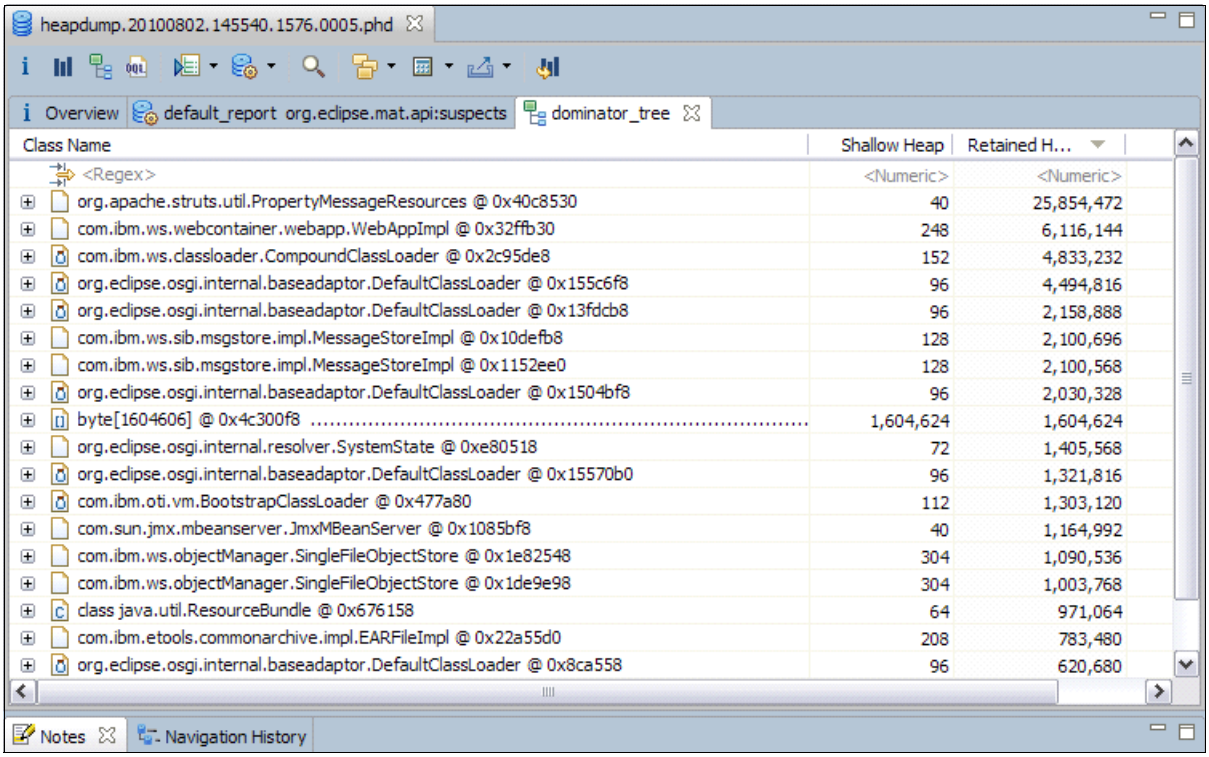


Figure 8-30 Memory Analyzer dominator tree

This view is useful for identifying memory-excessive elements, because you can sort the entries by their retained heap size. This view is too large to search for a specific object or a small group of objects. Instead, it is best to filter the results through the Object Query Language (OQL) scripting option. OQL works similarly to SQL. It queries the current JVM state for objects that fit a filter in a syntax, similar to SQL.

For example, for DictionaryApp V4, you can query for the memory usage of all of the DictionaryServlet instances in the JVM by using the following procedure:

1. In the Memory Analyzer, click the **OQL** icon that displays `Open Object Query Language studio to execute statements`, as shown in Figure 8-31. Clicking this icon presents an OQL script editor.

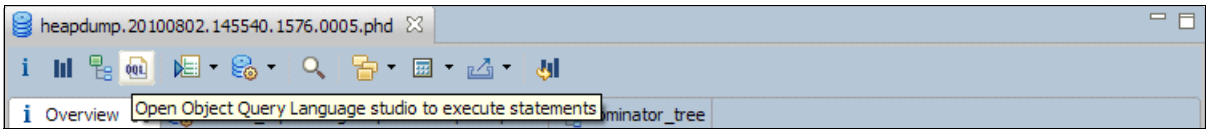


Figure 8-31 Memory Analyzer menu options

2. In the script editor, enter the following text:

```
SELECT * FROM
com.ibm.dictionaryapp.servlet.DictionaryServlet
```

3. Run the script by clicking the exclamation mark (!) icon that says `Execute Query`. If DictionaryApp V4 is running within the application server (the application has been deployed and accessed at least one time), you get output similar to the output that is shown in Figure 8-32.

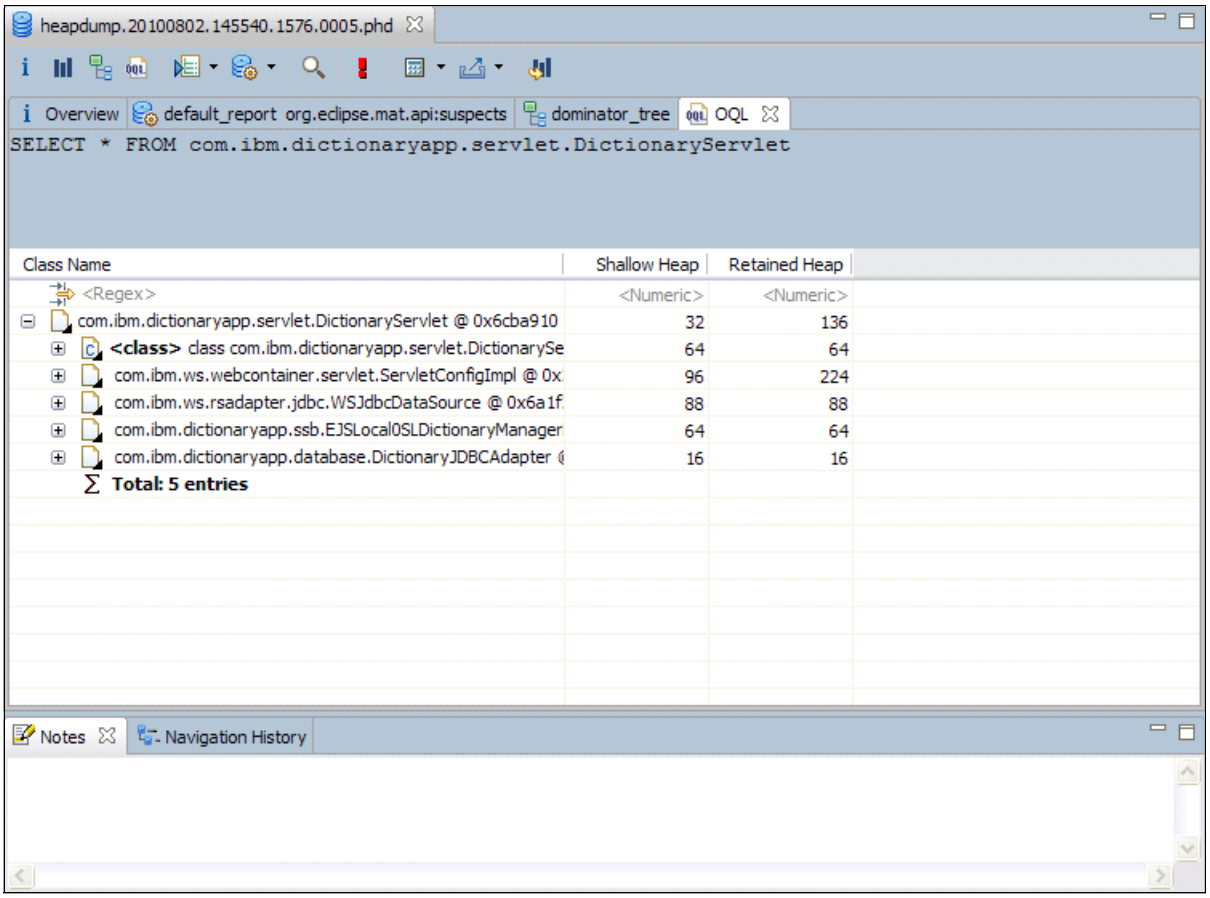


Figure 8-32 OQL query result

In the output table, you can check the individual memory makeup for each instance of DictionaryServlet and for each of its components. The Shallow Heap column shows the amount of heap memory that is consumed by that object. The Retained Heap column shows

the amount of heap memory that will be garbage-collected if this object were to be garbage-collected. The retained heap shows the sum of other reachable objects. If you see a large retained heap, or if the retained heap continues to grow in successive heap dumps, this object is a candidate to investigate. You can investigate whether this object, or other objects to which it points, might be the cause of a memory leak.

