Borland AppServer™ 6.7

Development Plug-in for Eclipse Guide

Borland Software Corporation
20450 Stevens Creek Blvd., Suite 800
Cupertino, CA 95014 USA
www.borland.com

BAS67WTPGuide
December 2006

Borland®

Contents

1

Introduction to Borland AppServer

Borland AppServer (AppServer) is a set of services and tools that enable you to build, deploy, and manage distributed enterprise applications in your corporate environment.

The AppServer is a leading implementation of the J2EE 1.4 standard, and supports the latest industry standards such as EJB 2.1, JMS 1.1, Servlet 2.4, JSP 2.0, CORBA 2.6, XML, and SOAP. Borland provides two versions of AppServer, which include leading enterprise messaging solutions for Java Messaging Service (JMS) management (Tibco and OpenJMS). You can choose the degree of functionality and services you need in AppServer, and if your needs change, it is simple to upgrade your license.

The AppServer allows you to securely deploy and manage all aspects of your distributed Java and CORBA applications that implement the J2EE 1.4 platform standard.

With AppServer, the number of server instances per installation is unlimited, so the maximum of concurrent users is unlimited.

AppServer includes:

- Implementation of J2EE 1.4.
- Apache Web Server version 2.2.3
- Borland Security, which provides a framework for securing AppServer.
- Single-point management of leading JMS management solutions included with AppServer (Tibco, and OpenJMS).
- Strong management tools for distributed components, including applications developed outside of AppServer.

AppServer features

AppServer offers the following features:

- Support for BAS platforms (please refer to http://support.borland.com/kbcategory.jspa?categoryID=389 for a list of the platforms supported for AppServer).
- Full support for clustered topologies.
- Seamless integration with the VisiBroker ORB infrastructure.
- Integration with the Borland JBuilder integrated development environment.
- Enhanced integration with other Borland products including Borland Optimizeit Profiler and ServerTrace.
- AppServer allows existing applications to be exposed as Web Services and integrated with new applications or additional Web Services. Borland Web Services support is based on Apache Axis 1.2 technology, the next-generation Apache SOAP server that supports SOAP 1.2.

Borland AppServer Documentation

The AppServer documentation set includes the following:

- *Borland AppServer Installation Guide*—describes how to install AppServer on your network. It is written for system administrators who are familiar with Windows or UNIX operating systems.
- *Borland AppServer Developer's Guide*—provides detailed information about packaging, deployment, and management of distributed object-based applications in their operational environment.
- *Borland Management Console User's Guide*—provides information about using the Borland Management Console GUI.
- *Borland Security Guide*—describes Borland's framework for securing AppServer, including VisiSecure for VisiBroker for Java and VisiBroker for C++.
- Borland *VisiBroker for Java Developer's Guide*—describes how to develop VisiBroker applications in Java. It familiarizes you with configuration and management of the Visibroker ORB and how to use the programming tools. Also described is the IDL compiler, the Smart Agent, the Location, Naming and Event Services, the Object Activation Daemon (OAD), the Quality of Service (QoS), and the Interface Repository.
- Borland *VisiBroker VisiTransact Guide*—describes Borland's implementation of the OMG Object Transaction Service specification and the Borland Integrated Transaction Service components.

The documentation is typically accessed through the Help Viewer installed with your AppServer product. You can choose to view help from the standalone Help Viewer or from within a AppServer GUI tool. Both methods launch the Help Viewer in a separate window and give you access to the main Help Viewer toolbar for navigation and printing, as well as access to a navigation pane. The Help Viewer navigation pane includes a table of contents for all AppServer books and reference documentation, a thorough index, and a comprehensive search page.

The PDF books, Borland AppServer Developer's Guide and Borland Management Console User's Guide are available online at http://info.borland.com/techpubs/appserver.

Accessing AppServer online help topics

To access the online help, use one of the following methods:

Windows

Choose Start|Programs|Borland Deployment Platform|Help Topics

or, launch the Web browser and open <AppServer_Home>/doc/index.html.

UNIX

Launch a Web browser and open <AppServer_Home>/doc/index.html.

Accessing AppServer online help topics from within a AppServer GUI tool

To access the online help from within a AppServer GUI tool, use one of the following methods:

- From within the Borland Management Console, choose Help|Help Topics
- From within the Borland Deployment Descriptor Editor (DDEditor), choose Help|Help Topics

Documentation conventions

The documentation for AppServer uses the typefaces and symbols described below to indicate special text:

Convention	Used for
italics	Used for new terms and book titles.
`computer`	Information that the user or application provides, sample command lines and code.
`bold computer`	In text, bold indicates information the user types in. In code samples, bold highlights important statements.
[ ]	Optional items.
...	Previous argument that can be repeated.
|	Two mutually exclusive choices.

