



macromedia[®]
JRUN™4

JRun Assembly and Deployment Guide

Trademarks

Afterburner, AppletAce, Attain, Attain Enterprise Learning System, Attain Essentials, Attain Objects for Dreamweaver, Authorware, Authorware Attain, Authorware Interactive Studio, Authorware Star, Authorware Synergy, Backstage, Backstage Designer, Backstage Desktop Studio, Backstage Enterprise Studio, Backstage Internet Studio, ColdFusion, Design in Motion, Director, Director Multimedia Studio, Doc Around the Clock, Dreamweaver, Dreamweaver Attain, Drumbeat, Drumbeat 2000, Extreme 3D, Fireworks, Flash, Fontographer, FreeHand, FreeHand Graphics Studio, Generator, Generator Developer's Studio, Generator Dynamic Graphics Server, JRun, Knowledge Objects, Knowledge Stream, Knowledge Track, Lingo, Live Effects, Macromedia, Macromedia M Logo & Design, Macromedia Flash, Macromedia Xres, Macromind, Macromind Action, MAGIC, Mediamaker, Object Authoring, Power Applets, Priority Access, Roundtrip HTML, Scriptlets, SoundEdit, ShockRave, Shockmachine, Shockwave, Shockwave Remote, Shockwave Internet Studio, Showcase, Tools to Power Your Ideas, Universal Media, Virtuoso, Web Design 101, Whirlwind and Xtra are trademarks of Macromedia, Inc. and may be registered in the United States or in other jurisdictions including internationally. Other product names, logos, designs, titles, words or phrases mentioned within this publication may be trademarks, servicemarks, or tradenames of Macromedia, Inc. or other entities and may be registered in certain jurisdictions including internationally.

This guide contains links to third-party websites that are not under the control of Macromedia, and Macromedia is not responsible for the content on any linked site. If you access a third-party website mentioned in this guide, then you do so at your own risk. Macromedia provides these links only as a convenience, and the inclusion of the link does not imply that Macromedia endorses or accepts any responsibility for the content on those third-party sites.

Apple Disclaimer

APPLE COMPUTER, INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING THE ENCLOSED COMPUTER SOFTWARE PACKAGE, ITS MERCHANTABILITY OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY PROVIDES YOU WITH SPECIFIC LEGAL RIGHTS. THERE MAY BE OTHER RIGHTS THAT YOU MAY HAVE WHICH VARY FROM STATE TO STATE.

Copyright © 2002 Macromedia, Inc. All rights reserved. This manual may not be copied, photocopied, reproduced, translated, or converted to any electronic or machine-readable form in whole or in part without prior written approval of Macromedia, Inc.
Part Number ZJR40M500

Acknowledgments

Project Management: Randy Nielsen

Writing: Michael Peterson

Editing: Linda Adler, Noreen Maher

First Edition: May 2002

Macromedia, Inc.
600 Townsend St.
San Francisco, CA 94103

CONTENTS

ABOUT THIS BOOK	V
Developer resources	vi
About JRun documentation	vii
Printed and online documentation set	vii
Accessing online documentation	vii
Other resources	viii
Contacting Macromedia	xi
CHAPTER 1 Configuring J2EE Modules	1
Understanding JRun deployment descriptors	2
Understanding J2EE roles in module configuration	3
Configuring web applications	4
Understanding the web.xml deployment descriptor	4
Understanding the jrun-web.xml deployment descriptor	7
Configuring Enterprise JavaBeans	9
Understanding the ejb-jar.xml deployment descriptor	9
Understanding the jrun-ejb-jar.xml deployment descriptor	17
Configuring enterprise resource adapters	24
Understanding the ra.xml deployment descriptor	24
Understanding the jrun-ra.xml deployment descriptor	26
Configuring enterprise applications	28
Understanding the application.xml deployment descriptor	28
Handling J2EE module dependencies	29
CHAPTER 2 Packaging J2EE Modules	31
Module packaging overview	32
Choosing between archive files and expanded directories	32
Understanding assembler, deployer, and administrator roles	32
Packaging web applications	33
Disabling dynamic JSP compilation	35
Precompiling JSPs	35
Packaging Enterprise JavaBeans	37
Packaging enterprise resource adapters	39
Packaging enterprise applications	40

CHAPTER 3 Deploying J2EE Modules	43
Module deployment overview	44
Dynamic module deployment	45
Controlling dynamic deployment	46
Deploying modules for development	46
Deploying modules for production	47
Undeploying and redeploying modules	47
Defining users, groups, and roles for authentication	47
Working with JRun-specific deployment descriptors	48
Generating a JRun-specific deployment descriptor	48
Using a JRun-specific deployment descriptor outside an archived module	48
 INDEX	 49

ABOUT THIS BOOK

The *JRun Assembly and Deployment Guide* describes how to configure, package, and deploy J2EE modules in JRun. It is intended for individuals performing any of these roles:

- **Developer** Creates modules and initial versions of the deployment descriptors that describe how modules should be deployed.
- **Assembler** Combines one or more modules into a deployable enterprise application.
- **Deployer** Maps deployment descriptor references to corresponding entities in the target JRun server environment.
- **Administrator** Configures the system resources that modules depend on.

Contents

- Developer resources vi
- About JRun documentation..... vii
- Other resources..... viii
- Contacting Macromedia xii

Developer resources

Macromedia, Inc. is committed to setting the standard for customer support in developer education, documentation, technical support, and professional services. The Macromedia website is designed to give you quick access to the entire range of online resources. The following table shows the locations of these resources:

Resource	Description
Information on JRun http://www.macromedia.com/products/jrun/	Detailed product information on JRun and related topics.
JRun Online Forum http://webforums.macromedia.com/jrun	Access to experienced JRun developers through participation in the Macromedia Online Forums, where you can post messages and read replies on many subjects relating to JRun.
Developer Resources http://www.macromedia.com/desdev/developer/	All of the resources that you need to stay on the cutting edge of JRun development, including online discussion groups, Component Exchange, Resource Library, technical papers, and more.
JRun Support Center http://www.macromedia.com/support/jrun	Professional support programs that Macromedia offers.
Training http://www.macromedia.com/support/training/	Information about classes, on-site training, and online courses offered by Macromedia.
Macromedia Alliance http://www.macromedia.com/partners/	Connection with the growing network of solution providers, application developers, resellers, and hosting services creating solutions with JRun.

About JRun documentation

JRun documentation provides support for all JRun users, including JSP developers, servlet developers, EJB client developers, EJB bean developers, and system administrators. The printed and online versions are organized to let you quickly locate the information that you need. JRun online documentation is provided in HTML and Adobe Acrobat formats.

Printed and online documentation set

The JRun documentation set consists of the following titles:

Book	Description
<i>Installing JRun</i>	Describes installing and configuring JRun.
<i>Getting Started with JRun</i>	Provides a J2EE overview, concepts, and tutorials for JSPs, servlets, EJBs, and web services.
<i>JRun Administrator's Guide</i>	Describes how to integrate a JRun server into an existing environment.
<i>JRun Programmer's Guide</i>	Describes how to use JRun to develop JSPs, servlets, custom tags, EJBs, and web services.
<i>JRun Assembly and Deployment Guide</i>	Describes how to assemble and deploy the components of a J2EE application.
<i>JRun SDK Guide</i>	Provides information to OEM/ISV customers and advanced users who embed, customize, or use the APIs in JRun.
<i>JSP Quick Reference</i>	Provides brief descriptions and syntax for JavaServer Pages (JSP) directives, actions, and scripting elements.
Online Help	Provides JMC users with usage notes, procedures, and concepts.

Accessing online documentation

All JRun documentation is available online in HTML and Adobe Acrobat formats. To access the documentation, open the following file on the server running JRun: *JRun_root/docs/dochome.htm*. *JRun_root* is the directory into which you installed JRun.

Macromedia provides online versions of all JRun books as Adobe Acrobat Portable Document Format (PDF) files. The PDF files are included on the JRun CD and installed in the JRun /docs directory, although they are an optional part of the installation. You can access them by clicking the Product Documentation link on the JRun Management Console Welcome window.

Other resources

You can consult the following resources for more information on topics described in JRun documentation.

Books

Servlets, JavaServer Pages, and Tag Libraries	
<i>JRun Web Application Construction Kit</i>	Drew Falkman Macromedia Press, 2001 ISBN: 0789726009
<i>Java Server Pages Application Development</i>	Scott M. Stirling, et al. Sams, 2000 ISBN: 067231939X
<i>More Servlets and JavaServer Pages</i>	Marty Hall Prentice Hall PTR, 2001 ISBN: 0130676144
<i>Core Servlets and JavaServer Pages</i>	Marty Hall Prentice Hall PTR, 2000 ISBN: 0130893404
<i>Java Servlet Programming, Second Edition</i>	Jason Hunter and William Crawford O'Reilly & Associates, 2001 ISBN: 0596000405
<i>Java Servlets Developer's Guide</i>	Karl Moss McGraw-Hill/Osborne Media, 2002 ISBN: 0-07-222262-X
<i>Inside Servlets: Server-Side Programming for the Java Platform, Second Edition</i>	Dustin R. Callaway Addison-Wesley, 2001 ISBN: 0201709066
<i>Web Development with JavaServer Pages</i>	Duane K. Fields and Mark A. Kolb Manning Publications Company, 2000 ISBN: 1884777996
<i>Enterprise Java Servlets</i>	Jeff Genender Addison-Wesley, 2001 ISBN: 020170921X
<i>Advanced JavaServer Pages</i>	David Geary Prentice Hall, 2001 ISBN: 0130307041

<i>JavaServer Pages</i>	Hans Bergsten O'Reilly & Associates, 2000 ISBN: 156592746X
<i>JSP Tag Libraries</i>	Gal Schachor, Adam Chace, and Magnus Rydin Manning Publications Company, 2001 ISBN: 193011009X
<i>Core JSP</i>	Damon Hougland and Aaron Tavistock Prentice Hall, 2000 ISBN: 0130882488
<i>JSP: Javaserer Pages (Developer's Guide)</i>	Barry Burd Hungry Minds Inc., 2001 ISBN: 0764535358
Enterprise JavaBeans	
<i>Mastering Enterprise JavaBeans, Second Edition</i>	Ed Roman John Wiley & Sons, 2002 ISBN: 0471417114
<i>Enterprise JavaBeans, Third Edition</i>	Richard Monson-Haefel O'Reilly & Associates, 2001 ISBN: 0596002262.
<i>Professional EJB</i>	Rahim Adatia, et al Wrox Press, 2001 ISBN: 1861005083
<i>Special Edition Using Enterprise JavaBeans (EJB) 2.0</i>	Chuck Cavaness and Brian Keeton Que, 2001 ISBN: 0789725673
<i>Applying Enterprise JavaBeans: Component-Based Development for the J2EE Platform</i>	Vlada Matena and Beth Stearns Addison-Wesley Pub Co, 2000 ISBN: 0201702673
Enterprise Java Programming	
<i>Professional Java Server Programming J2EE 1.3 Edition</i>	Subrahmanyam Allamaraju, et al Wrox Press, 2001 ISBN: 1861005377
<i>Server-Based Java Programming</i>	Ted Neward Manning Publications Company, 2000 ISBN: 1884777716