This exercise only explores the surface of Memory Analyzer and OQL. For more information about the ISA Memory Analyzer tool, go to the following website:

http://www.ibm.com/developerworks/java/jdk/tools/memoryanalyzer/

For more information about OQL, go to the following website and search for “OQL”:

http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1

8.8 Using IBM Thread and Monitor Dump Analyzer for Java

The *IBM Thread and Monitor Dump Analyzer for Java* analyzes Java core dumps and diagnoses monitor locks and thread activities to identify the root cause of the system stopping (or “hanging”), deadlocks, resource contentions, and monitor bottlenecks.

Important: Before you use the Thread and Monitor Dump Analyzer, make sure that you have generated the Java core dump files for analysis, as explained in 8.4, “Generating Java core dump” on page 195.

Use the following procedure to learn how to use several of the features of Thread and Monitor Dump Analyzer for Java:

1. In the ISA home page, select **Analyze Problem**, as shown in Figure 8-33.

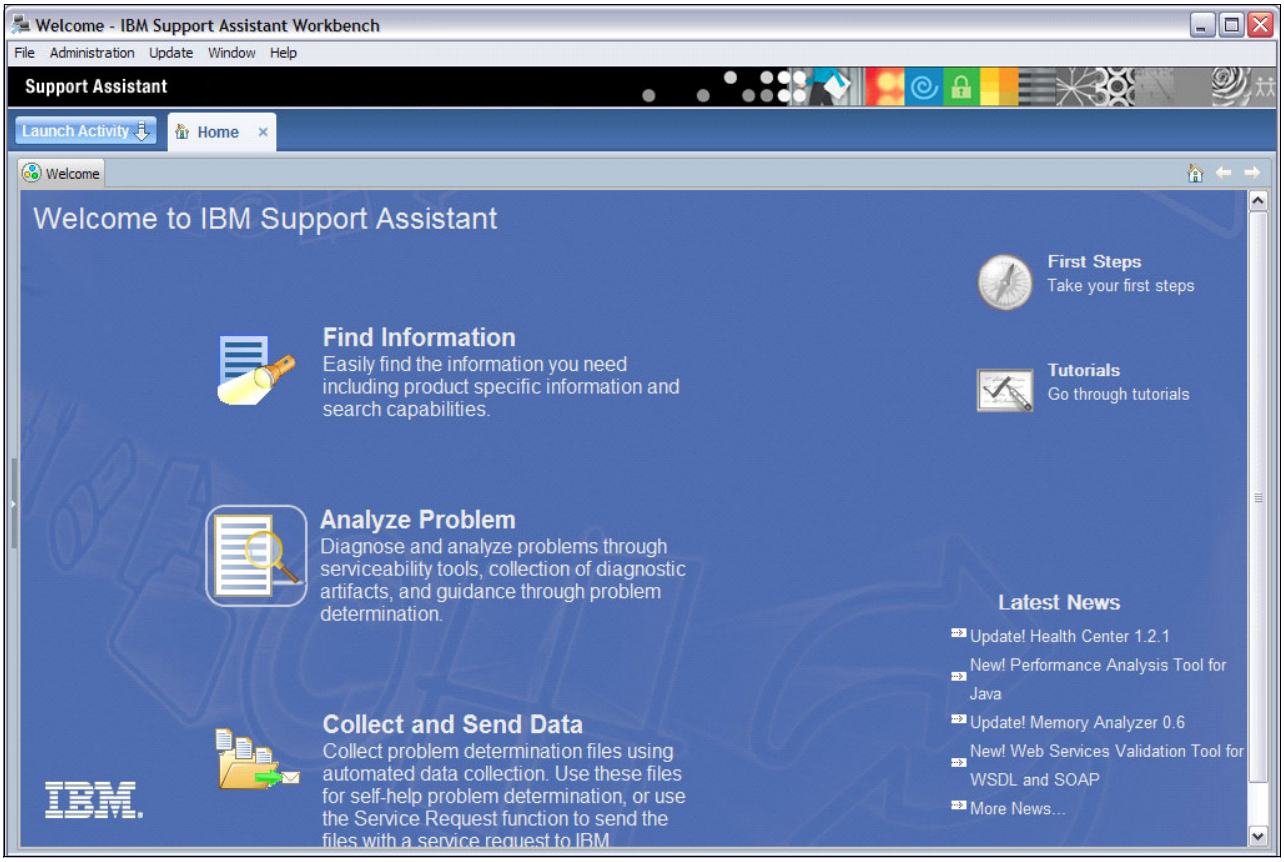


Figure 8-33 IBM Support Assistant Workbench home page

2. From the list of tools, select **IBM Thread and Monitor Dump Analyzer for Java**, as shown in Figure 8-34.

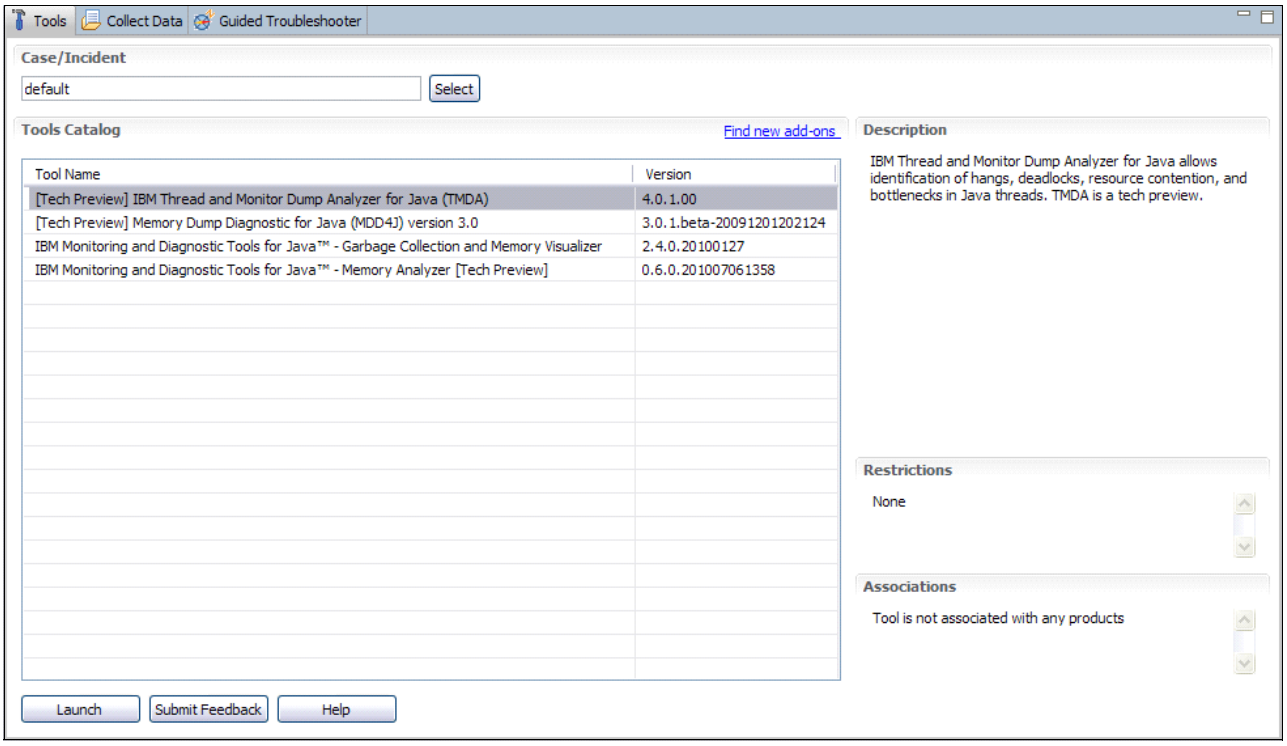


Figure 8-34 IBM Support Assistant: Tools Catalog page

3. The IBM Thread and Monitor Dump Analyzer for Java window opens, as shown in Figure 8-35. To browse a new thread dump, click **File** → **Open**. Browse to the Java core dump file, and click **New**.