Platform conventions

The AppServer documentation uses the following symbols to indicate platform-specific information:

Symbol	Indicates
Windows	All supported Windows platforms.
Win2003	Windows 2003 only
WinXP	Windows XP only
Win2000	Windows 2000 only
UNIX	UNIX platforms
Solaris	Solaris only

Contacting Borland support

Borland offers a variety of support options. These include free services on the Internet where you can search our extensive information base and connect with other users of Borland products. In addition, you can choose from several categories of telephone support, ranging from support on installation of Borland products to fee-based, consultant-level support and detailed assistance.

For more information about Borland's support services or contacting Borland Technical Support, please see our web site at http://support.borland.com and select your geographic region.

When contacting Borland's support, be prepared to provide the following information:

- Name
- Company and site ID
- Telephone number
- Your Access ID number (U.S.A. only)
- Operating system and version
- Borland product name and version
- Any patches or service packs applied
- Client language and version (if applicable)
- Database and version (if applicable)
- Detailed description and history of the problem
- Any log files which indicate the problem
- Details of any error messages or exceptions raised

Online resources

You can get information from any of these online sources:

World Wide Web: http://www.borland.com

Online Support: http://support.borland.com (access ID required)

World Wide Web

Check http://www.borland.com regularly. The AppServer Product Team posts white papers, competitive analyses, answers to FAQs, sample applications, updated software, updated documentation, and information about new and existing products.

You may want to check these URLs in particular:

- http://www.borland.com/downloads/download_appserver.html (AppServer software and other files)
- http://support.borland.com (AppServer FAQs)

Borland newsgroups

You can participate in many threaded discussion groups devoted to the AppServer. Visit http://www.borland.com/newsgroups for information about joining user-supported newsgroups for Enterprise Server and other Borland products.

Note

These newsgroups are maintained by users and are not official Borland sites.

2

Installing the Borland AppServer Development Plug-in

This chapter explains how to install the Borland AppServer (BAS) development plug-in for Eclipse.

Before you Begin

Before you install the Borland AppServer development plug-in for Eclipse, ensure that you have the following installed:

- Borland Application Server
- Eclipse 3.2 (http://download.eclipse.org/webtools/downloads/drops/R1.5/R-1.5.0-200606281455/)
- WTP 1.5 (http://download.eclipse.org/webtools/downloads/drops/R1.5/R-1.5.0-200606281455/)
- XDoclet 1.2.3 (http://sourceforge.net/project/showfiles.php?group_id=31602). Download the file named xdoclet-bin-1.2.3.zip.
- JDK 5.0

Installing the BAS Development Plug-in

To install the Borland AppServer development plug-in for Eclipse:

1. Save your project and exit Eclipse.
2. From the <APPSERVER_HOME>/etc/eclipse directory, copy the following folders:
   a. com.borland.bas.jst.server
   b. com.borland.enterprise.ui
   c. com.borland.enterprise.util
   d. com.borland.visibroker.sdk.core
3. Paste these folders to the <ECLIPSE_HOME>/plugins folder.
4. Start Eclipse.

Caution

if the system displays the Error in Loading Class and Dependency File dialog box after you start Eclipse, delete the .bundledata.<x> file from <ECLIPSE_HOME>\configuration\org.eclipse.osgi, where <x> is the version number of the bundledata file.

Note

You must ensure that you enable the XDoclet builder and set the XDoclet preferences correctly in Eclipse. To set XDoclet Runtime preferences in Eclipse, select Windows|Preferences and click XDoclet.

3

Overview

This chapter provides an overview about the Borland AppServer development plug-in for Eclipse.

Introduction

The Borland AppServer development plug-in for Eclipse enables you to do the following:

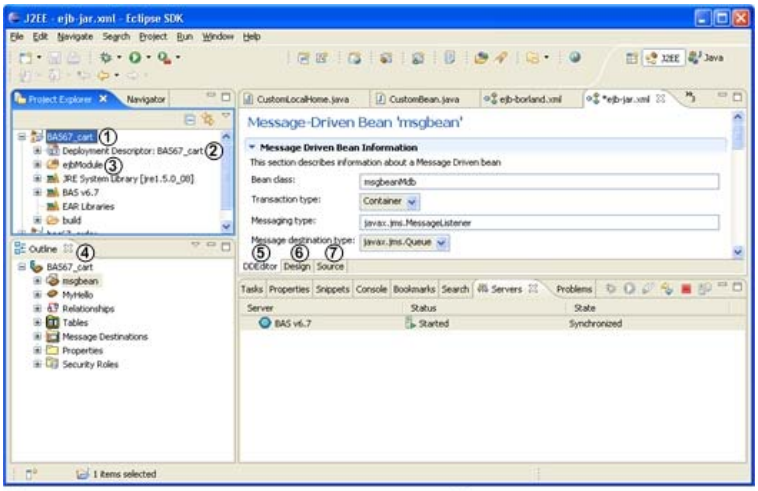
- Add the Borland AppServer as an installed runtime environment in Eclipse, Add the Borland AppServer as a new server, and start and stop the Borland AppServer in Eclipse
- Work with EJB projects. For more information on working with EJB projects, see “Working with EJB Projects”.
- Work with Web projects. For more information on working with Web projects, see “Working with Web Projects”.
- Work with EAR projects. For more information on working with WAR projects, see “Working with EAR Projects”.
- Work with Application Client projects. For more information on working with EJB projects, see “Working with Application Client Projects”.