<i>Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition</i>	Nicholas Kassem Addison-Wesley, 2000 ISBN: 0201702770 (free download at java.sun.com/j2ee/download.html#blueprints)
<i>Building Java Enterprise Systems with J2EE</i>	Paul Perrone and Venkata S.R. "Krishna" .R. Chaganti Sams, 2000 ISBN: 0672317958
<i>J2EE: A Bird's Eye View</i> (e-book)	Rick Grehan Fawcette Technical Publications, 2001 ISBN: B00005BAZV
<i>Java Message Service</i>	Richard Monson-Haefel and David Chappell O'Reilly and Associates, 2001 ISBN: 0596000685
<i>J2EE Connector Architecture and Enterprise Application Integration</i>	Rahul Sharma, et al Addison-Wesley, 2001 ISBN: 0201775808
<i>Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI</i>	Sim Simeonov, Glen Daniels, et al Prentice Hall, 2002 ISBN: 0672321815
<i>Architecting Web Services</i>	William L. Oellermann Jr. Apress, 2001 ISBN: 1893115585

Online resources

Java servlet API	http://java.sun.com/products/servlet
JavaServer Pages API	http://java.sun.com/products/jsp
Enterprise JavaBeans API	http://java.sun.com/products/ejb/
Java 2 Standard Edition API	http://java.sun.com/j2se/
Servlet Source	http://www.servletsource.com
JSP Resource Index	http://www.jspin.com
Server Side	http://www.theserverside.com
Dot Com Builder	http://dcb.sun.com
Servlet Forum	http://www.servletforum.com

Contacting Macromedia

Corporate headquarters	Macromedia, Inc. 600 Townsend Street San Francisco, CA 94103 Tel: 415.252.2000 Fax: 415.626.0554 Web: http://www.macromedia.com
Technical support	Macromedia offers a range of telephone and web-based support options. Go to http://www.macromedia.com/support/ for a complete description of technical support services. You can make postings to the JRun Support Forum (http://webforums.macromedia.com) at any time.
Sales	Toll Free: 888.939.2545 Tel: 617.219.2100 Fax: 617.219.2101 E-mail: sales@macromedia.com Web: http://commerce.macromedia.com/purchase/index.cfm
OEM/hosting sales	For information about OEM/hosting, such as inquiries regarding licensing and pricing for product, support, training and consulting, contact: Toll Free: 888.939.2545, request OEM Sales Desk extension 2157. OEM Sales Desk: 617.219.2157 Fax: 617.219.2102 E-mail: oemsales@macromedia.com Web: http://www.macromedia.com/software/jrun/oem

CHAPTER 1

Configuring J2EE Modules

This chapter provides information about configuring J2EE modules for deployment in JRun.

Contents

- Understanding JRun deployment descriptors..... 2
- Understanding J2EE roles in module configuration..... 3
- Configuring web applications 4
- Configuring Enterprise JavaBeans..... 9
- Configuring enterprise resource adapters 24
- Configuring enterprise applications 28

Understanding JRun deployment descriptors

You use deployment descriptors to configure many J2EE modules without modifying Java code. A **deployment descriptor** is an XML file that describes how to deploy a module into a JRun server and lists the resources on which the module depends. Each type of J2EE module has a standard deployment descriptor based on a DTD (Document Type Description) for that type of module. In addition to elements specific to each type of module, the standard deployment descriptors for web applications and EJBs have common elements for environment entries, EJB references, resource references, and security roles. With JRun-specific deployment descriptors for web applications, EJBs, resource adapters, and J2EE client applications, you can configure features that are unique to JRun. For example, you can enable or disable dynamic servlet compilation and reloading in a JRun-specific web application deployment descriptor, `jrun-web.xml`. JRun-specific deployment descriptor names are made up of the names of the corresponding standard deployment descriptors preceded by the prefix `jrun-`.

Note: You can deploy JRun 3.x enterprise applications, EJBs, and web applications without changing their deployment descriptors.

You can also deploy enterprise applications, web applications, and enterprise resource adapters packaged for the Sun Reference Implementation (RI), even if those modules contain RI-specific deployment descriptors. You can use the RI deployment tool for modules you plan to deploy on JRun. However, JRun offers more advanced support for container-managed persistence than you can configure for the RI.

JRun uses the following deployment descriptors. Each of these files is discussed in the following sections of this chapter. For more information, see the deployment descriptor documentation listed in the `jrun_root/docs/descriptor/docs/index.html` file.

Module type	Deployment descriptors
Web application	<code>web_app/WEB-INF/web.xml</code> , <code>jrun-web.xml</code> Note: JRun also uses a deployment descriptor called <code>default-web.xml</code> , which is located in the <code>jrun_root/server/jrun_server/SERVER-INF</code> directory. This file has the same structure as the standard <code>web.xml</code> file, and lets you apply configuration settings globally to all the web applications in a JRun server. Entries in a web application's <code>web.xml</code> file override those in the <code>default-web.xml</code> file.
Enterprise JavaBean	<code>ejb_module/META-INF/ejb-jar.xml</code> , <code>jrun-ejb-jar.xml</code>
Enterprise resource adapter	<code>adapter_module/META-INF/ra.xml</code> , <code>jrun-ra.xml</code>
Enterprise application	<code>enterprise_app/META-INF/application.xml</code>

Note: The JRun installation includes **XDoclet**, an open source tool that you can use to generate EJB interfaces and deployment descriptors based on special Javadoc comments in a bean implementation source file. You can also use XDoclet to generate web application deployment descriptors and JSP tag library descriptors based on comments in servlet or tag library source files. JRun extends XDoclet to support the JRun-specific deployment descriptors, and provides automatic compilation. For more information, see *JRun Programmer's Guide* and the XDoclet website at <http://xdoclet.sourceforge.net>.

Understanding J2EE roles in module configuration

Before deploying a J2EE module on a JRun server, you must appropriately configure the module for the target environment. During configuration, you might assume the roles of developer, assembler, deployer, and system administrator, or you might share these roles with other people.

- The module **developer** creates the module and the initial versions of the deployment descriptors. Some information in the deployment descriptors will change when the module moves from a development environment to an application in a production environment.

For example, the production environment could require changes to the data source mappings, security role references, and the JNDI names of EJBs in an EJB or web application deployment descriptor. The developer must include the appropriate `resource-ref`, `security-role-ref`, and `ejb-ref` elements in the deployment descriptor to make these changes possible.

- The **assembler** combines one or more modules into a deployable enterprise application. The assembler creates the enterprise application deployment descriptor, and identifies the system resources and class libraries on which the application depends. In some situations, multiple modules in an enterprise application must access the same class libraries.

The assembler also modifies the deployment descriptors of the individual modules to accommodate application-wide resource references, security roles, and EJB references. For EJBs, the assembler might set method permissions or transaction attributes in EJB deployment descriptors.

- The **deployer** maps resource, EJB, and security-role references to corresponding entities in the target JRun server environment by editing module deployment descriptors. For CMP beans, the deployer also maps CMP settings to actual fields in a database.
- The **system administrator** configures the JRun system resources on which modules depend, such as security roles and users, JDBC data sources, JMS resources, and JavaMail sessions.

Configuring web applications

This section discusses the standard web application deployment descriptor, `web.xml`, and the JRun-specific deployment descriptor, `jrun-web.xml`. For more information, see the deployment descriptor documentation listed in the `jrun_root/docs/descriptordocs/index.html` file.

Many of the settings in the deployment descriptors directly correspond to the servlets and JSPs in a web application. For information about programming servlets and JSPs, see *JRun Programmer's Guide* and the resources listed in “Developer resources” on page vi.

Understanding the `web.xml` deployment descriptor

You use the standard web application deployment descriptor, `web.xml`, to configure the elements of a web application listed in the following table. A valid `web.xml` file does not require any child elements, so the elements you include depend on what is in your web application. The root element of the file is `web-app`.

XML element	Description
icon, display-name, description	Image(s), web application name, and description for use by tools.
context-param (zero or more)	Servlet context initialization parameters.
filter (zero or more)	Servlet filter definitions.
filter-mapping (zero or more)	Servlet filter mapping to a servlet name or URL pattern.
listener (zero or more)	Deployment properties for a web application listener bean.
servlet (zero or more)	Servlet definition.
servlet-mapping (zero or more)	Servlet mapping to a URL pattern.
session-config	Web application session parameter.
mime-mapping (zero or more)	A mapping between a file extension and a mime type.
welcome-file-list	Ordered list of welcome file elements.
error-page (zero or more)	Mapping between an error code or exception type to the path of a resource in the web application.
taglib (zero or more)	JSP tag library definition.
resource-env-ref (zero or more)	Web application's reference to an administered object in a JRun server environment.
resource-ref (zero or more)	Web application's reference to an external resource.
security-constraint (zero or more)	Association of security constraints with one or more web resource collections.
login-config	Declaration of the authentication method, realm name, and attributes needed by the form login mechanism.
security-role (zero or more)	Declaration of a security role.
env-entry (zero or more)	A web application's environment entry.

XML element	Description
ejb-ref (zero or more)	<p>Reference in servlet, JSP, or tag library code to an EJB's home.</p> <p>In the role of assembler, you can include an optional <code>ejb-link</code> element to specify that an EJB reference is linked to an EJB in the same enterprise application as the web application. You must precede the <code>ejb-name</code> value with the relative path to the EJB module followed by a pound sign (#).</p> <p>If the EJB's <code>jrnl-ejb-jar.xml</code> file changes its default JNDI name in an <code>ejb-ref</code> element, or the EJB is not in the same enterprise application as the web application, the <code>jrnl-web.xml</code> file must contain a corresponding <code>ejb-local-ref</code> element that contains its JNDI name.</p>
ejb-local-ref (zero or more)	<p>Reference in servlet or JSP code to an EJB's home.</p> <p>In the role of assembler, you can include an optional <code>ejb-link</code> element to specify that an EJB local reference is linked to an EJB in the same enterprise application. If the EJB is in a different EJB module, you must precede the <code>ejb-name</code> value with the relative path to the EJB module and a pound sign (#).</p> <p>If the EJB's <code>jrnl-ejb-jar.xml</code> file changes the default JNDI name of its local home in an <code>ejb-local-ref</code> element, or the EJB is not in the same enterprise application as the web application, the <code>jrnl-web.xml</code> file must contain a corresponding <code>ejb-local-ref</code> element that contains its JNDI name.</p>

Examples: web.xml deployment descriptors

The examples in this section show the following `web.xml` files:

- An empty `web.xml` file
- A `web.xml` file that configures a servlet and welcome file
- A `web.xml` file that configures security

You can also view the `web.xml` files of the applications on the JRun samples server in the `jrnl_root/servers/samples` directory.

web.xml: empty file

The following empty deployment descriptor is the most basic one that you can use. You might deploy a web application containing JSPs and no servlets (`web-app`) with this descriptor in the earliest stages of development, and add to the descriptor as the complexity of your application increases. The default-`web.xml` file in the `jrnl_root/servers/jrnl_server/SERVER-INF` directory sets the default welcome files as `index.html` and `index.jsp`.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.3.dtd">
<web-app>
</web-app>

```

Note: The JRun default server and each JRun server you add using the JMC have default-ear enterprise applications containing default-war web applications with simple web.xml files. To quickly deploy and test new JSPs and servlets, you can put them in a default-war web application.

web.xml: servlet and welcome file configuration

The following deployment descriptor configures a servlet, maps the servlet to a URL pattern, and specifies a welcome file for the web application:

```

?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <servlet>
    <servlet-name>AdminServlet</servlet-name>
    <display-name>AdminServlet</display-name>
    <servlet-class>packagename.Administrator</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>AdminServlet</servlet-name>
    <url-pattern>/admin</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>

```

web.xml: security configuration

The following deployment descriptor configures these elements of web application security:

- `web-resource-collection` specifies the secured resource.
- `auth-constraint` specifies the JRun security role that can access the resource.
- `login-config` specifies the authentication mechanism.
- `security-role` references a JRun security role.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.3.dtd">
<web-app>
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Name1</web-resource-name>
    <url-pattern>/services/AdminService</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>

```

```

        <role-name>Manager</role-name>
    </auth-constraint>
</security-constraint>
<login-config>
    <auth-method>BASIC</auth-method>
</login-config>
<security-role>
    <role-name>Manager</role-name>
</security-role>
</web-app>

```

Understanding the jrun-web.xml deployment descriptor

You use the JRun-specific web application deployment descriptor, `jrun-web.xml`, to configure the elements of a web application listed in the following table. The root element of the file is `jrun-web-app`. A `jrun-web.xml` file is not needed if your web application does not require any of these elements.