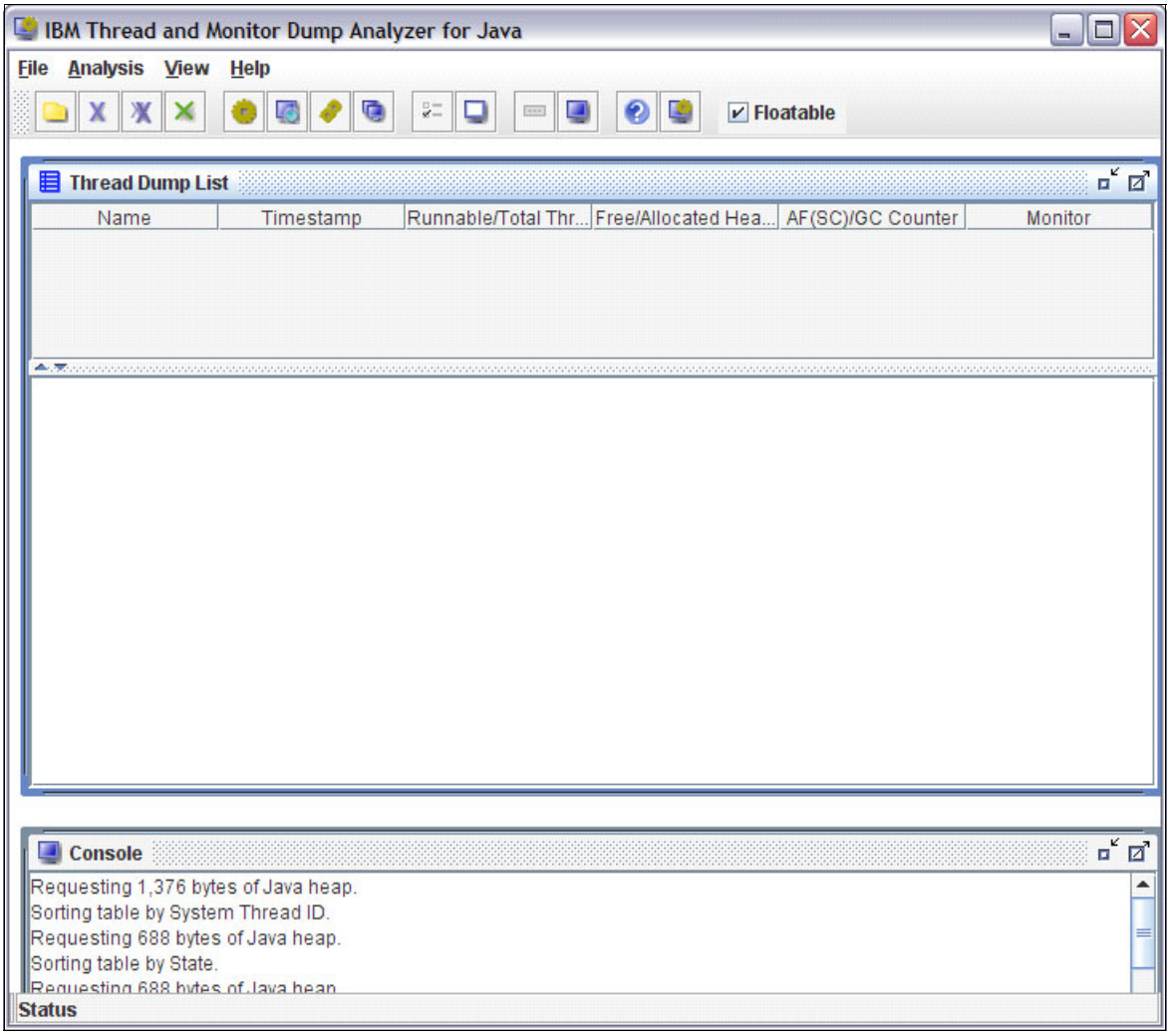


Figure 8-35 IBM Thread and Monitor Dump Analyzer for Java

4. View the summary for a thread dump by clicking it in the Thread Dump List, as shown in Figure 8-36.

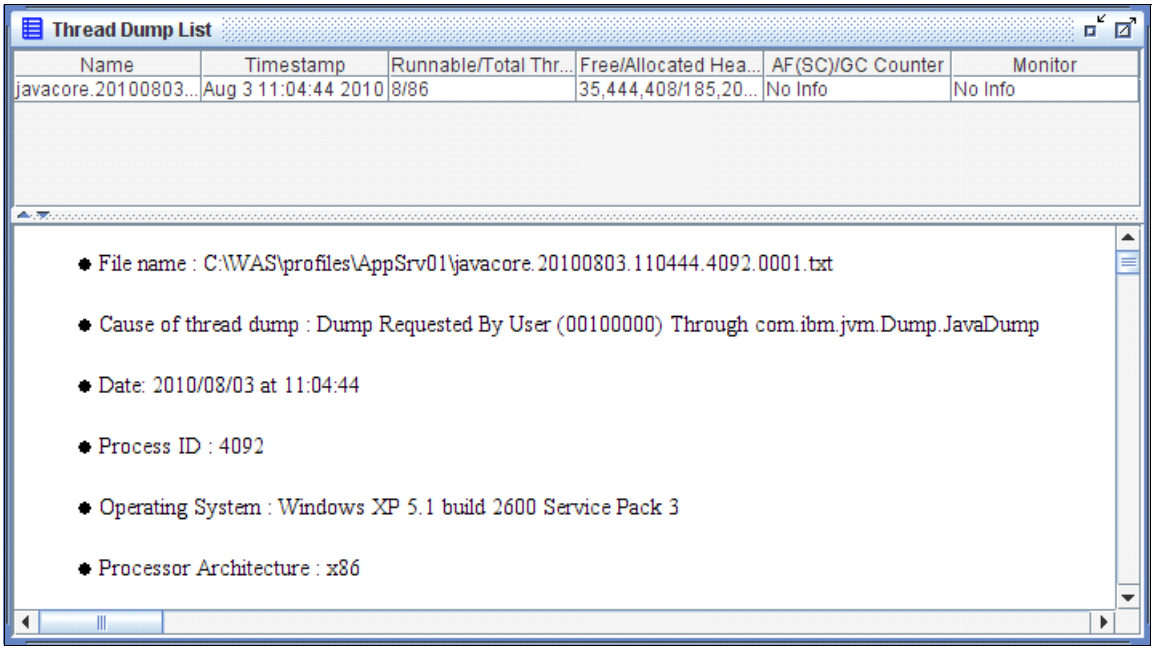


Figure 8-36 Thread Dump List

5. To view the threads that are running in the Java core dump, select **Analysis** → **Thread Detail** in the menu, as shown in Figure 8-37.