Understanding the User Interface

Borland AppServer development plug-in provides the following views:

- Project Explorer view
- Editor view for the standard deployment descriptor file. The editor view contains the following:
  - Deployment Descriptor editor view.
  - Outline tree view.

The screen below describes the Eclipse user interface after you install the Borland AppServer development plug-in for Eclipse.



① EJB Project
② Logical Deployment Descriptor Node (maps to ejb-jar.xml)
③ Source Directory
④ Deployment Descriptor Outline Tree View
⑤ BAS Deployment Descriptor Editor View
⑥ XML Tree Editor View
⑦ XML Editor view

Project Explorer View

The Project Explorer view displays various files associated with a project. It also enables you to validate, deploy, and export the project.

Understanding the Project Structure

EJB Project

The EJB project contains the following:

- META-INF folder. This contains standard and vendor deployment descriptor files.
  - ejb-jar.xml
  - ejb-borland.xml
- XDoclet's merge files. These merge files are called by the Borland AppServer XDoclet Builder to merge custom data to the standard deployement descriptor file (ejb-jar.xml).
  - session-beans.xml
  - entity-beans.xml
  - message-driven-beans.xml

  Note

  We recommend that you do not modify these XDoclet merge files.
- Backup directory. Important business logic is often implemented in the abstract bean class. To prevent losing valuable code when you delete an EJB, the Boroland AppServer development plug-in backs up the abstract Bean class to the Backup directory.

Web Project

The Web project contains the following:

- The WebContent\WEB-INF directory. This contains standard and vendor deployment descriptor files.
  - web.xml
  - web-borland.xml

DDEditor View

The DDEditor view provides a user-friendly interface that enables you to configure and edit deployment descriptor artifacts. To activate the DDEditor view, click the DDEditor tab.

Deployment Descriptor Outline Tree View

The Deployment Descriptor Outline Tree view displays the deployment descriptor navigation tree. To display the Deployment Descriptor Outline Tree view, select Window|Show view|Outline. This view enables you to to do the following:

- Model various deployment descriptor files.

  For information on modelling EJB projects, see "Modelling Bean Artifacts".

  For information on modelling Web projects, see "Modelling Web Artifacts".

  For information on modelling EAR projects, see "Modelling EAR Artifacts".

  For information on modelling Application Clients, see "Modelling Application Client Artifacts".

- Validate the Project. For more information on validating projects, see "Validating Projects".

- Export the Project. For more information on exporting projects, see "Exporting Projects".

- Reload the project. The Borland AppServer development plug-in for Eclipse enables you to reload the deployment descriptor tree based on the physical deployment descriptor files.

  To reload your project, in the Outline view, right-click on the project and select Reload.

- Save the project. The Borland AppServer development plug-in for Eclipse enables you to perform an implicit save operation. This option persists the data in memory to the disk.

  To save your project, in the Outline view, right-click on the project and select Save.

- Fix the Project. The Borland AppServer development plug-in for Eclipse enables you to fix projects. To fix your project, in the Outline view, right-click on the project and select Fix Project.

  When you select the Fix Project option, the Borland AppServer development plug-in performs the following tasks:

  - Adds Borland AppServer functionality to an imported project.
  - Fixes Borland AppServer functionalities for the project. For example, the Borland AppServer Builder and the Borland AppServer nature.
  - Fixed merge file entries for the project.
  - Persists the deployment descriptor data in memory to the disk.

- View Borland AppServer development plug-in for Eclipse Help. To view the Borland AppServer development plug-in for Eclipse help, in the Outline tree view, right-click on the project node and select Borland AppServer Project Help.

4

Working with Projects

This chapter explains how the Borland AppServer development development plug-in enables you to create, validate, export, reload, save, and fix projects.

Creating Projects

You can create a new project or create a project using existing source code and deployment descriptor file.

For information on creating EJB Projects, see “Creating EJB Projects”.

For information on creating Web Projects, see “Creating Web Projects”.

For information on creating EAR Projects, see “Creating EAR Projects”.

For information on creating Application Client Project, see “Creating Client Projects”.

Validating Projects

The Borland AppServer development plug-in for Eclipse enables you to validate the deployment descriptor files and verify whether the project is ready for deployment. This menu option validates Borland’s deployment descrioptor files and standard descriptor files. When you validate a project, the Validation dialog box displays errors or warnings encountered while validating the project.

To validate your project:

1. In the Project Explorer view, right-click on the project and select Borland|Validate.

   Or

   In the Outline view, right-click on the project and select Validate.

2. Look at the validation results displayed in the Validation dialog box, and click OK.

Exporting Projects

The Borland AppServer development plug-in for Eclipse enables you to export Borland AppServer's module as an archive, which include stubs and skeletons.