Note: You can automatically generate a `jrun-web.xml` file. For more information, see “Working with JRun-specific deployment descriptors,” in Chapter 3.

XML element	Description
<code>context-root</code>	Declaration of the context root (logical root) of a web application. When a web application is in an enterprise application, you must set the <code>context-root</code> in the enterprise application’s <code>application.xml</code> file rather than in the <code>jrun-web.xml</code> file.
<code>reload</code>	Boolean setting that specifies whether to automatically reload servlets, servlet helper classes, and JSP helper classes when they change. The default value is <code>true</code> .
<code>compile</code>	Whether to automatically compile servlets and servlet helper classes when their source files change. The default value is <code>true</code> .
<code>session-config</code>	Contains the following sections for configuring web application sessions: <ul style="list-style-type: none"> • <code>persistence-config</code> contains elements for configuring session persistence properties. • <code>replication-config</code> contains elements for configuring session replication. • <code>cookie-config</code> contains elements configuring cookies.
<code>load-system-classes-first</code>	Whether system classes should be loaded before enterprise and web application classes. The default classloader delegation model loads the system classes first; however the servlet specification suggests that you load web application classes first. The default value is <code>true</code> .

XML element	Description
ejb-ref	<p>A mapping between the ejb-ref name that the EJB developer provides and its JNDI name. The deployer provides the actual JNDI name.</p> <p>This element is only required to bind an EJB class into JNDI more than once.</p>
ejb-local-ref	<p>A mapping between the local home's ejb-ref name that the EJB developer provides and its JNDI name. The deployer provides the actual JNDI name.</p> <p>This element is only required to bind an EJB class into JNDI more than once.</p>
resource-env-ref	<p>A mapping between the resource-env-ref-name that the web application developer provides and its JNDI name. The deployer provides the JNDI name.</p> <p>This element is only required to bind an administered object into JNDI more than once.</p>
resource-ref	<p>A mapping between the resource-ref-name that the web application developer provides and its JNDI name. The deployer provides the actual JNDI name.</p> <p>This element is only required to bind a resource into JNDI more than once.</p>
virtual-mapping	<p>A mapping between a resource path and a physical location not necessarily within the web application root. A resource path must always begin with a slash, and can end with a wildcard (*), indicating that all resources paths that start with the given path will be resolved using the system path.</p> <p>If you map /WEB-INF/classes or /WEB-INF/lib to another location, the web application classpath includes that location.</p> <p>The use-web-server-root property used in JRun 3.1 does not exist in JRun 4. The virtual-mapping property replaces it. You must manually replace use-web-server-root settings with virtual-mapping settings.</p>

Example: jrun-web.xml deployment descriptor

The following deployment descriptor sets a context root, enables dynamic servlet reloading and compiling, and defines two virtual path mappings:

```
<?xml version="1.0"?>
<!DOCTYPE jrun-web-app SYSTEM
    "jrun-web.dtd">
<jrun-web-app>
  <context-root>myapp</context-root>
  <reload>true</reload>
  <compile>true</compile>
  <virtual-mapping>
    <resource-path>/images/*</resource-path>
    <system-path>usr/local/images</system-path>
  </virtual-mapping>
```

```
<virtual-mapping>
  <resource-path>/WEB-INF/classes</resource-path>
  <system-path>d:/app1/library</system-path>
</virtual-mapping>
</jrun-web-app>
```

Configuring Enterprise JavaBeans

This section discusses the standard EJB deployment descriptor, `ejb-jar.xml`, and the JRun-specific deployment descriptor, `jrun-ejb-jar.xml`. For more information, see the deployment descriptor documentation listed in the `jrun_root/docs/descriptor/docs/index.html` file.

For information about EJB programming, see *JRun Programmer's Guide* and the resources listed in "About This Book" on page v.

JRun provides the following tools that let you develop, package, and deploy EJBs without working directly in the deployment descriptor files:

- **Enterprise Deployment Wizard** Helps you with the entire development, assembly, and deployment process, including configuration of container-managed persistence for CMP entity beans. It is also ideal for creating deployment descriptors for existing EJB code. You can use the Enterprise Deployment Wizard as a stand-alone application or with Borland JBuilder or Forte for Java. For more information, see *JRun Programmer's Guide*.
- **XDoclet** is an open source tool that generates EJB interfaces and deployment descriptors based on special Javadoc comments in a bean implementation source file. The JRun installation includes XDoclet. JRun extends XDoclet to support JRun-specific deployment descriptors, and provides automatic compilation. For more information, see *JRun Programmer's Guide* and the XDoclet website at <http://xdoclet.sourceforge.net>.

Understanding the ejb-jar.xml deployment descriptor

You use the standard EJB deployment descriptor, `ejb-jar.xml`, to configure the elements of an EJB module listed in the following table. A valid `ejb-jar.xml` file only requires a valid `enterprise-beans` section, in which the bean developer declares one or more beans. The assembler works with the `enterprise-beans` section and the `assembly-descriptor` section. The root element of the file is `ejb-jar`.

XML element	Required/ Optional	Description
<code>description</code> , <code>display-name</code> , <code>small icon</code> , <code>large icon</code>	Optional	Description, name, and images for use by tools.
<code>enterprise-beans</code>	Required	<p>Declaration of one or more session, entity, or message-driven beans. The bean developer provides the initial values.</p> <p>The assembler can modify or add the following information when adding an EJB module to an enterprise application:</p> <ul style="list-style-type: none">• Resource references• Security role references that can contain role-links between the security role references the EJB declares and the security roles the assembler defines• EJB references that can contain <code>ejb-links</code> to specific beans in an EJB module
<code>relationships</code>	Optional	Description of the relationships in which entity beans with container-managed persistence (CMP) 2.0 participate.
<code>assembly-descriptor</code>	Optional	Assembler's description of how multiple EJBs are composed into an application unit. Includes transaction attributes, method permissions, security roles, and excluded methods.
<code>ejb-client-jar</code>	Optional	Declaration of a JAR file containing the classes a client requires to access the EJB(s).

Understanding the enterprise-beans section

The enterprise-beans section of an `ejb-jar.xml` file is the only required section. This is where the bean developer declares the beans that are part of the application.

Common elements

The following table describes only those elements under the `ejb-jar/enterprise-beans` elements that are common to session, entity, and message-driven beans:

XML element	Required/ Optional	Description
session, entity, or message-driven (parent element)	Required	Type of EJB.
description, display-name, small icon, large icon	Optional	Description, name, and images for use by tools.
ejb-name	Required	Name identifying the bean. If a JNDI name is not specified in a corresponding <code>jni-name</code> element in the <code>jrnl-ejb-jar.xml</code> file, JRun uses the <code>ejb-name</code> as the JNDI name.
You must include <code>home</code> and <code>remote</code> , <code>local-home</code> and <code>local</code> , or all four of the following elements:		
home		Fully qualified name of the EJB's home interface.
local-home		Fully qualified name of the EJB's local home interface.
remote		Fully qualified name of the EJB's remote interface.
local		Fully qualified name of the EJB's local interface.
ejb-class	Required	Fully qualified name of the EJB's implementation class.
env-entry (zero or more)	Optional	EJB's environment entry.
ejb-ref (zero or more)	Optional	Reference in EJB code to another EJB's home, which allows bean-to-bean method calls. In the role of assembler, you can include an optional <code>ejb-link</code> element to specify that an EJB reference is linked to an EJB in the same enterprise application. If the EJB is in a different EJB module, you must precede the <code>ejb-name</code> value with the relative path to the EJB module followed by a pound sign (<code>#</code>). If the other EJB's <code>jrnl-ejb-jar.xml</code> file changes its default JNDI name in an <code>ejb-ref</code> element, or the EJB is not in the same EJB module or enterprise application, this EJB's <code>jrnl-ejb-jar.xml</code> file must contain a corresponding <code>ejb-local-ref</code> element that contains its JNDI name.

XML element	Required/ Optional	Description
ejb-local-ref (zero or more)	Optional	Reference in EJB code to another EJB's local home, which allows bean-to-bean method calls. In the role of assembler, you can include an optional ejb-link element to specify that an EJB local reference is linked to an EJB in the same enterprise application. If the EJB is in a different EJB module, you must precede the ejb-name value with the relative path to the EJB module and a pound sign (#). If the other EJB's jrun-ejb-jar.xml file changes the default JNDI name of its local home in an ejb-local-ref element, or the EJB is not in the same EJB module or enterprise application, this EJB's jrun-ejb-jar.xml file must contain a corresponding ejb-local-ref element that contains its JNDI name.
security-role-ref (zero or more) (not message-driven beans)	Optional	Reference in EJB code to a security role.
security-identity	Optional	Whether to use the caller's associated security role or a specific run-as security role for execution of the EJB's methods. Contains either an empty use-caller-identity element or a run-as element that contains description (optional) and role-name elements.
resource-ref (zero or more)	Optional	Reference in EJB code to an external resource.
resource-env-ref (zero or more)	Optional	Reference in EJB code to an administered object associated with an external resource.

Required session bean elements

In addition to the elements required for all types of beans, session beans require the following elements under the ejb-jar/enterprise-beans/session elements:

XML element	Description
session-type	Whether the session bean is a stateful session or stateless session bean; values are Stateful and Stateless.
transaction-type	Whether the EJB uses bean-managed or container-managed transactions; values are Bean and Container.

Required entity bean elements

In addition to the elements required for all types of beans, entity beans require the following elements under the `ejb-jar/enterprise-beans/entity` elements.

XML element	Description
<code>persistence-type</code>	Whether persistence is container-managed or bean-managed; the values are <code>Container</code> and <code>Bean</code> .
<code>prim-key-class</code>	Fully qualified name of an entity bean's primary-key class. If the bean developer defers the definition of the primary key class until deployment time, set it to <code>java.lang.Object</code> .
<code>reentrant</code>	Boolean value specifying whether the bean is reentrant, which means it can invoke methods on itself or call another bean that invokes a method on it.
<code>abstract-schema-name</code>	Name of the abstract schema type of an entity bean using <code>cmp-version 2.x</code> and EJB Query Language (EJB QL) queries.

Required message-driven bean elements

In addition to the elements required for all types of beans, message-driven beans require the following element under the `ejb-jar/enterprise-beans/message-driven` elements:

XML element	Description
<code>transaction-type</code>	Whether the EJB uses bean-managed or container-managed transactions; values are <code>Bean</code> and <code>Container</code> .

Understanding the assembly-descriptor section

In the `assembly-descriptor` section of an `ejb-jar.xml` file, the assembler defines security roles, method permissions, transaction attributes for EJBs that use container-managed transactions, and methods to exclude from deployment. All of the top-level assembly descriptor elements are optional and are described in the following table:

XML element	Description
<code>security-role</code> (zero or more)	Declaration of a security role. Must match a security role defined in the JRun environment; for more information, see JRun Administrator's Guide.
<code>method-permission</code> (zero or more)	Specification that one or more security roles can invoke one or more enterprise bean methods.