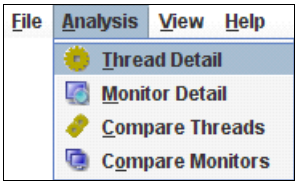


Figure 8-37 Thread and Monitor Dump Analyzer menu

6. The Thread Detail window opens, as shown in Table 8-1.

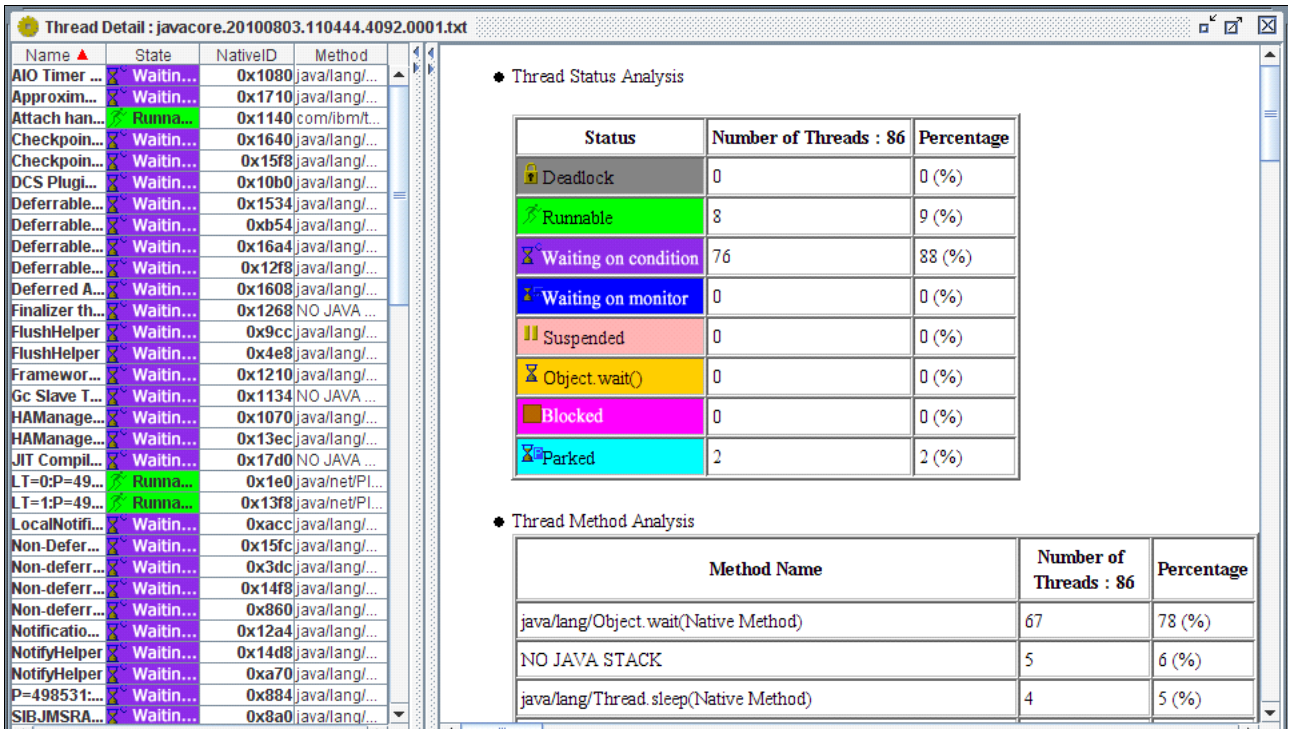


Figure 8-38 Thread and Monitor Dump Analyzer for Java: Thread Detail

7. The Thread Status Analysis that is shown in Figure 8-38 lists the states of the threads. Threads can be in any of the states that are described in Table 8-1.

Table 8-1 Thread analysis states

State	Description
Runnable	The thread can run when given the chance.
Condition Wait	The thread is waiting, for example, because of one of these reasons: ► A sleep() call is made. ► The thread has been blocked for I/O. ► A wait() method is called to wait on a monitor being notified. ► The thread is synchronizing with another thread with a join() call.
Monitor Wait	The thread is waiting on a monitor lock.
Suspended	The thread has been suspended by another thread.
Object.wait()	The method Object.wait() has been called on the thread.
Blocked	The thread is waiting to obtain a lock that something else currently owns.
Deadlock	The thread is in deadlock.

8. The Thread and Monitor Dump Analyzer also allows you to compare thread dumps taken at separate times. Select the thread dumps to be compared. Then, select **Analysis** → **Compare Threads** from the menu, as shown in Figure 8-39.

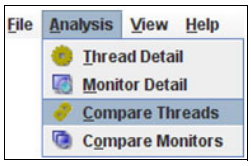


Figure 8-39 Thread dump options

9. This tool, as shown in Figure 8-40, is especially useful for comparing the progress of tasks within the application to identify the tasks that might cause slowdowns. You can detect slowdowns if you take thread dump snapshots at regular intervals, for example, one minute, and then use the Compare Threads option to compare these thread dumps.

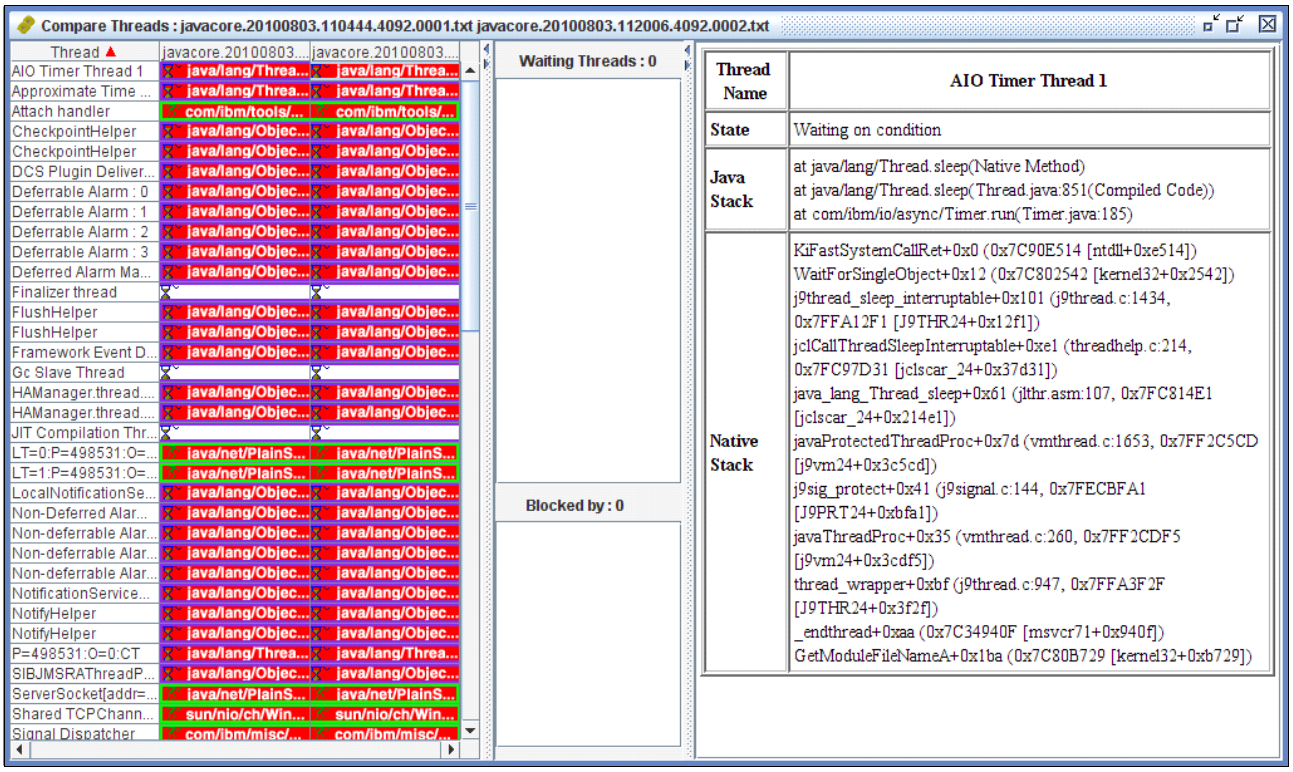


Figure 8-40 Thread and Monitor Dump Analyzer: Compare Threads option

10. Thread and Monitor Dump Analyzer also allows you to monitor the locks that each thread holds at the time of the snapshot. Click **Analysis** → **Monitor Detail** from the menu, as shown in Figure 8-41.

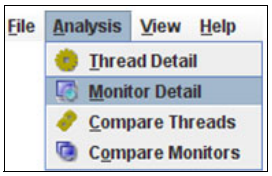


Figure 8-41 Thread dump options

11. This view, as shown in Figure 8-42, is especially useful for determining resource contentions between separate tasks.