To export your project:

1. In the Project Explorer view, right-click on the project and select Borland|Export As.

   Or

   In the Outline view, right-click on the project and select Export.
2. Select the EJB project to export from the EJB Module drop-down list.
3. In the Destination field, enter the complete path and JAR file name for the EJB module.
4. To include source files in the exported JAR file, select the Export source files check box.
5. Optional: If you are exporting to an existing JAR file and you do not want to be warned about overwriting it, select Overwrite existing file.
6. Click Finish.

Reloading Projects

The Borland AppServer development plug-in for Eclipse enables you to reload the deployment descriptor information and rebuild the entire project based on the physical deployment descriptor files.

To reload your project, in the Outline view, right-click on the project and select Reload.

Saving Projects

The Borland AppServer development plug-in for Eclipse enables you to perform an implicit save operation. This option persists the data in memory to the disk.

To save your project, in the Outline view, right-click on the project and select Save.

Fixing Projects

The Borland AppServer development plug-in for Eclipse enables you to fix projects. When you select the Fix Project option, the Borland AppServer development plug-in performs the following tasks:

- Fixes Borland AppServer functionalities for the project. For example, the Borland AppServer Builder and the Borland AppServer nature.
- Fixed merge file entries for the project.
- Persists the deployment descriptor data in memory to the disk.

To fix your project, in the Outline view, right-click on the project and select Fix Project.

5

EJB Modelling

This chapter explains how the Borland AppServer development development plug-in for Eclipse enables you to work with EJB projects, work with beans, and model EJB artifacts.

Understanding the Project Structure

The EJB project contains the following:

- META-INF folder. This contains standard and vendor deployment descriptor files.
  - ejb-jar.xml
  - ejb-borland.xml
- XDoclet's merge files. These merge files are called by the XDoclet Builder to merge custom data to the standard deployement descriptor file (ejb-jar.xml).
  - session-beans.xml
  - entity-beans.xml
  - message-driven-beans.xml
  - relationship.xml
  - assembly-descriptor.xml

  Note

  We recommend that you do not modify these XDoclet merge files.
- Backup directory. When a user deletes an EJB, the Boroland AppServer development plug-in will back up the abstract Bean class to the Backup directory.

Working with EJB Projects

Creating EJB Projects

You can create a new EJB project or create an EJB project using existing source code and deployment descriptor file.

Creating a New EJB Project

1. Choose File|New|Project.
   The New Project dialog box appears.
2. Select EJB|BAS EJB Project and click Next.
3. In the Project name field, enter a name for this EJB project.
4. Select a Borland AppServer 6.7 runtime target from the Target Runtime drop-down list. For example, BAS v6.7.
5. Click Finish.

Creating an EJB Project Using Existing Source Code and Deployment Descriptor Files

1. Choose File|New|Project.
   The New Project dialog box appears.
2. Select EJB|BAS EJB Project and click Next.
3. In the Project name field, enter a name for this EJB project.
4. Select BAS v6.7 from the Target Runtime drop-down list.
5. If you want to add this EJB project to an EAR project, check the Add project to an EAR and select the EAR project name from the EAR Project Name drop-down list or create a new EAR project by clicking New. For more information on creating an EAR project, see “Creating EAR Projects”.
6. Click Finish.
7. Copy the existing descriptor files to the <Workspace_Home>\<projectname>\<ejbModule>\META-INF directory, where <Workspace_Home> is the Eclipse workspace directory for this project and <projectname> is the name of the project you specified in step 3, and <ejbModule> is the source directory.
8. Copy the existing source code to the <Workspace_Home>\<projectname>\<ejbModule> directory, where <Workspace_Home> is the Eclipse workspace directory for this project, <projectname> is the name of the project you specified in step 3, and <ejbModule> is the source directory.
9. To reload and build the project, select the project and choose File|Refresh.
10. In the Project Explorer view, double-click on the project you created.
11. In the Project Explorer view, double-click on the deployment descriptor that belongs to the project.
12. In the Outline view, expand the root node.
13. Right-click on the root node and select Save to ensure that the deployment decriptor information is saved into the merge file.

Working with Beans

Creating Session Beans

1. In the Project Explorer view, double-click the Deployment Descriptor node.
2. To create a new session bean, in the Outline view, right-click on the project name and select New Session Bean.
3. From the Project drop-down list, select the project that will contain the new session bean.
4. In the Folder field, enter the folder for the new bean.
5. In the Java package field, enter the package name for the new bean.

   Note

   By convention, package names should begin with a lowercase letter.
6. In the Class name field, type a name for the enterprise bean.