XML element	Description
container-transaction (zero or more)	How the container must manage transaction scopes for the EJB's method invocations. Specify one of the following transaction attributes in a trans-attribute element: <ul style="list-style-type: none"> • NotSupported • Supports • Required • RequiresNew • Mandatory • Never
exclude-list	Set of methods marked as uncallable. Methods in the exclude list are not callable.

Examples: ejb-jar.xml deployment descriptors

The following deployment descriptors provides configuration information for a stateless session bean, a BMP entity bean, a CMP 1.1 entity bean, and a CMP 2.0 entity bean. You can also view the `ejb-jar.xml` files of the applications on the JRun samples server in the `jrun_root/servers/samples` directory.

ejb-jar.xml: stateless session bean declaration, assembly descriptor

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//
    DTD EnterpriseJavaBeans 2.0//EN" "http://java.sun.com/dtd/
    ejb-jar_2_0.dtd">
<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>CreditCard</ejb-name>
      <home>compass.CreditCardHomeRemote</home>
      <remote>compass.CreditCardRemote</remote>
      <ejb-class>compass.CreditCardBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
      <security-identity><use-caller-identity/></security-identity>
    </session>
  </enterprise-beans>
  <assembly-descriptor>
    <security-role>
      <role-name>everyone</role-name>
    </security-role>
    <method-permission>
      <role-name>everyone</role-name>
      <method>
        <ejb-name>CreditCard</ejb-name>
        <method-name>*</method-name>
      </method>
    </method-permission>
    <container-transaction>
      <method>
        <ejb-name>CreditCard</ejb-name>
```

```

        <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
</container-transaction>
</assembly-descriptor>
</ejb-jar>

```

ejb-jar.xml: BMP entity bean declaration, assembly descriptor

```

<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//
    DTD EnterpriseJavaBeans 2.0//EN" "http://java.sun.com/dtd/
    ejb-jar_2_0.dtd">
<ejb-jar>
    <enterprise-beans>
        <entity>
            <ejb-name>Order</ejb-name>
            <home>compass.OrderHomeRemote</home>
            <remote>compass.OrderRemote</remote>
            <ejb-class>compass.OrderBean</ejb-class>
            <persistence-type>Bean</persistence-type>
            <prim-key-class>java.lang.Integer</prim-key-class>
            <reentrant>False</reentrant>
            <security-identity><use-caller-identity/></security-identity>
        </entity>
    </enterprise-beans>
    <assembly-descriptor>
        <security-role>
            <role-name>everyone</role-name>
        </security-role>
        <method-permission>
            <role-name>everyone</role-name>
            <method>
                <ejb-name>Order</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>
        <container-transaction>
            <method>
                <ejb-name>Order</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>

```

ejb-jar.xml: CMP 1.1 entity bean declaration

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0/
    /EN" "http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd">
<ejb-jar>

    <enterprise-beans>
        <entity>

```

```

    <ejb-name>Employee</ejb-name>
    <ejb-class>samples.cmp11.EmployeeBean</ejb-class>
    <home>samples.cmp11.EmployeeHome</home>
    <remote>samples.cmp11.Employee</remote>
    <persistence-type>Container</persistence-type>
    <primkey-field>employeeId</primkey-field>
    <prim-key-class>java.lang.String</prim-key-class>
    <reentrant>True</reentrant>
    <cmp-version>1.x</cmp-version>
    <cmp-field>
      <field-name>employeeId</field-name>
    </cmp-field>
    <cmp-field>
      <field-name>firstName</field-name>
    </cmp-field>
    <cmp-field>
      <field-name>lastName</field-name>
    </cmp-field>
    <cmp-field>
      <field-name>phone</field-name>
    </cmp-field>
  </entity>
</enterprise-beans>
</ejb-jar>

```

ejb-jar.xml: CMP 2.0 entity bean declaration

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0/
  /EN" "http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd">
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>samples/cmp20/Employee</ejb-name>
      <ejb-class>samples.cmp20.EmployeeBean</ejb-class>
      <home>samples.cmp20.EmployeeHome</home>
      <remote>samples.cmp20.Employee</remote>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.String</prim-key-class>
      <reentrant>True</reentrant>
      <primkey-field>employeeId</primkey-field>
      <abstract-schema-name>employeeschema</abstract-schema-name>
      <cmp-version>2.x</cmp-version>
      <cmp-field>
        <field-name>employeeId</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>firstName</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>lastName</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>phone</field-name>
      </cmp-field>
      <query>

```

```

        <query-method>
            <method-name />
            <method-params />
        </query-method>
        <return-type-mapping>Local</return-type-mapping>
        <ejb-ql>SELECT OBJECT(o) FROM employeeschema AS o</ejb-ql>
    </query>
    <query>
        <query-method>
            <method-name />
            <method-params />
        </query-method>
        <return-type-mapping>Local</return-type-mapping>
        <ejb-ql>SELECT OBJECT(o) FROM employeeschema AS o WHERE o.lastName =
            ?1</ejb-ql>
    </query>
</entity>
</enterprise-beans>
</ejb-jar>

```

Understanding the jrun-ejb-jar.xml deployment descriptor

The JRun-specific EJB deployment descriptor, `jrun-ejb-jar.xml`, is not required in all situations. You use it to configure elements specific to JRun and not included in the `ejb-jar.xml` file, such as JNDI names, resource mappings, and JDBC mappings that specify the SQL used to create, load, store, find, and remove CMP 1.1 entity beans.

Note: You can automatically generate an `jrun-ejb-jar.xml` file. For more information, see “Working with JRun-specific deployment descriptors,” in Chapter 3.

The `jrun-ejb-jar.xml` file contains the following top-level elements under `jrun-ejb-jar`:

XML element	Description
<code>description</code>	Description of the EJB module.
<code>source</code>	JNDI name of the data source to be used to deploy the bean(s) or execute a SQL statement.
<code>enterprise-beans</code>	Elements corresponding to the <code>enterprise-beans</code> section in the <code>ejb-jar.xml</code> file. Use this section to configure JNDI mappings, resource mappings, EJB clustering, stateful session bean timeout values, and stateless session bean instance pool sizes.

Understanding the enterprise-beans section

This section describes the subelements of enterprise beans.

Common elements

The following table describes the elements under the `run-ejb-jar/enterprise-beans` elements that are common to session, entity, and message-driven beans; only the `ejb-name` and `jndi-name` elements are required:

XML element	Description
session, entity, or message-driven (parent element)	Type of EJB.
ejb-name	Name that identifies the EJB.
jndi-name	JNDI name of the EJB. If you do not specify a JNDI name here, JRun uses the <code>ejb-name</code> as the JNDI name.
tx-domain-name	Name of the transaction domain in which EJB transactions will take place.
ejb-ref	A mapping between the <code>ejb-ref-name</code> the bean developer provides and its JNDI name. The deployer provides the JNDI name. This element is required only to bind an EJB class into JNDI more than once.
ejb-local-ref	A mapping between the local home's <code>ejb-ref-name</code> provided by the EJB developer and its JNDI name. The deployer provides the actual JNDI name. This element is required only to bind an EJB class into JNDI more than once.
resource-env-ref	A mapping between the <code>resource-env-ref-name</code> that the bean developer provides and its JNDI name. The deployer provides the JNDI name. This element is required only to bind an administered object into JNDI more than once.
resource-ref	A mapping between the <code>resource-ref-name</code> that the bean developer provides and its JNDI name. The deployer provides the JNDI name. This element is required only to bind a resource into JNDI more than once.

XML element	Description
cluster-home	Whether the EJB home should be clustered for this bean. If clustering is enabled in the <i>jrun_root/servers/jrun_server/SERVER-INF/jrun.xml</i> file, this value is true by default. You can override this behavior on a bean-by-bean basis using this element.
cluster-object	Whether the EJB object should be clustered for this bean. If clustering is enabled in the <i>jrun_root/servers/jrun_server/SERVER-INF/jrun.xml</i> file, this value is true by default. You can override this behavior on a bean-by-bean basis using this element.

Session bean elements

In addition to the elements required for all types of beans, session bean configuration can include the following optional elements under the *jrun-ejb-jar/enterprise-beans/session* elements:

XML element	Description
timeout	Timeout value, in seconds, of a stateful session bean. The EJB instance is passivated (disassociated from its EJB object to conserve memory) if left idle for this period of time.
instance-pool	The maximum and minimum size parameters for stateless session bean instance pools.

Entity bean elements

In addition to the elements required for all types of beans, entity bean configuration can include the following optional elements under the *jrun-ejb-jar/enterprise-beans/entity* elements:

XML element	Description
commit-option	Commit option as described in the EJB 2.0 specification. Valid values are A, B, and C.
always-dirty	Empty XML element that forces synchronization with the data source at the ends of transactions, even when the entity bean's fields have not changed.
jdbc-mappings	Information specific to JRun and not declared in the <i>ejb-jar.xml</i> file about the CMP mappings for a CMP 1.1 entity bean. The contained <i>jdbc-mapping</i> elements specify the SQL used to create, load, store, find, and remove entity beans.

Message-driven bean elements

In addition to the elements required for all types of beans, message-driven bean configuration can include the following optional elements under the `jrun-ejb-jar/enterprise-beans/entity` elements:

XML element	Description
<code>message-driven-subscription</code>	Client ID to be used by the message-driven container for durable subscription handling
<code>message-driven-destination</code>	Destination the message-driven container uses

Examples: `jrun-ejb-jar.xml` deployment descriptors

The following deployment descriptors provide configuration information for a stateful session bean, a stateless session bean, and a CMP 1.1 entity bean. You can also view the `jrun-ejb-jar.xml` files of the applications on the JRun samples server in the `jrun_root/servers/samples` directory.

`jrun-ejb-jar.xml`: stateful session bean declaration

This example sets a stateful session bean's JNDI name, disables its home and object clustering, and sets its timeout value.

```
<?xml version="1.0"?>
<!DOCTYPE jrun-ejb-jar PUBLIC "-//Macromedia, Inc.//DTD jrun-ejb-jar 4.0//EN"
    'http://jrun.macromedia.com/dtds/jrun-ejb-jar.dtd'>
<jrun-ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>StatefulEJB</ejb-name>
      <jndi-name>StatefulEJB</jndi-name>
      <cluster-home>false</cluster-home>
      <cluster-object>false</cluster-object>
      <timeout>900</timeout>
    </session>
  </enterprise-beans>
</jrun-ejb-jar>
```

`jrun-ejb-jar.xml`: stateless session bean declaration

This example sets a stateless session bean's JNDI name and maximum and minimum instance pool sizes; also see the corresponding “`ejb-jar.xml`: stateless session bean declaration, assembly descriptor” on page 14.

```
<?xml version="1.0"?>
<!DOCTYPE jrun-ejb-jar PUBLIC "-//Macromedia, Inc.//DTD jrun-ejb-jar 4.0//EN"
    'http://jrun.macromedia.com/dtds/jrun-ejb-jar.dtd'>
<jrun-ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>CreditCard</ejb-name>
      <jndi-name>ejb/CreditCard</jndi-name>
      <cluster-home>False</cluster-home>
      <cluster-object>False</cluster-object>
```

```

    <instance-pool>
      <minimum-size>1</minimum-size>
      <maximum-size>5</maximum-size>
    </instance-pool>
  </session>
</enterprise-beans>
</jrun-ejb-jar>

```

jrun-ejb-jar.xml: CMP 1.1 entity bean declaration

This example sets a CMP 1.1 entity bean's JNDI name and JDBC mappings; also see the corresponding "ejb-jar.xml: CMP 1.1 entity bean declaration" on page 15.