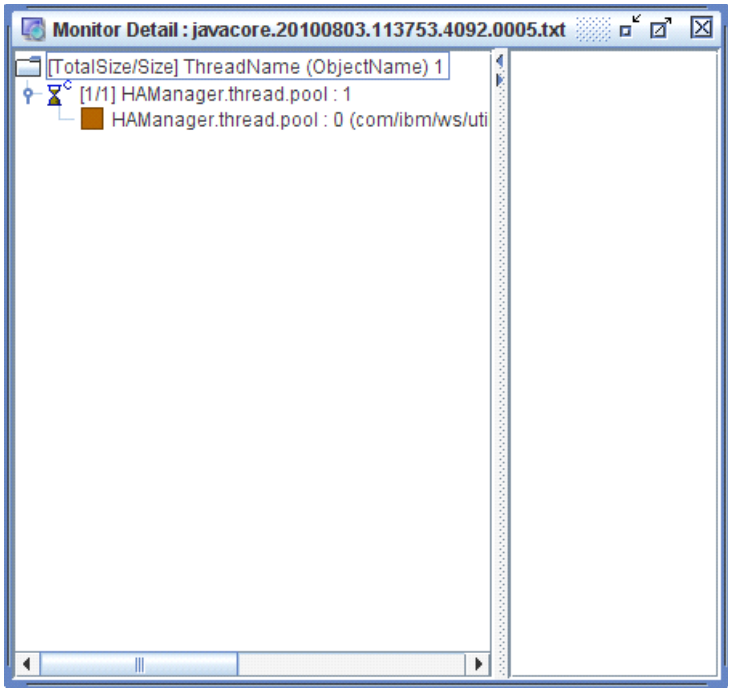


Figure 8-42 Thread Monitor Detail option

In this chapter, we have provided only a brief introduction to Thread and Monitor Dump Analyzer. You can obtain more information at the following website:

`http://www.alphaworks.ibm.com/tech/jca`

A

Development tools reference

The WebSphere run time places no restriction on how you create your application. You can develop Java Platform, Enterprise Edition (JEE) applications without an integrated development environment (IDE) by building your code with a text editor and by assembling your application with tools, such as ant or maven. You can customize your ant script to call into WebSphere scripting to automate your deployment tasks.

You can use Eclipse, which you can download at no charge. Eclipse offers a graphical user interface (GUI)-based development environment for JEE applications and includes a Java debugger. You can export your application from Eclipse and deploy it to WebSphere by using the administrative console or scripting. Eclipse is the focus of this appendix.

You can also move up from Eclipse to IBM Rational Application Developer for WebSphere Software. Rational Application Developer for WebSphere has these characteristics:

- Runs on top of Eclipse
- Integrates directly with WebSphere
- Supports more than the JEE programming model
- Offers a built-in unit test environment
- Improves turnaround with large applications

Go to the following website to get more information about Rational Application Developer for WebSphere Software and to download a trial copy:

`http://www.ibm.com/software/awdtools/developer/application/`

Or, you can move up to the full version of Rational Application Developer, which gives you additional capabilities:

- Code visualization, such as Java, Enterprise JavaBeans (EJB), Java Persistence application programming interface (API) or JPA, and Web Services Description Language (WSDL)
- Connection to Enterprise Information Systems, such as CICS®, IMS™, and SAP
- Additional developer testing and analysis tools
- Collaborative coding and debugging

© Copyright IBM Corp. 2010. All rights reserved.

- Integration with source control, project management, and life cycle management tools, such as IBM Rational TeamConcert, IBM Rational ClearCase®, and IBM Rational RequisitePro®

You can obtain more information about Rational Application Developer at the following website:

http://www-01.ibm.com/software/awdtools/developer/application/

Training: Because Rational Application Developer is a super-set of Rational Application Developer for WebSphere Software, you can use the materials that are available in the education and training section of the Rational Application Developer product wiki with the trial copy of Rational Application Developer for WebSphere.

A.1 Downloading Eclipse

Eclipse is a Java IDE that was originally created by IBM. The Eclipse Foundation maintains Eclipse. We recommend that you use this lightweight yet powerful IDE for viewing the `DictionaryApp` example application.

This task involves downloading the Eclipse IDE with Web Tools Platform (WTP) for JEE developers. If you already have Eclipse for Java 2 Platform, Standard Edition (J2SE) for developers installed, download Eclipse with the WTP package, because it is easier and more reliable than using the built-in updater to find the Java EE plug-ins. For those individuals who already have Eclipse for Java EE developers, you can skip this section, but be sure to configure Eclipse to run with the IBM Java Runtime Environment (JRE) to ensure compatibility with WebSphere.

Follow these steps to download Eclipse:

1. Go to the following website to access the Eclipse Downloads page, as shown in Figure A-1:

   `http://www.eclipse.org/downloads/`

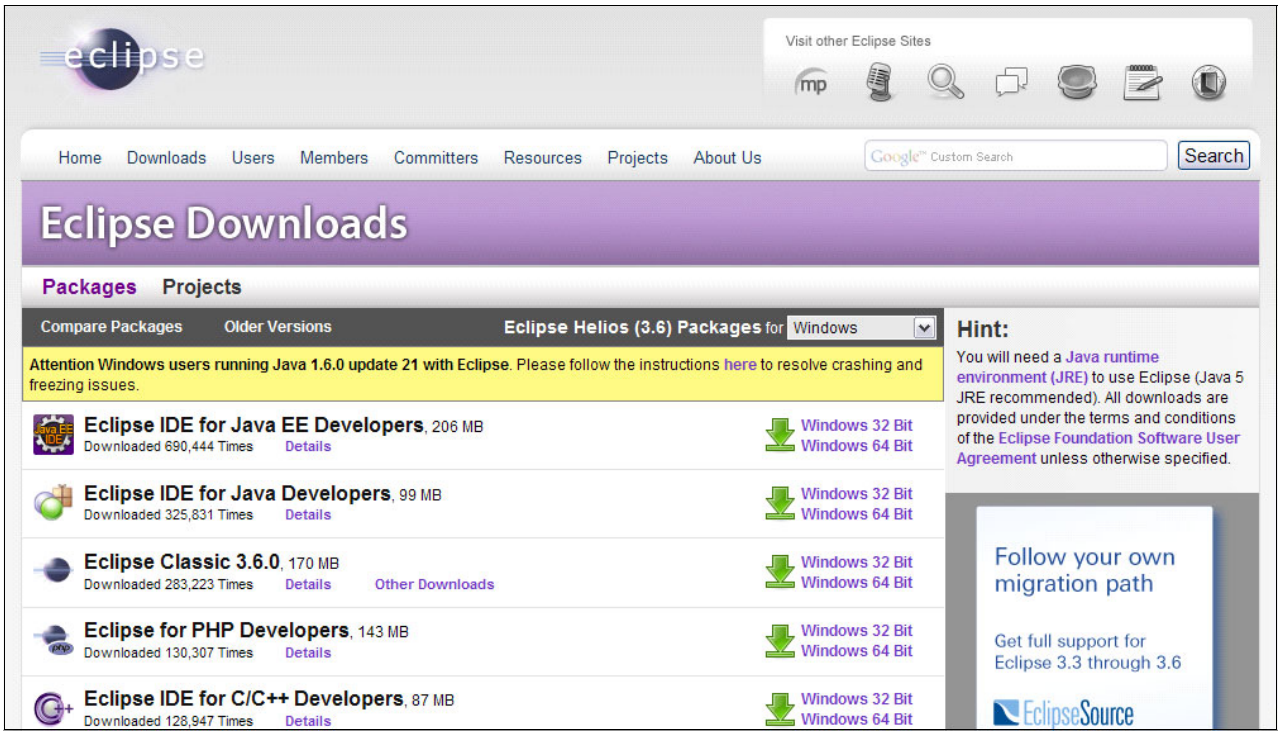


Figure A-1 Eclipse Download page

2. In the Eclipse IDE for Java EE Developers section, click either Windows 32 Bit or Windows 64 Bit.

3. In the Eclipse downloads - mirror selection page, click the Download link that is provided to begin downloading the Eclipse IDE, as shown in Figure A-2.