   Note

   By convention, class names should begin with an uppercase and you must use the word Bean as a suffix to the class name. You can use Unicode characters for the bean name, but Unicode characters are not supported for enterprise bean packages and classes associated with enterprise beans.
7. Click Next.
8. In the EJB Name field, enter the name of the enterprise bean class.
9. In the JNDI Name field, enter the logical name used by the server to locate an enterprise bean at runtime.
10. in the Display Name field, enter a short name for the enterprise bean. This is used by tools.
11. in the Description field, enter a description for the bean.
12. From the State Type drop-down list, select the state type for the new bean. For more information on updating the state type for the bean, see “Updating the Session Type for Session Beans”.
13. From the Transaction Type drop-down list, select Container.
14. Click Finish.

Creating Entity Beans

1. In the Project Explorer view, double-click the Deployment Descriptor node.
2. In the Outline view, right-click on the project name and select New Entity Bean.
3. From the Project drop-down list, select the project that will contain the new entity bean.
4. In the Folder field, enter the folder for the new bean.
5. In the Java package field, enter the package name for the new bean.

   Note

   By convention, package names should begin with a lowercase letter.

6. In the Class name field, type a name for the enterprise bean.

   Note

   By convention, class names should begin with an uppercase and you must use the word Bean as a suffix to the class name. You can use Unicode characters for the bean name, but Unicode characters are not supported for enterprise bean packages and classes associated with enterprise beans.

7. Click Next.
8. In the EJB Name field, enter the name of the enterprise bean class.
9. In the Schema field, enter a schema name to specify the abstract schema of the bean.
10. In the Display Name field, enter a short name for the enterprise bean. This name is used by tools.
11. In the description field, enter a description for the bean.
12. From the CMP Version drop-down list, select 2.x.
13. Select a Usecase for the new bean.
    a. Import attributes from table specifies that the CMP entity bean attributes will be imported from a database table.
       1. Click Next.
       2. To select an available connection definition, click a connection in the Available Connection Definitions list, and then click Next.

          To create a new JDBC connection definition, perform the following:

          - Click New.
          - In the Connection Parameters window, specify the required JDBC connection parameters on the Connection parameters page of the New Connection wizard.
          - Select a database manager, a JDBC driver, and specify other connection details. To specify JDBC connection filters, clear the Disable filter check box and specify appropriate connection filters.
          - Click Finish.

b Define new attributes specifies that the CMP entity bean attributes will be user-defined. Click Next.

1 To create a CMP attribute for the entity bean, click Add.

2 To specify a name for the attribute, click in the Name field and enter a name.

3 To specify a type for the attribute, click in the Type field and enter a type.

4 To make the attribute a key field for the entity bean, select the Primary Key check box.

5 To specify a table name for the entity bean, enter a name in the Table field.

6 To add more attributes, repeat steps 1 to 4.

14 Click Next.

15 Click Finish.

Creating Message Beans

1 In the Project Explorer view, double-click the Deployment Descriptor node.

2 To create a new message bean, in the Outline view, right-click on the project name and select New Message Bean.

3 From the Project drop-down list, select the project that will contain the new message bean.

4 In the Folder field, enter the folder for the new bean.

5 In the Java package field, enter the package name for the new bean.

Note

By convention, package names should begin with a lowercase letter.

6 In the Class name field, type a name for the enterprise bean.

Note

By convention, class names should begin with an uppercase and you must use the word Bean as a suffix to the class name. You can use Unicode characters for the bean name, but Unicode characters are not supported for enterprise bean packages and classes associated with enterprise beans.

7 Click Next.

8 In the EJB Name field, enter the name of the enterprise bean class.

9 In the JNDI Name field, enter the logical name used by the server to locate an enterprise bean at runtime.

10 in the Display Name field, enter a short name for the enterprise bean. This is used by tools.

11 in the Description field, enter a description for the bean.

12 From the State Type drop-down list, select the state type for the new bean.

13 From the Transaction Type drop-down list, select Container.

14 Click Finish.

Renaming Beans

1. In the Outline view, right-click on the session bean, message bean, or entity bean, and select Rename.
2. In the New name field, enter the new name for the bean.
3. Click OK.

Deleting Beans

1. In the Outline view, right-click on the session bean, message bean, or entity bean, and select Delete.

   The Delete Artifact dialog box appears.
2. Click Yes.

Building Beans

When you build beans individually, the XDoclet builder is forced to run so that the updated content in the abstract bean file is exposed to various interfaces and to the deployment descriptor. To build a bean, right-click on the bean and select Build.

Working with Container-managed Relationships

Creating a relationship

1. Right-click on Relationship and select New Relation.
2. In the Relation name field, enter a name for this relationship.
3. Select the appropriate EJBs in the From EJB and To EJB drop-down lists.
4. Click OK.

Defining CMR field for a bean

1. In the Outline view, expand the Relationships node.
2. Expand the EJB relation you created.
3. Click on the relationship role.
4. In the Name field, enter a name for the CMR field.
5. Select the appropriate CMR Field type from the Type drop-down list.
6. In the Description box, enter a description for the CMR field.