```

<?xml version="1.0"?>
<!DOCTYPE jrun-ejb-jar PUBLIC "-//Macromedia, Inc.//DTD jrun-ejb-jar 4.0//EN"
    'http://jrun.macromedia.com/dtds/jrun-ejb-jar.dtd'>
<jrun-ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>Employee</ejb-name>
      <jndi-name>Employee</jndi-name>
      <jdbc-mappings>
        <jdbc-mapping>
          <name>create</name>
          <statement>
            <action>INSERT INTO employees (employee_id , first_name , last_name,
              phone) VALUES ( ? , ? , ? , ? )</action>
            <source>samples</source>
            <params>
              <param>
                <name>employeeId</name>
                <type>VARCHAR</type>
              </param>
              <param>
                <name>firstName</name>
                <type>VARCHAR</type>
              </param>
              <param>
                <name>lastName</name>
                <type>VARCHAR</type>
              </param>
              <param>
                <name>phone</name>
                <type>VARCHAR</type>
              </param>
            </params>
          </statement>
        </jdbc-mapping>
        <jdbc-mapping>
          <name>load</name>
          <statement>
            <action>SELECT employee_id, first_name, last_name, phone FROM employees
              WHERE employee_id=?</action>
            <source>samples</source>
            <params>
              <param>

```

```

        <name>employeeId</name>
        <type>VARCHAR</type>
    </param>
</params>
<fields>
    <field>employeeId</field>
    <field>firstName</field>
    <field>lastName</field>
    <field>phone</field>
</fields>
</statement>
</jdbc-mapping>
<jdbc-mapping>
    <name>remove</name>
    <statement>
        <action>DELETE FROM employees WHERE employee_id=?</action>
        <source>samples</source>
        <params>
            <param>
                <name>employeeId</name>
                <type>VARCHAR</type>
            </param>
        </params>
    </statement>
</jdbc-mapping>
<jdbc-mapping>
    <name>store</name>
    <statement>
        <action>UPDATE employees SET first_name=?, last_name=?, phone=? WHERE
            employee_id=?</action>
        <source>samples</source>
        <params>
            <param>
                <name>firstName</name>
                <type>VARCHAR</type>
            </param>
            <param>
                <name>lastName</name>
                <type>VARCHAR</type>
            </param>
            <param>
                <name>phone</name>
                <type>VARCHAR</type>
            </param>
            <param>
                <name>employeeId</name>
                <type>VARCHAR</type>
            </param>
        </params>
    </statement>
</jdbc-mapping>
<jdbc-mapping>
    <name>findAll</name>
    <statement>
        <action>SELECT employee_id FROM employees</action>
        <source>samples</source>

```

```

        <field>employeeId</field>
    </fields>
</statement>
</jdbc-mapping>
<jdbc-mapping>
    <name>findByLastName</name>
    <statement>
        <action>SELECT employee_id FROM employees WHERE last_name=?1</action>
        <source>samples</source>
        <params>
            <param>
                <name>lastName</name>
                <type>VARCHAR</type>
            </param>
        </params>
        <fields>
            <field>employeeId</field>
        </fields>
    </statement>
</jdbc-mapping>
<jdbc-mapping>
    <name>findByPrimaryKey</name>
    <statement>
        <action>SELECT employee_id FROM employees WHERE employee_id=?</action>
        <source>samples</source>
        <params>
            <param>
                <name>employeeId</name>
                <type>VARCHAR</type>
            </param>
        </params>
        <fields>
            <field>employeeId</field>
        </fields>
    </statement>
</jdbc-mapping>
</jdbc-mappings>
</entity>
</enterprise-beans>

```

Configuring enterprise resource adapters

This section discusses the standard resource adapter deployment descriptor, `ra.xml`, and the JRun-specific deployment descriptor, `jrun-ra.xml`. A `jrun-ra.xml` file is not required when there is a `display-name` element in the `ra.xml` file and only one managed connection factory instance is required. For more information, see the deployment descriptor documentation listed in the `jrun_root/docs/descriptor/docs/index.html` file.

For information about resource adapter programming, see *JRun Programmer's Guide* and the resources listed in "Developer resources" on page vi.

Understanding the `ra.xml` deployment descriptor

The resource adapter developer creates the `ra.xml` file, which contains information about resource adapter implementation code, configuration properties, and security information. The deployer modifies the `resourceadapter/config-properties` section of the file to set the configurable properties of the resource adapter(s). The deployer also configures JRun for transaction management based on the transaction management level defined in the `ra.xml` file, and sets up security based on the information in the `resourceadapter/authentication-mechanism` section of the file.

The `ra.xml` file contains the following top-level elements under the connector element:

XML element	Required/ Optional	Description
<code>display-name</code> , <code>description</code> , <code>icon</code>	Optional	Name, description, and images for use by tools.
<code>vendor-name</code>	Required	Name of resource adapter provider.
<code>spec-version</code>	Required	Version of the connector architecture specification supported by the resource adapter. Lets the deployer configure the resource adapter to support the deployment and runtime requirements of the specification.
<code>eis-type</code>	Required	Type of enterprise information system (EIS) for which the adapter is written.
<code>version</code>	Required	Version of the resource adapter from the resource adapter provider.

XML element	Required/ Optional	Description
license	Optional	Whether a license is required to deploy and use the resource adapter, and an optional description of the licensing terms.
resourceadapter	Required	Information about the resource adapter, including the following: <ul style="list-style-type: none"> Fully qualified names of the classes and interfaces required as part of the connector architecture contracts Level of transaction support Configurable properties for ManagedConnectionFactory instances One or more supported authentication mechanisms Additional required security permissions

Example: ra.xml deployment descriptor

The following ra.xml file, provided with the J2EE Reference Implementation, configures a resource adapter for a JDBC database:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE connector PUBLIC "-//Sun Microsystems, Inc.//DTD Connector 1.0//EN"
    'http://java.sun.com/dtd/connector_1_0.dtd'>
<connector>
  <display-name>BlackBoxNoTx</display-name>
  <vendor-name>Java Software</vendor-name>
  <spec-version>1.0</spec-version>
  <eis-type>JDBC Database</eis-type>
  <version>1.0</version>
  <resourceadapter>
    <managedconnectionfactory-class>
      com.sun.connector.blackbox.NoTxManagedConnectionFactory
    </managedconnectionfactory-class>
    <connectionfactory-interface>javax.sql.DataSource
    </connectionfactory-interface>
    <connectionfactory-impl-class>
      com.sun.connector.blackbox.JdbcDataSource
    </connectionfactory-impl-class>
    <connection-interface>
      java.sql.Connection
    </connection-interface>
    <connection-impl-class>
      com.sun.connector.blackbox.JdbcConnection
    </connection-impl-class>
    <transaction-support>NoTransaction</transaction-support>
    <config-property>
      <config-property-name>ConnectionURL</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value>
        jdbc:cloudscape:rmi:CloudscapeDB;create=true
      </config-property-value>
    </config-property>
  </resourceadapter>
</connector>
```

```

</config-property>
<authentication-mechanism>
  <authentication-mechanism-type>
    BasicPassword
  </authentication-mechanism-type>
  <credential-interface>
    javax.resource.security.PasswordCredential
  </credential-interface>
</authentication-mechanism>
  <reauthentication-support>>false</reauthentication-support>
</resourceadapter>
</connector>

```

Understanding the jrun-ra.xml deployment descriptor

If a resource adapter has a `display-name` value in its `ra.xml` file, JRun can deploy it without a `jrun-ra.xml` file; the display name becomes the JNDI name for application components, such as EJBs, to look up the adapter. A `jrun-ra.xml` file is required to modify configuration properties, set connection pooling values, include security information, or use multiple `ManagedConnectionFactory` instances.

Note: You can automatically generate a `jrun-ra.xml` file. For more information, see “Working with JRun-specific deployment descriptors,” in Chapter 3.

Example: jrun-ra.xml deployment descriptor

The following `jrun-ra.xml` file configures a resource adapter instance that requires authentication and one that does not:

```

<?xml version="1.0" encoding="UTF-8"?>
<jrun-connectionfactory>
  <managedconnectionfactory-class>
    com.sun.connector.blackbox.NoTxManagedConnectionFactory
  </managedconnectionfactory-class>
  <managedconnectionfactory-instance>
    <jndi-name>J2EEConnector/secure-blackbox-notx</jndi-name>
    <config-property>
      <config-property-name>ConnectionURL</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value>
        jdbc:cloudscape:CloudscapeDB;create=true
      </config-property-value>
    </config-property>
    <security-info>
      <username>victor</username>
      <password>victor</password>
    </security-info>
    <connection-pool>
      <name>mypool</name>
      <objectType>java.lang.StringBuffer</objectType>
      <minimumSize>1</minimumSize>
      <maximumSize>10</maximumSize>
      <connectionTimeout>6</connectionTimeout>
      <userTimeout>12</userTimeout>
      <skimmerFrequency>300</skimmerFrequency>
      <shrinkBy>2</shrinkBy>
    </connection-pool>
  </managedconnectionfactory-instance>
</jrun-connectionfactory>

```

```

        <debugging>true</debugging>
    </connection-pool>
</managedconnectionfactory-instance>
<managedconnectionfactory-instance>
    <jndi-name>J2EEConnector/blackbox-notx</jndi-name>
    <config-property>
        <config-property-name>ConnectionURL</config-property-name>
        <config-property-type>java.lang.String</config-property-type>
        <config-property-value>
            jdbc:cloudscape:CloudscapeDB;create=true
        </config-property-value>
    </config-property>
</connection-pool>
    <name>mypool</name>
    <objectType>java.lang.StringBuffer</objectType>
    <minimumSize>1</minimumSize>
    <maximumSize>10</maximumSize>
    <connectionTimeout>6</connectionTimeout>
    <userTimeout>12</userTimeout>
    <skimmerFrequency>300</skimmerFrequency>
    <shrinkBy>2</shrinkBy>
    <debugging>true</debugging>
</connection-pool>
</managedconnectionfactory-instance>
</jrun-connectionfactory>

```

Configuring enterprise applications

This section discusses the standard enterprise application deployment descriptor, `application.xml`, and the J2EE **extension mechanism**, which lets you reconcile the dependency of individual modules on the same class libraries.

Understanding the `application.xml` deployment descriptor

The application assembler uses the `application.xml` file to declare an enterprise application and the J2EE modules that it contains. An enterprise application contains one or more web applications, EJBs, or enterprise resource adapters. The application assembler can also use the `application.xml` file to specify security roles that apply to all of the application modules.

The `application.xml` file's `alt-dd` element lets the assembler use an alternative deployment descriptor rather than the one inside a module archive file. You do not need to change deployment descriptors inside individual WAR, JAR, and RAR files.

The `application.xml` file contains the following top-level elements under the `application` element:

XML element	Required/ Optional	Description
<code>icon</code>	optional	Images for use by tools.
<code>display-name</code>	required	Short name for use by tools.
<code>description</code>	optional	Description of the enterprise application for use by tools.
<code>module</code> (zero or more)	required	<p>Path to J2EE module archive files or expanded directories in the enterprise application.</p> <p>The following J2EE modules are supported:</p> <ul style="list-style-type: none">• <code>connector</code>: enterprise resource adapter• <code>ejb</code>: EJB• <code>web</code>: web application <p>JRun deploys EJBs and resource adapters before web applications because web applications might use EJBs and resource adapters. Multiple EJB modules are deployed in the order specified in the deployment descriptor.</p> <p>The optional <code>alt-dd</code> element specifies the full path to an alternative J2EE module deployment descriptor relative to the enterprise application's root directory. If there is no <code>alt-dd</code> listed, JRun reads the original module deployment descriptor.</p> <p>The optional <code>web/context-root</code> element specifies the context root (logical root) of a web application.</p>
<code>security-role</code> (zero or more)	optional	Declaration of an application-wide security role.