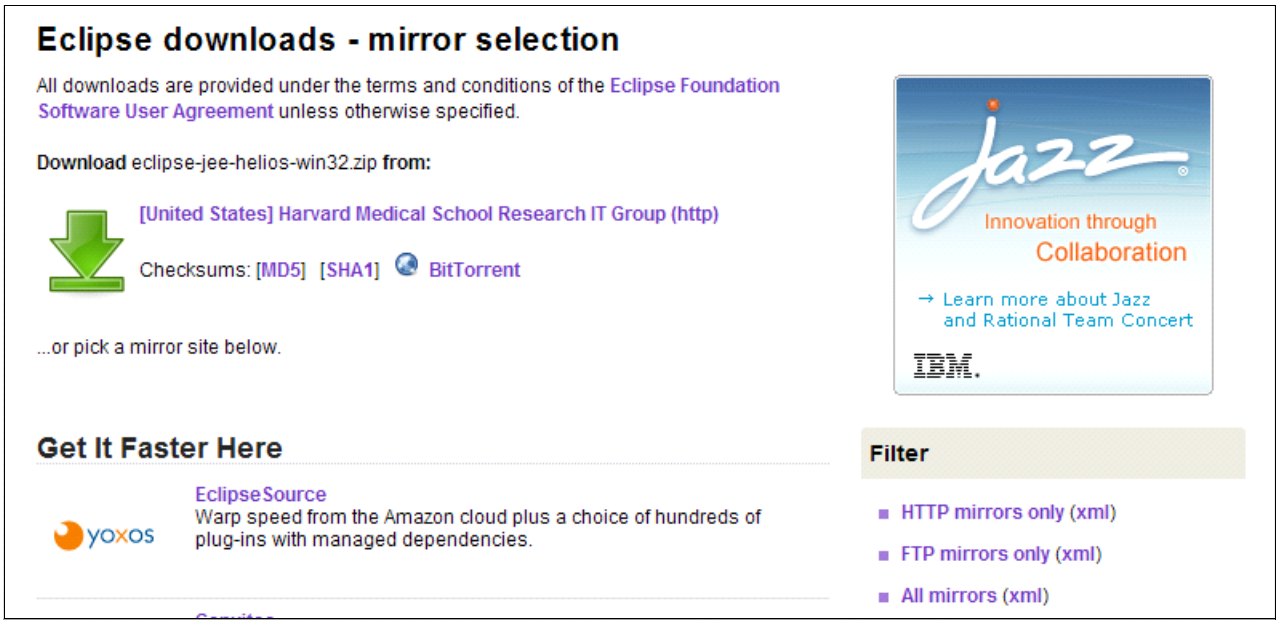


Figure A-2 Eclipse downloads - mirror selection page

A.2 Installing Eclipse

The Eclipse download package does not come with a setup wizard. The executable for the IDE is packaged directly in the compressed archive file. We explain the details for accessing and preparing the Eclipse IDE in the following procedure:

1. Extract the downloaded package to a convenient location, where Eclipse will be accessed.
2. Navigate to the extracted directory and run **`eclipse.exe`** in the eclipse folder.
3. After Eclipse starts, click **Windows** → **Preferences**.
4. In the Preferences window, expand the Java menu, and click **Installed JREs**.
5. To develop web applications to be deployed on the application server, specify Eclipse to use the IBM Java developer kit. Therefore, click **Add**, and select **Standard VM**. Specify the following file path for the JRE home:

   `install root``\java\jre`
6. Click **Finish**, and click **OK** to apply the changes. The Eclipse IDE is set up and ready for development.

A.3 Importing DictionaryApp into Eclipse

You can either deploy the EAR files that are provided with this book directly on the application server, or you can import them into Eclipse for reference:

1. In Eclipse, click **File** → **Import**.
2. In the import wizard, select **Java EE** → **EAR** file, and click **Next**.

3. Specify the file path to the version of `DictionaryApp.ear` that you want to import. You can also rename the EAR project that will be imported. Click **Next**.
4. Click **Select all** in the Utility JARs and web libraries page to import all utility jars, and then click **Next**, which includes the `DictionaryUtility.jar` file for all of the versions of DictionaryApp. In addition, DictionaryApp V4 and V5 also contain web services-related jars.
5. Click **Finish** to import the project.
6. Add WebSphere `j2ee.jar` to the build class path. This step is needed so that you do not get compilation errors for code that references JEE specification-defined classes. Follow these steps:
   a. Go to **Windows** → **Open Perspective** → **Web**.
   b. For each module and utility jar, including DictionaryWeb, DictionaryEJB (if present), and DictionaryUtility, right-click and choose **Properties**.
   c. In the pop-up window, click the **Libraries** tab.
   d. Ensure that **Java Build path** is selected on the left side of the window, and click **Add External Jars** on the right.
   e. Browse, and add *`install_root`*`/lib/j2ee.jar`.

A.3.1 Working with separate versions of DictionaryApp

The easiest way to work with separate versions of DictionaryApp in Eclipse is to import each version into a separate workspace. For each version of the application that you want to import, go to **File** → **switch workspace** → **other** to switch to another workspace. You need to provide a separate directory name for each workspace. After you make the switch, Eclipse relaunches itself.

A.3.2 DictionaryApp V3

DictionaryApp V3 adds logging function to DictionaryApp to demonstrate the use of Java Message Service (JMS) and message-driven beans (MDBs). The most important files to follow in this version are the `LookupLoggerBean.java` and `DefineLoggerBean.java` MDBs. You must also add the handleSession() method to the DictionaryManager session bean. Table A-1 shows the locations of the files.

Table A-1 DictionaryApp3 file locations

File	Location
LookupLoggerBean.java	DictionaryEJB\ejbModule\com.ibm.dictionaryapp.mdb
DefineLoggerBean.java	DictionaryEJB\ejbModule\com.ibm.dictionaryapp.mdb
DictionaryManagerBean	DictionaryEJB\ejbModule\com.ibm.dictionaryapp.ssb

A.3.3 DictionaryApp V4

DictionaryApp V4 adds a web service component to DictionaryApp. The two most important files to follow in this version are `DictionaryApplication.java` and `DictionaryService.java`. Table A-2 shows the locations of the files.

Table A-2 DictionaryApp4 file locations

File	Location
DictionaryService.java	DictionaryWeb\src\com.ibm.dictionaryapp.rest\
DictionaryApplication.java	DictionaryWeb\src\com.ibm.dictionaryapp.rest\
web.xml	DictionaryWeb\WebContent\WEB-INF\

A.4 Exporting DictionaryApp EAR in Eclipse

After importing the `DictionaryApp.ear` file, you see an Enterprise Application project named DictionaryApp in your workspace. This project contains information about the modules that are contained in the EAR file. If you make changes to any source files in DictionaryApp and want to deploy DictionaryApp on the application server, you can export the Enterprise Application project through Eclipse as a new EAR file containing the updated modules. Use the following procedure to export the DictionaryApp project:

1. Right-click the DictionaryApp Enterprise Application project, and select **Export** → **Ear File**.
2. An export wizard displays. In the export wizard, select the destination file path to which you want to export the new EAR file under Destination. Select **Export source files** so that Eclipse exports the source files in the EAR file. Select **Overwrite existing file** so that Eclipse overwrites any existing EAR files under the destination file path. Click **Finish** to export the EAR file.

Now, the EAR file that is produced is ready to be deployed on the application server.

A.5 Debugging DictionaryApp in Eclipse

The Eclipse Remote Debugger is a powerful debugging tool with a user friendly interface. You can debug any application, including DictionaryApp, that is deployed on the application server with the Eclipse Remote Debugger. In this section, we explain how to access DictionaryApp through the Eclipse debugger. You can apply this method to any application that is deployed on the application server, as long as you are able to import the source files in Eclipse. Use the following procedure to start and use the Eclipse Remote Debugger:

1. Make sure that the application server is configured to start in debug mode. Refer to Chapter 8, “Debugging” on page 185 for instructions to enable debug mode on WebSphere.
2. In Eclipse, click **Run** → **Debug Configurations**.
3. Select **Remote Java Application** in the navigation menu. Click the **New launch configuration** icon.

4. In the Create Configuration window, enter the name for this debugging configuration. This name is useful for referencing the same settings for future debugging sessions:
   - For Project, select the name of the project that you want to debug. For DictionaryApp, you can select DictionaryWeb to debug DictionaryServlet.
   - For Host, enter the host name or IP address for the application that you want to debug. For DictionaryApp, if it is running within a WebSphere installation on the same machine, you can use `localhost`.
   - For Port, enter the debug port for the server hosting the application that you want to debug (port `7777` if you use the default profile that was configured during installation).

   Click **Apply** to apply the debug configuration. Now that the configurations have been created, you can reapply this configuration every time that you need to debug the same application with the same configurations.
5. Open DictionaryServlet and add a breakpoint to the beginning of the doGet method by right-clicking at the left margin of the method, and choosing **Toggle Breakpoint**. With remote debugging, Eclipse does not control the JVM that it attempts to debug. Instead, the application server must trigger the breakpoints. Therefore, deploy DictionaryApp if it has not already been deployed. Afterward, simply access the application normally. The browser stops while displaying DictionaryApp, because the application server process managing DictionaryApp triggered the breakpoint set in doGet and reported it to the Eclipse debugger. Check the Eclipse debug perspective. If the breakpoint was triggered correctly, you see the contents of the thread that is running DictionaryServlet displayed.

You can now debug step-by-step through DictionaryServlet.