Configuring Table Reference

1. In the Outline view, expand the Relationships node.
2. Expand the EJB relation you created.
3. Right-click on the relationship role and select Configure Table Reference. You can perform the following actions:
   a. Create a relationship of table reference. To do this:
      1. Select the columns of the table on the left.
      2. Select the columns of the table on the right.
      3. Click Link.
      4. Click OK.
   b. Add cross table. You can use the Add Cross Table button to add an intermediate table that enables you to relate two unrelated tables. This is often the case for many-to-many relationships.
      1. Select the columns of the table on the left.
      2. Click Add Cross table.
      3. Select the appropriate table from the Cross table drop-down list.
      4. Select the columns of the table on the right.
      5. Click Link Cross Table.
      6. Click OK.
   c. Remove Link
      1. Select the link from the list.
      2. Click Remove Link.
      3. Click OK.
   d. Remove cross table
      1. Select cross table link from the list.
      2. Click Remove Cross Table.
      3. Click OK.

Creating User-defined Business Methods

Note

You can add a new method only to an an EJB that uses Xdoclet to generate source code.

1. In the Outline view, right-click on the bean and select new Method.

   The New Method dialog box appears.
2. Enter the appropriate information and click OK.

Updating the Interface Type for Entity and Session Beans

By default, when you create a new session bean, the interface type is set to remote and when you create an entity bean, the interface type is set to local.

Note

You can update the interface type only if you are using Xdoclet to generate source code.

To update the interface type for entity and session beans:

1. In the Outline view, right-click on the bean and select Interface Type.

   The Change Interface Type dialog box appears.

2. Select the appropriate interface type.
3. Click OK.

Updating the Session Type for Session Beans

By default, when you create a new session bean, the session type is set to Stateless.

To update the session type for session beans:

1. In the Outline view, right-click on the session bean and select Session Type.

   The Change Session Type dialog box appears.

2. Select the appropriate session type.
3. Click OK.

Updating the Persistence Type for Entity Beans

By default, when you create a new entity bean, the persistence type is set to Container (container-managed persistence).

To update the persistence type for entity beans:

1. In the Outline view, right-click on the entity bean and select Persistence Type.

   The Change Persistence Type dialog box appears.

2. Select the appropriate persistence type.
3. Click OK.

Modelling Bean Artifacts

Understanding Artifacts

Entity Bean Artifacts

EJB Local References

The ejb-local-ref element denotes an EJB reference that can be resolved by the EJB container locally.

```
<xsd: element name="ejb-local-ref" type="borl:ejb-local-refType" minOccurs="0"
maxOccurs="unbounded"/>

<complexType name="ejb-local-refType">
   <sequence>
      <element name="ejb-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string" minOccurs="0"/>
   </sequence>
</complexType>
```

Example

```
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>clerk</ejb-name>
            <bean-home-name>insurance/remote/clerk</bean-home-name>
            <timeout>5</timeout>
            <ejb-local-ref>
                 <ejb-ref-name>ejb/insurance/claim</ejb-ref-name>
            </ejb-local-ref>
            <resource-ref>
                 <res-ref-name>jms/insurance/ConnectionFactory</res-ref-name>
                 <jndi-name>jms/xacf</jndi-name>
            </resource-ref>
        </session>
        <entity>
              <ejb-name>claim</ejb-name>
              <bean-local-home-name>Claim</bean-local-home-name>
              ...
        </entity>
        ...
     </enterprise-beans>
</ejb-jar>
```

EJB References

This element is used to define EJB references used by the bean. Each EJB reference contains an ejb-ref-name used by the client application and its associated jndi-name.

```
<!ELEMENT ejb-ref (ejb-ref-name, jndi-name?)>
```

Example

```
<ejb-ref>
    <ejb-ref-name>ejb/Sort</ejb-ref-name>
    <jndi-name>sort</jndi-name>
</ejb-ref>
```

Message Destination Refs

This element is used to define a message destination reference, such as a JMS Queue or Topic within the context of an enterprise bean. Each message destination reference contains an message-destination-ref-name used by the bean and an associated jndi-name from which the desired object is resolved from a JNDI lookup.

```
<complexType name="message-destination-refType">
   <sequence>
      <element name="message-destination-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>

<xsd: element name="message-destination-ref" type="borl:message-destination-
refType" minOccurs="0" maxOccurs="unbounded"/>
```

Example

```
<ejb-jar>
        <enterprise-beans>
           <session>
              ...
              <message-destination-ref>
                  <message-destination-ref-name>jms/StockQueue</message-
destination-ref-name>
                  <jndi-name>jms/queues/Queue1</message-destination-type>
              </message-destination-ref>
              ...
           </session>
           ...
        </enterprise-beans>
    </ejb-jar>
```

Resource Environment Refs

This element is used to define resource environment references used by the bean. Each resource environment reference contains a res-env-ref-name used by the bean and its associated jndi-name.

```
<!ELEMENT resource-env-ref (res-env-ref-name, jndi-name?)>
```

Example

```
<resource-env-ref>
    <res-env-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-env-ref>
```