Example: application.xml deployment descriptor

The following deployment descriptor declares an EJB module in an expanded directory, an EJB module in a JAR file, a web application module in an expanded directory, a web application in a WAR file, and a security role:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE application PUBLIC
    "-//Sun Microsystems, Inc.//DTD J2EE Application 1.3//EN"
    "http://java.sun.com/dtd/application_1_3.dtd">
<application>
  <display-name>samples-ear</display-name>
  <module>
    <ejb>account</ejb>
  </module>
  <module>
    <ejb>shoppingcart.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>samples-war</web-uri>
      <context-root>/samples</context-root>
    </web> <web>
      <web-uri>storefront.war</web-uri>
      <context-root>/storefront</context-root>
    </web>
  </module>
  <security-role>
    <description>Administrator role</description>
    <role-name>Admin</role-name>
  </security-role>
</application>
```

Handling J2EE module dependencies

JRun gives the assembler a convenient way to reconcile the dependency of individual J2EE modules on the same class libraries. Files listed in a `Class-Path` attribute in a module's `META-INF/manifest.mf` file are automatically added to the classpath. A `Class-Path` attribute can list any file that you can use in a normal classpath, including JAR files, ZIP files, and directories. This feature is available for enterprise applications, EJBs, web applications, and resource adapters in archive files or expanded directories.

The assembler can place a single copy of a class library in an enterprise application, and specify its classpath in `Class-Path` attribute of each module that requires it.

To use a `Class-Path` attribute in a `manifest.mf` file:

- 1 Open an existing `manifest.mf` file or create a new one in a text editor.
- 2 Add a `Class-Path` attribute on its own line that includes the relative URL of each JAR file you want added to the classpath separated by spaces. The URLs must be relative to the module and not the enterprise application that contains it.

For example:

```
Class-Path: common/util1.jar common/log4j.jar
```

- 3 Save the manifest.mf file.
- 4 Add the manifest.mf file to a META-INF directory under the module's root directory. For archived modules, this requires re-archiving the module.
- 5 Redeploy the module. You can hot deploy by changing a deployment descriptor or modifying a module archive file.

CHAPTER 2

Packaging J2EE Modules

This chapter provides information about packaging J2EE modules for deployment in JRun.

Contents

- Module packaging overview..... 32
- Packaging web applications..... 33
- Packaging Enterprise JavaBeans 37
- Packaging enterprise resource adapters..... 39
- Packaging enterprise applications..... 40

Module packaging overview

JRun supports the following types of J2EE modules:

- Web application (WAR file or directory)
- Enterprise JavaBean (JAR file or directory)
- Enterprise resource adapter (RAR file only)
- Enterprise application (EAR file or directory)

If you are unfamiliar with these J2EE modules, see *Getting Started with JRun*.

Prior to deploying a module, you must create the module's deployment descriptors and package its constituent parts in the appropriate directory structure. For more information about deployment descriptors, see Chapter 1, "Configuring J2EE Modules" on page 1.

Choosing between archive files and expanded directories

JRun lets you deploy J2EE modules from Java archive files or expanded directories. In a production environment, it is best to deploy archive files for greater portability. During development, deploying expanded directories provides the greatest flexibility and ease-of-use.

A J2EE module must conform to its specified directory structure. The module directory structure must contain the appropriate standard deployment descriptor in the subdirectory specified for that type of module; depending on the module, the directory structure might also contain a JRun-specific deployment descriptor.

Understanding assembler, deployer, and administrator roles

Before packaging individual J2EE modules in an enterprise application or moving a J2EE module from a development environment to a production environment, you or others might have to perform several tasks as assembler, deployer, or administrator. For more information, see "Understanding J2EE roles in module configuration," in Chapter 1.

Packaging web applications

Web applications must conform to the directory structure in the following table; you can deploy them on their own or as part of an enterprise application:

Directory	Description
<i>web_app</i> (root directory or WAR root)	Contains the WEB-INF directory and all files that must be accessible by the user's web browser, such as JSPs, HTML pages, cascading stylesheets, images, and JavaScript files. You can place these files directly in the web application root directory or in arbitrary subdirectories that do not use the reserved name WEB-INF.
/WEB-INF	Contains the classes and lib directories, the standard web application deployment descriptor (<i>web.xml</i>), and possibly a JRun-specific deployment descriptor (<i>jrun-web.xml</i>). For information about deployment descriptors, see Chapter 1, "Configuring J2EE Modules" on page 1.
/WEB-INF/classes	Contains servlets, other Java classes, and associated resources that are not packaged in JAR files.
/WEB-INF/lib	Contains JAR files containing classes and associated resources.

This section provides general information about preparing web applications for deployment in JRun, and does not discuss specific tools. JRun provides **XDoclet**, an open source tool that generates web application deployment descriptors and JSP tag library descriptors (TLDs) based on special Javadoc comments in servlet and tag library source files with filenames that end with *Servlet* or *Library*. JRun extends XDoclet to support the JRun-specific deployment descriptors, and provides automatic compilation and deployment. For more information, see *JRun Programmer's Guide* and the sourceforge.xdoclet.org website.

To package a web application:

- 1 Create a staging directory containing the JSPs, HTML files, images, and other referenced files that must be accessible by the user's web browser. To easily identify the directory as a web application directory, use a directory name that ends with *-war*. Maintain the original directory structure of referenced files.
- 2 In the staging directory, create a directory called *WEB-INF*.
- 3 In the *WEB-INF* directory, create directories called *classes* and *lib*.
- 4 Copy the application's servlets and any other unarchived class files to the *WEB-INF/classes* directory.
- 5 If the web application uses JSP tag libraries, copy any tag libraries packaged in a JAR file to the *WEB-INF/lib* directory. Copy any unarchived tag library classes to the *WEB-INF/classes* directory.

When deployed inside a JAR file, a TLD file must be in the *META-INF* directory or a subdirectory of the *META-INF* directory. When deployed directly to a web application, the tag library descriptor file(s) must be in the *WEB-INF* directory or a subdirectory of

it. The path to a TLD file in a JSP tag library directive's `uri` attribute starts with a forward slash and is relative to the root of the web application containing the JSP; the tag library directive can point to a TLD file directly or to a JAR file containing a TLD file.

- 6 Place the web application deployment descriptor(s) in the WEB-INF directory. Web application deployment descriptors include the following:
 - A standard J2EE web application deployment descriptor (`web.xml`) in the WEB-INF directory.
 - If necessary, a JRun-specific deployment descriptor (`jrun-web.xml`) in the WEB-INF directory. This deployment descriptor lets you set a context-root for the web application, enable and disable automatic servlet compilation and reloading, set virtual paths, configure servlet session persistence and cookies, and set values for JNDI names.

For more information about deployment descriptors, see Chapter 1, “Configuring J2EE Modules” on page 1.

- 7 (Optional) For performance and security reasons, before moving the web application to a production environment, you might want to disable JSP page compilation and distribute only binary `.class` files rather than text-based `.jsp` files. This is a two-step process:
 - a Disable dynamic JSP compilation. See the following section, “Disabling dynamic JSP compilation”.
 - b Precompile JSPs with the JSPC compiler. See “Precompiling JSPs” on page 35.

Note: You cannot use JSPs precompiled for JRun 3.x in JRun 4. You must delete the original compiled JSPs and recompile them for JRun 4.

- 8 To deploy the web application as a stand-alone module in a development environment, copy the directory structure to a JRun deploy directory. The default deploy directory is the root directory of a JRun server (`jrun_root/servers/jrun_server`). You can use the JMC to add or remove deploy directories. JRun automatically deploys modules in a deploy directory. For more information about deployment, see Chapter 3, “Deploying J2EE Modules” on page 43.
- 9 To prepare the web application for a production environment, package it in a WAR file using the following `jar` utility command from the top level of the staging directory:

```
jar cvf web_module.war .
```

where `web_module` is the web application name.
- 10 If the web application is part of an enterprise application, see “Packaging enterprise applications” on page 40.

Disabling dynamic JSP compilation

For performance and security reasons, you can disable dynamic JSP compilation and distribute binary .class files rather than text-based .jsp files.

To disable dynamic JSP compilation:

In the `jrun_root/servers/jrun_server/SERVER-INF/default-web.xml` file, set the JSPServlet's `translationDisabled` initialization parameter to true as shown in bold in the following example:

```
<servlet>
<servlet-name>JSPServlet</servlet-name>
<servlet-class>jrun.jsp.JSPServlet</servlet-class>
<init-param>
<param-name>translationDisabled</param-name>
<param-value>true</param-value>
</init-param>
</servlet>
```

When you restart the JRun server, the change takes effect.

Precompiling JSPs

There are several reasons to compile your JSPs:

- During development, you may find it faster to fix errors by explicitly compiling your pages, rather than by triggering the compilation through a web request for the page.
- For improved security, you can disable JSP compilation and distribute only the binary .class files rather than your text-based .jsp files.

The JSPC compiler is a command-line tool that you use to compile JSPs outside the context of a web server. With the JSPC compiler, you can explicitly compile JSPs from a command line, rather than use JRun to compile them upon a request of the JSP. JSPC uses the IBM jikes compiler if it is located in the `jrun_root/bin` directory. Otherwise, it uses the Sun javac compiler.

Note: The JSPC compiler is the same compiler that JRun uses to compile JSPs. The only difference is that you invoke the JSPC compiler from a command line.

Invoking JSPC

You invoke the JSPC compiler using the following command. JSPC uses the IBM jikes compiler when it is in the `jrun_root/bin` directory; otherwise it uses the Sun javac compiler.

```
jspc
[-w web_app_root]
[-d output_directory]
[-k keep processing on errors]
[JSP names]
```

JSPC compiler arguments

JSPC uses the following arguments:

Argument	Description
-w	(Optional) Full path to the web application's root directory. By default, this is the current working directory.
-d	(Optional) Location where you want the JSPC compiler to write the output .class and .java files. By default, this is the WEB-INF/jsp directory relative to the web application's root directory.
-k	(Optional) Instructs JSPC to continue processing when it encounters errors.
<i>JSP names</i>	(Optional) Names of the JSPs to compile. If you do not specify any JSPs, JSPC compiles any JSPs it finds in the web application.

You must specify the `-w` argument when using the JSPC compiler to process web applications that have virtual mappings specified in the web application's `run-web.xml` deployment descriptor.

JSPC compiler examples

The following example compiles the page `foo.jsp`, which is part of a web application with the document root directory at `c:/myapps/store`. In this example, the `CLASSPATH` environment variable contains `run.jar`, which is required to execute the JSPC compiler.