A.6 Creating your own bindings file

The WebSphere-specific bindings files can be extremely useful for expediting the deployment process. For Java Platform, Enterprise Edition 5 (JEE5) or later applications, WebSphere provides the XML schema for these bindings files to help developers create them for their applications. You can import these XML schemas into Eclipse to get content assist while writing these files.

A.6.1 Importing WebSphere XML schema into Eclipse

Use the following procedure to import WebSphere XML schemas into Eclipse:

1. In Eclipse, click **Windows** → **Preferences**.
2. Select **XML** → **XML Catalog**. The XML Catalog Entries section lists user-defined and plug-in-defined catalog entries. To view details about an entry, select the entry.
3. Click **Add** to create a new catalog entry.
4. In the Location field, browse to the location of the WebSphere XML schema files, which are at this location:

   install root`\properties\schemas`

   Select the **ibm-web-bnd_1_0.xsd** schema for `ibm-web-bnd.xml` and the **ibm-ejb-jar-bnd_1_0.xsd** schema for `ibm-ejb-jar-bnd.xml`.

5. Select **Namespace Name** for the Key Type field, and accept the defaults for the Key value. Enter the Key, and click **OK**.

> **Important:** No two XML catalog entries can have the same Key type and Key pair.

You can repeat steps 4 on page 225 and 5 to import more XML schemas to Eclipse.

A.6.2 Creating an XML file with the XML schema

After you have imported the XML schema into Eclipse, you can explore the content assist feature by creating several of the WebSphere-specific customization files based on the schema. If you have already deployed an application through the administrative console or scripting, you can directly import the XML files that were created by WebSphere into your application as a starting point. If you want to create an XML file manually, we describe how to add `ibm-web-bnd.xml` to a web module in the following procedure:

1. Right-click the **WEB-INF** folder in the WebContent directory of your web module (the dynamic web project corresponding to your web module), and click **New** → **XML**.
2. For File name, enter `ibm-web-bnd.xml`. Click **Next**.
3. Select **Create XML file** from an XML schema file. Click **Next**.
4. Select the **Select XML** catalog entry, and choose the XML schema that references `ibm-web-bnd_0_1.xsd` from the catalog. Click **Next**.
5. Under Namespace Information, select the default prefix “**p**” listed, and click **Edit**. In the pop-up window, delete the entry in the Prefix text field, and click **OK**, which removes the prefix that is associated with the namespace in the XML file. Click **Finish** to create the XML file.

B

WebSphere configurations

This appendix contains the following sections:

- WebSphere Application Server directory reference
- Directory structure
- Changing server ports

© Copyright IBM Corp. 2010. All rights reserved.

B.1 WebSphere Application Server directory reference

This section contains information about the directory structure of the WebSphere installation, including the WebSphere directory structure, configuration directory, and configuration documents.

B.2 Directory structure

Table B-1 shows the directory structure of the WebSphere installation. The top-level directory where WebSphere is installed, *install_root*, is referred to as the install level in Table B-1. The profile level is located at *install_root*/profiles/*profile_name*.

Table B-1 WebSphere directory structure

Directory	Level	Description
bin	Install and profile	Contains WebSphere binary executables (scripts) or links to them.
config	Profile	Contains XML files for environment configurations.
configuration	Install	Contains the `config.ini` file used to control the Eclipse Equinox Open Service Gateway initiative (OSGi) base configuration and startup.
deploytool	Install	Contains the ejbdeploy tool.
derby	Install	Contains the embedded and network server versions of Derby database management server.
dev	Install	Contains jar files needed to compile application code, such as `j2ee.jar`.
etc	Install and profile	Contains support files, such as WIM Virtual Member Manager support, Java Management Extensions (JMX) support files, and WebSphere-Security web services setup samples.
features	Install	Contains several Eclipse feature files used to control, group, and update the WebSphere Eclipse plug-ins. Each subdirectory contains a `feature.xml` file that lists plug-ins and versions that have prerequisites and dependencies that must be configured, managed, and upgraded together.
firststeps	Install and profile	Contains the files for the firststeps application, which helps users configure and get started using WebSphere.

Directory	Level	Description
installableApps	Install and profile	Contains pre-packaged applications that you can install to the application server.
installedApps	Profile	Contains the deployed code from installableApps that the profile instances are configured to run.
installedConnectors	Install and profile	Contains the Java Enterprise Edition (EE) Connector Architecture (JCA) resource adapter archive files with connectors used by the application server to interface with external or existing systems.
java	Install	Contains the Java Software Development Kit.
lafiles	Install	Contains the WebSphere installation license agreement, which has been translated into multiple languages.
lib	Install	Contains support for Java JAR and RAR library files for the Java 2 Platform, Enterprise Edition (J2EE) environment, including files for WebSphere MQ Java support, command-line utilities, mail, and Java TCL (JACL) support.
links	Install	Contains links into the Eclipse plug-ins directory structure.
logs	Install and profile	Contains the log files for the major installation or, at the profile level, the server instances (`startServer.log`/`stopServer.log`, `stdout`, and so forth) running for this installation.
optionalLibraries	Install	Contains directories for optional Java JAR files that are available for the environment, such as JavaScript Widget Library (JWL), WebSphere Java Persistence API (JPA), Jython, Struts, and the JavaServer Faces (JSF)-Portlet bridge.
plugins	Install	Contains the Eclipse Equinox OSGi runtime bundles, plug-ins, and the core WebSphere runtime environment.
profileTemplates	Install	Contains directories for each of the installation profile types and the XML configuration files and ant scripts used to set up the profiles, that is, the installed configurations of WebSphere to fulfill a particular role.

Directory	Level	Description
profiles	Install	The default directory to store the configured profile installations of WebSphere for the platform based on profileTemplates. Note that profiles might be created and stored in separate directories.
properties	Install and profile	Contains the document type definition (DTD), XML configuration, and property files for the installation and profile.
runtimes	Install	Contains various client-side runtime JAR files for use outside of the application server. Examples include administrative thin client, web services thin client, and Enterprise JavaBeans (EJB) thin client.
samples	Install and profile	Contains sample applications that demonstrate application architecture or WebSphere extensions.
sar2war_tool	Install	Contains a script and XML configuration to create a WAR file from a Session Initiation Protocol (SIP) SAR file to allow it to be deployed to WebSphere.
Scheduler	Install	Contains sql ddl scripts that create the required database tables for various database management systems (DBMSs).
scriptLibraries	Install	Contains supporting libraries that enable scripting for WebSphere deployment and management.
systemApps	Install	Contains the system-level application deployment EAR files, including the File Transfer, Scheduler Calendar, Event Service, and Virtual Member Manager support applications.
temp	Install and profile	Temporary storage for WebSphere general use.
tranlog	Profile	Contains the `tranlog` and `log` files used for recovery and XA distributed transaction support for the instances.
UDDIReg	Install	Contains the client, database scripts, and creation scripts for implementation of the IBM Universal Description, Discovery, and Integration (UDDI) registry.
uninstall	Install	Contains a Java application and the configuration files needed to uninstall WebSphere.

Directory	Level	Description
universalDriver	Install	Contains a Java DB2J Java Database Connectivity (JDBC) driver.
util	Install	Contains Jacl scripts or UNIX shell scripts for setting up various WebSphere features, such as WebSphere cell core group or the Scheduler.
web	Install	Contains web pages documenting several of the WebSphere architecture models.
wstemp	Profile	Contains temporary configuration changes from the Deployment Manager.

B.2.1 Configuration directory

The default profile configuration directory (Profile Config Directory) is at this location:

install root/profiles/*profile name*/config

Table B-2 describes the structure of the configuration directory.

Table B-2 Profile configuration direction

Directory	Description
cells/<*cell_name*>	This directory is the root level of configuration for the cell. The directory contains a number of cell-level configuration settings files. Depending on the types of resources that have been configured, you might see the following subdirectories.
Subdirectories	
cells/<*cell_name*>/applications/	Contains one subdirectory for every application that has been deployed within the cell.
cells/<*cell_name*>/buses/	Contains one directory for each service integration bus defined.
cells/<*cell_name*>/nodes	Contains the configuration settings for all nodes and servers managed as part of this cell. The directory contains one directory per node. Each cells/<*cell_name*>/nodes/<*node_name*> directory contains node-specific configuration files and a server directory, which, in turn, contain one directory per server and node agent on that node.
cells/<*cell_name*>/nodes/<*node_name*>/servers	Contains the configurations for the servers of this node. The directory contains one directory per server. Each cells/<*cell_name*>/nodes/<*node_name*>/servers/<*server_name*> directory contains server-specific configuration files, such as server.xml, described next.