Resource Reference

This element is used to define resource references used by the bean. Each resource reference contains an res-ref-name used by the client application and its associated jndi-name.

```
<!ELEMENT resource-ref (res-ref-name, jndi-name?)>
```

Example

```
<resource-ref>
    <res-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-ref>
```

CMP Fields

Basic field mapping is accomplished using the <cmp-field> element. In this element's child nodes, you specify a field name and a corresponding column to which it maps. Many users may employ coarse-grained entity beans that implement a Java class to represent more fine-grained data. A third child node, <cmp-field-map>, defines a field map between your fine-grained class and its underlying database representation, and can be used instead of the <column-name> element.

```
<!ELEMENT cmp-field (field-name, (cmp-field-map* | column-name),property*)>
```

Example

```
<cmp-field>
   <field-name>orderNumber</field-name>
   <column-name>ORDER_NUMBER</column-name>
 </cmp-field>
```

Regenerating CMP Fields

a In the Outline view, select the entity bean node.

b Right-click on the entity bean node, and then select Extract CMP Fields from Bean Class.

This regenerates CMP fields based on the bean class.

Finders

Use this element to define the finders used by the entity bean. When you construct a finder method, you are actually constructing an SQL select statement with a where clause. The select statement includes a clause that states what records or data are to be found and returned. Under container-managed persistence, you must specify the terms of the where clause using the child nodes of <finder>.

```
<!ELEMENT finder (method-signature, where-clause, load-state?)>
```

Example

```
<finder>
    <method-signature>findByStudent(Student s)</method-signature>
    <where-clause>SELECT course_dept, course_number FROM
                  Enrollment WHERE student  = :s[ejb/Student]</where-clause>
    <load-state>False</load-state>
</finder>
```

Properties

This element is used to specify property values for various resources included in or referenced by the archive or its components. Each property entry specifies the property's name, type, and value using the appropriate sub-elements.

```
<!ELEMENT property (prop-name, prop-type, prop-value)>
```

Example

```
<property>
    <prop-name>ejb.cacheCreate</prop-name>
    <prop-type>Boolean</prop-type>
    <prop-value>false</prop-value>
</property>
```

Relationships

To specify relationships between tables, you use the <relationships> element. Within the <relationships> element, you define an <ejb-relationship-role> containing the role's source (an entity bean) and a <cmr-field> element containing the relationship. The descriptor then uses <table-ref> elements to specify relationships between two tables, a <left-table> and a <right-table>. You must observe the following cardinalities: One <ejb-relationship-role> must be defined per direction; if you have a bi-directional relationship, you must define an <ejb-relationship-role> for each bean with each referencing the other. Only one <table-ref> element is permitted per relationship. If you define a many-to-many relationship, you must also have the CMP engine create a cross-table which models a relationship between the left table and the right table. This is performed using the <cross-table> element.

```
<!ELEMENT relationships (ejb-relation+)>
```

Example

```
<relationships>
 <ejb-relation>
  <ejb-relationship-role>
   <relationship-role-source>
    <ejb-name>Customer</ejb-name>
   </relationship-role-source>
   <cmr-field>
    <cmr-field-name>specialInformation</cmr-field-name>
    <table-ref>
     <left-table>
      <table-name>CUSTOMER</table-name>
      <column-list>CUSTOMER_NO</column-list>
     </left-table>
     <right-table>
      <table-name>SPECIAL_INFO</table-name>
      <column-list>CUSTOMER_NO</column-list>
     </right-table>
    </table-ref>
   </cmr-field>
  </ejb-relationship-role>
  <ejb-relationship-role>
   <relationship-role-source>
    <ejb-name>SpecialInfo</ejb-name>
   </relationship-role-source>
  </ejb-relationship-role>
 </ejb-relation>
</relationships>
```

Tables

Specifies the name of a database table used by a CMP 2.x entity bean to populate its fields.

```
<!ELEMENT table (#PCDATA)>
```

Example

```
<table>Course</table>
```

Message Destinations

This element is used to define a message destination, such as a JMS Queue or Topic, that corresponds to a message-destination-link of one or more message-destination-ref or message-driven elements in the standard descriptor of application component. Each message destination contains an message-destination-name, that matches the message-destination-link value, and an associated jndi-name from which the destination object is resolved from a JNDI lookup.

```
<element name="message-destination" type="borl:message-destinationType"
minOccurs="0" maxOccurs="unbounded"/>

<complexType name="message-destinationType">
   <sequence>
      <element name="message-destination-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>
```

Example

```
<ejb-jar>
    ...
    <assembly-descriptor>
        ...
        <message-destination>
            <message-destination-name>myAppQueue</message-destination-name>
            <jndi-name>jms/queues/TibcoQueue1</jndi-name>
        </message-destination>
        ...
    </assembly-descriptor>
</ejb-jar>
```

Security Roles

Provides the name of a security role and (if applicable) a deployment role used by modules within the archive.

```
<!ELEMENT security-role (role-name, deployment-role?)>
```

Example

```
<security-role>
        <role-name>administrator</role-name>
        <deployment-role>administrator</deployment-role>
</security-role>
```

Interface Type

Entity beans can expose their methods with a remote interface or with a local interface. The remote interface exposes the bean's methods across the network to other, remote components. The local interface exposes the bean's methods only to local clients; that is, clients located on the same EJB container.