```
jspc -w c:\jrun -d c:\myapps\store\WEB-INF\classes foo.jsp
```

The following example compiles multiple JSPs. Note that the JSP path uses wildcards:

```
jspc -d c:\myapps\store\WEB-INF\classes *.jsp store\*.jsp
```

Packaging Enterprise JavaBeans

EJB modules can contain one or more EJBs and must conform to the directory structure in the following table; you can deploy them on their own or as part of an enterprise application:

Directory	Description
<i>ejb_module</i> (root directory or JAR root)	Contains the compiled EJB interfaces and implementation classes in subdirectories that match their Java package structure.
/META-INF	Contains the standard EJB deployment descriptor (<i>ejb-jar.xml</i>) and possibly a JRun-specific deployment descriptor (<i>jrun-ejb-jar.xml</i>). For more information about deployment descriptors, see Chapter 1, “Configuring J2EE Modules” on page 1.

This section provides general information about preparing EJBs for deployment in JRun, and does not discuss specific tools. JRun provides two tools that help you develop, package, and deploy EJBs:

- The **Enterprise Deployment Wizard** helps you with the entire EJB development, assembly, and deployment process.
- **XDoclet** is an open source tool that generates EJB interfaces and deployment descriptors based on special Javadoc comments in a bean implementation source file. JRun extends XDoclet to support the JRun-specific deployment descriptors, and provides automatic compilation and deployment.

For more information about these tools, see *JRun Programmer's Guide* and the sourceforge.xdoclet.org website.

You can package an application's beans together or individually:

- Packaging beans individually provides the best reusability.
- Packaging all the beans together is the easiest method. It is best to use this method only when you know all the beans are going to be used.
- Packaging related beans together is a good choice for applications containing a number of beans that fall into specific categories.

To package an EJB module:

- 1 Create a staging directory containing the compiled EJB interfaces and implementation classes in subdirectories that match their Java package structure. To easily identify the directory as an EJB directory, use a directory name that ends with `-ejb`.
- 2 In the staging directory, create a directory called `META-INF`.
- 3 Place deployment descriptors in the `META-INF` directory. Deployment descriptors include the following:
 - A standard EJB deployment descriptor (*ejb-jar.xml*).
 - If necessary, a JRun-specific deployment descriptor (*jrun-ejb-jar.xml*). This deployment descriptor lets you define JRun-specific functionality for your EJB module, including JNDI names, container-managed persistence (CMP), and message-driven bean settings.

Note: You can automatically generate a default `jrunit-ejb-jar.xml` file, which you can edit to suit your needs. For more information, see “Working with JRun-specific deployment descriptors,” in Chapter 3.

You can use the JRun Enterprise Deployment Wizard or XDoclet to build deployment descriptors, or you can manually code them. For more information about EJB deployment descriptors, see Chapter 1, “Configuring J2EE Modules” on page 1.

- 4 To deploy the EJB module as a stand-alone module in a development environment, copy the directory structure to a JRun deploy directory. The default deploy directory is the root directory of a JRun server (*jrunit_root/servers/jrunit_server*). You can use the JMC to add or remove deploy directories. JRun automatically deploys modules in a deploy directory. For more information about deployment, see Chapter 3, “Deploying J2EE Modules” on page 43.
- 5 To prepare the EJB module for a production environment, package its directory structure in a JAR file using the following `jar` utility command from the top level of the staging directory:

```
jar cvf ejb_module.jar .
```

where *ejb_module* is the name of the EJB module.
- 6 If the EJBs are part of an enterprise application, see “Packaging enterprise applications” on page 40.

Packaging enterprise resource adapters

Enterprise resource adapters can contain one or more resource adapters in a RAR archive file that conforms to the directory structure in the following table; you can deploy them on their own or as part of an enterprise application:

Directory	Description
<i>ra_module</i> (RAR root)	Contains the META-INF directory and resource adapter classes in subdirectories that match their Java package structure.
/META-INF	Contains the standard resource adapter deployment descriptor (<i>ra.xml</i>) and possibly a JRun-specific deployment descriptor (<i>jrun-ra.xml</i>). For more information about deployment descriptors, see Chapter 1, “Configuring J2EE Modules” on page 1.

To package an enterprise resource adapter module:

- 1 Create a staging directory containing the resource adapter class files in subdirectories that match their Java package structure; optionally, you can package the resource adapter class files in a JAR file inside the staging directory. To easily identify the directory as a resource adapter directory, use a staging directory name that ends with *-rar*.
- 2 In the staging directory, create a subdirectory called META-INF.
- 3 Place deployment descriptor(s) in the META-INF directory. Deployment descriptors include the following:
 - A standard resource adapter deployment descriptor (*ra.xml*). This deployment descriptor provides information about the resource adapter implementation code, configuration properties, and security information.
 - If necessary, a JRun-specific deployment descriptor (*jrun-ra.xml*). A *jrun-ra.xml* file is not required when there is a `display-name` in the *ra.xml* file and you require only one managed connection factory instance. This deployment descriptor lets you define JRun-specific functionality, including JNDI names for resource adapter instances, configuration properties, security settings, and connection pool settings. For more information about resource adapter deployment descriptors, see Chapter 1, “Configuring J2EE Modules” on page 1.
- 4 To prepare the resource adapter(s) for deployment or distribution, package the directory structure in a RAR file using the following Java *jar* utility command from the top level of the staging directory:

```
jar cvf ra_module.rar .
```

where *ra_module* is the name of the resource adapter module.
- 5 To deploy the resource adapter module as a stand-alone module, copy the RAR file to a JRun deploy directory. The default deploy directory is the root directory of a JRun server (*jrun_root/servers/jrun_server*). You can use the JMC to add or remove deploy directories. JRun automatically deploys modules in a deploy directory. For more information about deployment, see Chapter 3, “Deploying J2EE Modules” on page 43.

- 6 If the resource adapters are part of an enterprise application, see “Packaging enterprise applications” on page 40.

Packaging enterprise applications

An enterprise application consists of one or more J2EE modules, and an enterprise application deployment descriptor (application.xml) packaged in an EAR file or an expanded directory. The enterprise application also contains any classes that its modules depend on.

An enterprise application must conform to the directory structure in the following table:

Directory	Description
<i>enterprise_app</i> (root directory or EAR root)	Contains the META-INF directory and: <ul style="list-style-type: none">• Web application WAR files or directories• EJB JAR files or directories• Enterprise resource adapter RAR files or directories• Packaged dependency and utility classes <p>You can place J2EE modules at the root level or in subdirectories.</p>
/META-INF	Contains a properly configured enterprise application deployment descriptor (application.xml). For more information about deployment descriptors, see Chapter 1, “Configuring J2EE Modules” on page 1.

To package an enterprise application:

- 1 Create a staging directory containing the enterprise application’s J2EE modules in archive files or expanded directories. To easily identify the directory as an enterprise application directory, use a directory name that ends with -ear.
- 2 To make the enterprise application portable, ensure that it has no dependencies outside its expanded directory or EAR file, other than dependencies on external resources such as JDBC, JMS, and JavaMail resources. For example, if a web application module depends on virtual mappings to access libraries not in the web application, copy the required files to the web application’s WEB-INF/lib directory.
- 3 Reconcile all module dependencies. More than one module in an enterprise application might require access to the same utility classes.
For more information on dealing with such situations, see Chapter 1, “Configuring J2EE Modules” on page 1.
- 4 Place the application.xml file in the META-INF directory of the enterprise application directory structure.
- 5 To deploy the enterprise application in a development environment, copy its directory structure to the root directory of a JRun server (*jrun_root/servers/jrun_server*). JRun automatically deploys modules in a JRun server’s root directory. For more information about deployment, see Chapter 3, “Deploying J2EE Modules” on page 43.

- 6 To prepare the enterprise application for a production environment, package its directory structure in an EAR file using the following jar command from the top level of the staging directory:

```
jar cvf enterprise_module.ear .
```

where *enterprise_module* is the name of the enterprise application.

If the enterprise application's modules are in expanded directories, you can optionally package them in appropriate archive files and package those in the EAR file.

CHAPTER 3

Deploying J2EE Modules

This chapter provides information about deploying J2EE modules in JRun.

Contents

- Module deployment overview 44
- Dynamic module deployment 45
- Working with JRun-specific deployment descriptors..... 48

Module deployment overview

JRun makes it easy to deploy and undeploy the following types of J2EE modules from Java archive files or expanded directories:

- Web application (WAR file or directory)
- Enterprise JavaBean (JAR file or directory)
- Enterprise resource adapter (RAR file only)
- Enterprise application (EAR file or directory)

If you are unfamiliar with J2EE modules, see *Getting Started with JRun*.

You can automatically deploy modules by placing them in an **autodeploy directory**, or you can deploy them using the JMC. You can also use the Enterprise Deployment Wizard (EJBs only) to create EJB template code and deployment descriptors, and deploy EJBs to a JRun server; for information, see *JRun Programmer's Guide*.

You can use the JMC to undeploy and redeploy modules, but you cannot use it to undeploy modules that are in a deploy directory. To undeploy modules in a deploy directory, manually remove them from the directory.

This chapter provides information only about deploying and undeploying modules. For information about packaging modules, see Chapter 2, "Packaging J2EE Modules" on page 31.

Dynamic module deployment

Overview

JRun provides dynamic deployment of new and modified enterprise applications, web applications, EJBs, and enterprise resource adapters. You do not have to restart the JRun server after adding a new module or changing a deployed module. This functionality is provided by the following features:

- **Autodeploy directories** When a JRun server starts, JRun automatically deploys a module when it detects a module archive file or an expanded module directory in a deploy directory. By default, this functionality is combined with hot deploy, which automatically deploys a module copied to a deploy directory of a running JRun server. The default deploy directory is *jrun_root/servers/jrun_server*, but you can create additional deploy directories, and modify or remove existing ones through the JMC or by manually adding or removing `deployDirectory` attributes to the `DeployerService` section of the JRun server's *jrun_root/servers/server_name/SERVER-INF/jrun.xml* file. For example, to configure the directory *c:/my_deployments* as a deploy directory, you would add the following entry to the *jrun.xml* file:

```
<attribute name="deployDirectory">c:/my_deployments</attribute>
```

JRun also automatically deploys a module across a server cluster when it detects a module archive file in a cluster deploy directory. This feature does not support expanded directories. The default cluster deploy directory is *jrun_root/servers/jrun_server/SERVER-INF/cluster*, but you can change the directory through the JMC or by changing the `deployDirectory` attribute in the `ClusterDeployerService` section of the JRun server's *jrun_root/servers/server_name/SERVER-INF/jrun.xml* file. When you use the JMC to deploy a module to a cluster, JRun copies the module archive file to a cluster deploy directory on one of the clustered servers.

- **Autodeploy files (JMC)** JRun automatically deploys a module WAR, JAR, RAR, or EAR file, or an expanded module directory you add using the JMC.

You can also manually configure an autodeploy file by specifying it in a `url` or `file` attribute of the `DeployerService` in the JRun server's *jrun.xml* file; for example, to auto deploy an EJB module located in *c:/my_files/mybean.jar*, you add either of the following entries to the `DeployerService` section of the *jrun_root/servers/jrun_server/SERVER-INF/jrun.xml* file:

```
<attribute name="file">c:/my_files/mybean.jar</attribute>
```

```
<attribute name="url">file:c:/my_files/mybean.jar</attribute>
```

- **Hot deploy** Hot deploy automatically redeploys (restarts) a module upon detection of physical changes to a module archive file or deployment descriptors in expanded directories. For example, a WAR file is hot deployed when the *jrun-web.xml* file in an expanded directory is modified. Hot deploy also deploys modules copied to a deploy directory of a running JRun server.

Note: Macromedia strongly recommends that you disable hot deploy in a production environment. For more information, see the next section, “Controlling dynamic deployment”. When using hot deploy in a development environment, using expanded directories saves system resources, such as disk space and processing activity; hot deploying an archived module causes JRun to expand the archived module to a temp directory.

When auto deploying a module, JRun creates a module name based on the module directory name or archive filename without the file extension. For web applications, JRun also creates a context root URL mapping based on the module directory name or archive filename without the file extension.

For example, if you add a web application, `newapp.war`, to the root directory of the default server, JRun creates the following:

- Web application name: `newapp`
- Context root: `/newapp`

Controlling dynamic deployment

You control autodeploy directories using the JMC or by manually creating or editing `deployDirectory` attributes in the `DeployerService` or `ClusterDeployerService` (for clusters) section of the `jrun_root/servers/jrun_server/SERVER-INF/jrun.xml` file. By default, there is one deploy directory, but you can create additional ones, and modify or remove existing ones.

JRun automatically deploys any new module archive file or directory found in the directory specified in a `deployDirectory` attribute. JRun checks this location upon server startup and also monitors for new files while the server is running. Deleting all `deployDirectory` attributes disables auto deploy and hot deploy. This is the default `deployDirectory` attribute:

```
<attribute name="deployDirectory">{jrun.server.rootdir}</attribute>
```

To disable hot deploy while auto deploy remains enabled, you use the JMC or manually add a `hotDeploy` attribute to the `jrun_root/servers/jrun_server/SERVER-INF/jrun.xml` file. By default, hot deploy is enabled and there is no `hotDeploy` attribute in the `jrun.xml` file. Adding the following line to the `DeployerService` or `ClusterDeployerService` (for clusters) section of the `jrun.xml` file disables hotdeploy:

```
<attribute name="hotDeploy">false</attribute>
```

Deploying modules for development

During development, it is very convenient to hot deploy expanded module directories in a JRun server deploy directory so you can test changes without repackaging and redeploying modules or restarting JRun. The default deploy directory is `jrun_root/servers/jrun_server`.

Note: You work directly with files in the deploy directory itself and not in another directory to which JRun copies files. This functionality differs from JRun 3.1.

By default, the JRun hot deploy feature watches for changes to the deployment descriptor files in an expanded directory, and redeploys the module when a file changes. You can change the designated deploy directory or disable dynamic deployment by editing properties in the server's `jrun_root/servers/server_name/SERVER-INF/jrun.xml` file. For more information, see “Dynamic module deployment” on page 45.

Before deploying a web application originally deployed in JRun 3.x to a JRun server of the same name, you must delete the .java files in the web_app/WEB-INF/jsp directory. Also, JRun does not keep generated JSPs by default; JRun 3.x did keep generated JSPs by default. To keep generated JSPs, in the *jrun_root/servers/jrun_server/SERVER-INF/default-web.xml* file, set the JSPServlet's `keepGenerated` initialization parameter to `true` as shown in bold in the following example. The change takes effect when you restart the JRun server.

```
<servlet>
<servlet-name>JSPServlet</servlet-name>
<servlet-class>jrun.jsp.JSPServlet</servlet-class>
<init-param>
<param-name>keepGenerated</param-name>
<param-value>true</param-value>
</init-param>
</servlet>
```

Deploying modules for production

It is good practice to package modules in the appropriate archive files for deployment in a production environment. This makes modules more portable, and also makes it easier to control module changes. You should also disable hot deploy in a production environment. The JMC gives you a simple way to disable hot deploy. For information about disabling hot deploy, see “Dynamic module deployment” on page 45.

Before you deploy a module in a production environment, you might have to perform several packaging tasks. For more information, see “Understanding assembler, deployer, and administrator roles,” in Chapter 2.

Undeploying and redeploying modules

You can use the JMC to undeploy modules in a JRun server regardless of whether the server is running. If you use the JMC to undeploy a module deployed using a deploy directory, the module will be redeployed the next time the JRun server starts unless you manually remove the module from the deploy directory. Outside the JMC, you can undeploy modules located in a deploy directory by manually removing them from the directory.

You have the following options for redeploying modules:

- While a JRun server is running, hot deploy watches modules for changes to module archive files and deployment descriptors in expanded directories, and dynamically redeploys modules when files change. Hot deploy also redeploys a module when you copy it to a deploy directory.
- The JMC lets you redeploy modules on an individual basis.
- Restarting a JRun server redeploys modules in a deploy directory or those deployed using the JMC.

Defining users, groups, and roles for authentication

For J2EE modules that use authentication, the deployer might work with the administrator to define users, groups, and roles for the target JRun server. These users, groups, and roles must correspond to the security roles defined in the J2EE modules. For more information, see *JRun Administrator's Guide*.

Working with JRun-specific deployment descriptors

Generating a JRun-specific deployment descriptor

JRun can automatically generate JRun-specific deployment descriptors for EJBs, web applications, and resource adapters when you deploy a module. This feature provides the following benefits:

- The generated descriptors are prefilled with default values, providing a starting point for custom configuration. If you are manually creating deployment descriptors, this method is easier than creating a `jrun-ejb-jar.xml` file from scratch.
- For previously deployed modules for which you modified settings using the JMC, the generated descriptor includes elements corresponding to the modified settings. For example, if you used the JMC to disable automatic servlet compilation, a corresponding element is added to the `jrun-web.xml` file.
- For JRun 3.x EJBs, JRun generates a `jrun-ejb-jar.xml` with elements corresponding to Ejbpt-specific environment entries in the JRun 3.x `ejb-jar.xml` file. JRun also generates an EJB 2.0 `ejb-jar.xml` file.
- For J2EE Reference Implementation (RI) EJBs, JRun generates a `jrun-ejb-jar.xml` file with elements that correspond to those in the RI-specific deployment descriptor.

To generate a JRun-specific deployment descriptor:

- 1 In the `jrun_root/servers/jrun_server/SERVER-INF/jrun.xml` file, set the `persistXML` attribute of the `DeployerService` to `true`.
- 2 Restart the JRun server.
- 3 Deploy a J2EE module containing a standard deployment descriptor.

If the module is in an expanded directory, the deployment descriptor is generated in the same directory as the standard deployment descriptor. If the module is in an archive file, the deployment descriptor is generated in the same directory as the archive file.

Using a JRun-specific deployment descriptor outside an archived module

For modules in archive files, you do not need to add a JRun-specific deployment descriptor to the archive file. Instead, you can include the deployment descriptor in the same directory as the archive file and rename it as `module_name.module_type.jrun.xml`, where `module type` is `ejb`, `war`, or `rar`.

For example, you would use the name `mywebapp.war.jrun.xml` for the JRun-specific deployment descriptor of a web application in an archive file named `mywebapp.war`.

INDEX

- A**
 - administrator role 3
 - application.xml 28
 - archive files and expanded
 - directories 32
 - assembler role 3
 - assembly descriptor, EJB 13
 - auto deploying 45
 - automatic compiling 7
 - B**
 - BMP entity bean declarations 15
 - C**
 - CMP
 - 1.1 entity bean declarations 15
 - 2.0 entity bean declarations 16
 - compiling
 - JSPC 35
 - servlets 7
 - configuration roles 3
 - configuring
 - EJBs 9
 - enterprise applications 28
 - resource adapters 24
 - web applications 4
 - context roots, web applications 7
 - D**
 - dependencies, J2EE modules 29
 - deployer role 3
 - deploying
 - auto 45
 - controlling 46
 - development 46
 - hot 45
 - modules 45
 - overview 44
 - production 47
 - tools 9
 - deployment descriptors 2
 - ejb-jar.xml 9
 - enterprise applications 28
 - generating JRun-specific 48
 - JRun 2
 - jrun-ejb-jar.xml 17
 - jrun-ra.xml 26
 - jrun-specific 48
 - jrun-web.xml 7
 - ra.xml 24
 - resource adapters 24
 - web applications 4
 - web.xml 4
 - developer role 3
 - directory structure
 - EJBs 37
 - enterprise applications 40
 - resource adapters 39
 - web applications 33
 - disabling JSP compilation 35
 - E**
 - EARs
 - configuring 28
 - packaging 40
 - ejb 15
 - EJB assembly descriptors 13
 - ejb-jar.xml
 - BMP entity bean declaration, assembly descriptor 15
 - CMP 1.1 entity bean declaration 15
 - CMP 2.0 entity bean declaration 16
 - stateless session bean declaration 14
 - EJBs
 - configuring 9
 - deployment descriptors 2
 - directory structure 37
 - packaging 37
 - enterprise applications
 - configuring 28
 - deployment descriptors 2
 - directory structure 40
 - packaging 40
 - Enterprise Deployment Wizard 9
 - enterprise resource adapters
 - configuring 24
 - deployment descriptors 24
 - packaging 39
 - entity bean declarations, CMP 1.1 15
 - entity bean elements 13
 - examples
 - application.xml 29
 - ejb-jar.xml 14
 - jrun-ejb-jar.xml 20
 - jrun-ra.xml 26
 - JSPC compiler 36
 - ra.xml 25
 - web.xml 5
 - expanded directories, archive files 32
 - H**
 - hot deploying 45
 - I**
 - instance pool size, session bean 19
 - J**
 - J2EE module dependencies 29
 - J2EE roles 3
-

- JARs
 - configuring 9
 - packaging 37
- JRun deployment descriptors 2
 - jrun-ejb-jar.xml 17
 - jrun-ra.xml 26
- JRun-specific deployment descriptors
 - generating 48
 - jrun-ejb-jar.xml 17
 - jrun-ra.xml 26
 - jrun-web.xml 7
- JSPC compiler 35
- JSPs
 - disabling compilation 35
 - disablingdynamiccompilation 35
 - JSPC compiler 35
 - keepGenerated 46
 - keeping generated 46
 - precompiling 35
 - translationDisabled 35
- K**
 - keepGenerated 46
 - keeping generated JSPs 46
- M**
 - Macromedia
 - developer resources vi
 - headquarters xi
 - sales xi
 - message-driven bean elements 13, 20
 - module dependencies 29
 - module deployment 45
 - controlling 46
 - overview 44
 - undeploying, redeploying 47
 - module packaging
 - EJBs 37
 - enterprise applications 40
 - enterprise resource adapters 39
 - overview 32
 - web applications 33
- P**
 - packaging
 - EJBs 37
 - enterprise applications 40
 - enterprise resource adapters 39
 - web applications 33
 - persistXML 48
 - precompiling JSPs 35
- R**
 - ra.xml deployment descriptor 24
- RARs
 - configuring 24
 - packaging 39
- redeploying, undeploying
 - modules 47
- reloading servlets 7
- resource adapters
 - configuring 24
 - deployment descriptors 2, 24
 - directory structure 39
 - packaging 39
- resources
 - books viii
 - online x
- roles
 - administrator 3
 - assembler 3
 - configuration 3
 - deployer 3
 - developer 3
 - J2EE 3
- S**
 - security, defining for web
 - applications 47
 - servlet
 - automatic compiling 7
 - automatic reloading 7
 - session configuration 7
 - session bean
 - instance pool size 19
 - timeout value 19
 - session bean elements 19
 - session configuration, web
 - applications 7
 - stateless session bean declarations 14
- T**
 - timeout value, session bean 19
 - translationDisabled 35
- U**
 - undeploying, redeploying
 - modules 47
 - Understanding 2, 4
 - use-web-server-root 40
- V**
 - virtual mappings
 - use-web-server-root 8
 - web applications 8
- W**
 - WARs
 - configuring 4
 - packaging 33
 - web applications
 - configuring 4
 - configuring security 6
 - context roots 7
 - deployment descriptors 2
 - directory structure 33
 - packaging 33
 - session configuration 7
 - use-web-server-root 8
 - virtual mapping 8
 - web.xml 4
- X**
 - XDoclet 9