B.2.2 Configuration documents

The Profile Config Directory contains many configuration documents. Most configuration documents have XML content. Table B-3 describes the purpose of each file and its scopes.

Table B-3 Configuration documents

Configuration	Scopes	Purpose
admin-authz.xml	cells/<*cell_name*>	Define a role for administrative operation authorization.
app.policy	cells/<*cell_name*>/nodes/<*node_name*>	Define security permissions for application code.
cell.xml	cells/<*cell_name*>	Identify a cell.
deployment.xml	cells/<*cell_name*>/applications/<*application_name*>	Configure application deployment settings, such as target servers and application-specific server configuration.
filter.policy	cells/<*cell_name*>	Specify security permissions to be filtered out of other policy files.
integral-jms-authorzations.xml	cells/<*cell_name*>	Provide security configuration data for the integrated messaging system.
library.policy	cells/<*cell_name*>/nodes/<*node_name*>	Define security permissions for shared library code.
multibroker.xml	cells/<*cell_name*>	Configure a data replication message broker.
namestore.xml	cells/<*cell_name*>	Provide persistent name binding.
naming-authz.xml	cells/<*cell_name*>	Define roles for a naming operation authorization.
node.xml	cells/<cell_name>/nodes<*node_name*>	Identify a node.
pmirm.xml	cells/<*cell_name*>	Configure Performance Monitoring Infrastructure (PMI) request metrics.
resources.xml	cells/<*cell_name*> cells/<*cell_name*>/nodes/<*node_name*> cells/<*cell_name*>/nodes/<*node_name*>/servers/<*server_name*>	Define operating environment resources, including JDBC, JMS, JavaMail, URL, JCA resource providers, and factories.
security.xml	cells/<*cell_name*>	Configure security, including all user ID and password data.
server.xml	cells/<*cell_name*>/nodes/<*node_name*>/servers/<*server_name*>	Identify a server and its components.
serverindex.xml	cells/<*cell_name*>/nodes/<*node_name*>	Specify communication ports that are used on a specific node.
spi.policy	cells/<*cell_name*>/nodes/<*node_name*>	Define security permissions for service provider libraries, such as resource providers.

Configuration	Scopes	Purpose
variables.xml	cells/<*cell_name*> cells/<*cell_name*>/nodes/<*node_name*> cells/<*cell_name*>/nodes/<*node_name*>/servers/<*server_name*>	Configure variables that are used to parameterize any part of the configuration settings.
virtualhosts.xml	cells/<*cell_name*>	Configure a virtual host and its Multipurpose Internet Mail Extensions (MIME) types.

B.3 Changing server ports

You can modify the ports that are listed in the `serverindex.xml` file by using the administrative console or **`wsadmin`**.

B.3.1 Administrative console

Follow these steps:

1. In the administrative console, click **Servers** → **Application servers** from the navigation tree.
2. Click the name of the server that you want to change.
3. Under Configurations, in the Communications section, click **Ports**. This selection displays a list of ports. To change the port number of a particular port, click the port name that you want to change.
4. Save the changes, and restart the application server.

B.3.2 wsadmin

You can use the command that is shown in Example B-1 to change the ports on which the server listens. These changes are reflected in the `serverindex.xml` file. Use the same *`endpoint name`* that is used in the `serverindex.xml` file, for example, `WC_defaulthost`.

Example B-1 wsadmin changing server ports

```
AdminTask.modifyServerPort ('server name', '[-nodeName  devNode -endPointName
endpoint name -host new host name -port new port number]')
```

C

Additional material

This book refers to additional material that you can download from the Internet in the following manner.

Locating the web material

The web material associated with this book is available in softcopy on the Internet from the IBM Redbooks web server. Point your web browser at this website:

`ftp://www.redbooks.ibm.com/redbooks/SG247913`

Alternatively, you can go to the IBM Redbooks website at this website:

`ibm.com/redbooks`

Select **Additional materials**, and open the directory that corresponds with the IBM Redbooks form number, SG247913.

Using the web material

The additional web material that accompanies this book includes the following folders and files:

File/folder name	*Description*
Version 1	`DictionaryApp.ear` that is used in Chapter 3, “DictionaryApp example application” on page 63.
Version 2	`DictionaryApp.ear` that is used in Chapter 4, “Incorporating EJB3 and JPA into DictionaryApp V2” on page 99.
Version 3	`DictionaryApp.ear` that is used in Chapter 5, “Incorporating JMS and MDBs into DictionaryApp V3” on page 121.
Version 4	`DictionaryApp.ear` that is used in Chapter 6, “Incorporating RESTful web services into DictionaryApp V4” on page 167.

© Copyright IBM Corp. 2010. All rights reserved.

Version 5 `DictionaryApp.ear` that is used in Chapter 7, “Including WebSphere-specific bindings files for DictionaryApp V5” on page 179.

Web Service client DictionaryClient class that is used in Chapter 6, “Incorporating RESTful web services into DictionaryApp V4” on page 167.

DictionaryAppDerbyScript.sql SQL statements to populate the database. This script is used in Chapter 3, “DictionaryApp example application” on page 63.

Create a subdirectory (folder) on your workstation, and extract the contents of the web material compressed file into this folder. The directions for using each file are contained in the chapters where the file is referenced.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks” on page 238. Note that several of the documents referenced here might be available in softcopy only.

- *WebSphere Application Server V7 Administration and Configuration Guide*, SG24-7615

Online resources

These websites are also relevant as further information sources:

- WebSphere Application Server For Developers

  http://www.ibm.com/developerworks/websphere/downloads/
- webSphere Application Server V7 Information Center

  http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp
- WebSphere Application Server feature packs

  http://www.ibm.com/software/webservers/appserv/was/featurepacks/
- List of supported software for WebSphere Application Server V7.0

  http://www.ibm.com/support/docview.wss?rs=180&uid=swg27012369
- IBM Support Assistant (ISA)

  http://www.ibm.com/software/support/isa/
- Rational Application Developer for WebSphere software

  http://www.ibm.com/software/awdtools/developer/application/
- Eclipse

  http://www.eclipse.org
- IBM Java developer kit diagnostics

  http://www.ibm.com/developerworks/java/jdk/diagnosis/index.html
- ISA Garbage Collection

  http://www.ibm.com/developerworks/java/library/j-ibmtools2/#N100AF
- ISA Memory Analyzer

  http://www.ibm.com/developerworks/java/jdk/tools/memoryanalyzer/
- Obect Query Language

  http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1
- ISA Thread and Monitor Dump Analyzer

© Copyright IBM Corp. 2010. All rights reserved.

http://www.alphaworks.ibm.com/tech/jca

How to get IBM Redbooks

You can search for, view, or download IBM Redbooks publications, IBM Redpaper™ publications, web docs, draft publications and Additional materials, as well as order hardcopy IBM Redbooks publications, at this website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

WebSphere Application Server for Developers V7

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

WebSphere Application Server for Developers V7

Learn about WebSphere Application Server V7

Create and deploy sample applications

Enhance application features and functionality

This IBM Redbooks publication can help you install, tailor, and configure WebSphere Application Server for Developers V7 on the Microsoft Windows platform. WebSphere Application Server for Developers is a no-charge version of WebSphere Application Server for use in a development environment only. It allows application developers to develop and unit test against the same run time as the production version of WebSphere Application Server.

This book tells you how to perform these tasks:

- Download and install WebSphere Application Server for Developers V7.
- Use the command-line tools, web-based administrative console, and scripting tools.
- Deploy a web application with Java Database Connectivity (JDBC) to the application server with the first version of a sample application.
- Configure the sample application with Enterprise JavaBeans 3 (EJB3) and Java Persistence API (JPA).
- Add Java Message Service (JMS) and message-driven beans (MDBs) to the sample application and configure the built-in system integration bus (SIBus) messaging infrastructure.
- Add Representational State Transfer (RESTful) web service to the sample application.
- Incorporate WebSphere-specific application bindings files with the application.
- Enable debugging and produce and analyze JVM outputs.
- Learn how to use Eclipse to view and debug the sample applications.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7913-00 ISBN 0738434930