Note

Deployment Descriptor files are regenerated every time you update the interface type for the bean.

Session Bean Artifacts

EJB Local References

The ejb-local-ref element denotes an EJB reference that can be resolved by the EJB container locally.

```
<xsd: element name="ejb-local-ref" type="borl:ejb-local-refType" minOccurs="0"
maxOccurs="unbounded"/>

<complexType name="ejb-local-refType">
   <sequence>
      <element name="ejb-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string" minOccurs="0"/>
   </sequence>
</complexType>
```

Example

```
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>clerk</ejb-name>
            <bean-home-name>insurance/remote/clerk</bean-home-name>
            <timeout>5</timeout>
            <ejb-local-ref>
                 <ejb-ref-name>ejb/insurance/claim</ejb-ref-name>
            </ejb-local-ref>
            <resource-ref>
                 <res-ref-name>jms/insurance/ConnectionFactory</res-ref-name>
                 <jndi-name>jms/xacf</jndi-name>
            </resource-ref>
        </session>
        <entity>
              <ejb-name>claim</ejb-name>
              <bean-local-home-name>Claim</bean-local-home-name>
              ...
        </entity>
        ...
     </enterprise-beans>
</ejb-jar>
```

EJB References

This element is used to define EJB references used by the bean. Each EJB reference contains an ejb-ref-name used by the client application and its associated jndi-name.

```
<!ELEMENT ejb-ref (ejb-ref-name, jndi-name?)>
```

Example

```
<ejb-ref>
    <ejb-ref-name>ejb/Sort</ejb-ref-name>
    <jndi-name>sort</jndi-name>
</ejb-ref>
```

Message Destination Refs

This element is used to define a message destination reference, such as a JMS Queue or Topic within the context of an enterprise bean. Each message destination reference contains an message-destination-ref-name used by the bean and an associated jndi-name from which the desired object is resolved from a JNDI lookup.

```
<complexType name="message-destination-refType">
   <sequence>
      <element name="message-destination-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>

<xsd: element name="message-destination-ref" type="borl:message-destination-
refType" minOccurs="0" maxOccurs="unbounded"/>
```

Example

```
<ejb-jar>
        <enterprise-beans>
           <session>
              ...
              <message-destination-ref>
                  <message-destination-ref-name>jms/StockQueue</message-
destination-ref-name>
                  <jndi-name>jms/queues/Queue1</message-destination-type>
              </message-destination-ref>
              ...
           </session>
           ...
        </enterprise-beans>
    </ejb-jar>
```

Resource Environment Refs

This element is used to define resource environment references used by the bean. Each resource environment reference contains a res-env-ref-name used by the bean and its associated jndi-name.

```
<!ELEMENT resource-env-ref (res-env-ref-name, jndi-name?)>
```

Example

```
<resource-env-ref>
    <res-env-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-env-ref>
```

Resource Reference

This element is used to define resource references used by the bean. Each resource reference contains an res-ref-name used by the client application and its associated jndi-name.

```
<!ELEMENT resource-ref (res-ref-name, jndi-name?)>
```

Example

```
<resource-ref>
    <res-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-ref>
```

Tables

Specifies the name of a database table used by a CMP 2.x entity bean to populate its fields.

```
<!ELEMENT table (#PCDATA)>
```

Example

```
<table>Course</table>
```

Message Destinations

This element is used to define a message destination, such as a JMS Queue or Topic, that corresponds to a message-destination-link of one or more message-destination-ref or message-driven elements in the standard descriptor of application component. Each message destination contains an message-destination-name, that matches the message-destination-link value, and an associated jndi-name from which the destination object is resolved from a JNDI lookup.

```
<element name="message-destination" type="borl:message-destinationType"
minOccurs="0" maxOccurs="unbounded"/>

<complexType name="message-destinationType">
   <sequence>
      <element name="message-destination-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>
```

Example

```
<ejb-jar>
    ...
    <assembly-descriptor>
        ...
        <message-destination>
            <message-destination-name>myAppQueue</message-destination-name>
            <jndi-name>jms/queues/TibcoQueue1</jndi-name>
        </message-destination>
        ...
    </assembly-descriptor>
</ejb-jar>
```

Security Roles

Provides the name of a security role and (if applicable) a deployment role used by modules within the archive.

```
<!ELEMENT security-role (role-name, deployment-role?)>
```

Example

```
<security-role>
        <role-name>administrator</role-name>
        <deployment-role>administrator</deployment-role>
</security-role>
```

Interface Type

Session beans can expose their methods with a remote interface or with a local interface. The remote interface exposes the bean's methods across the network to other, remote components. The local interface exposes the bean's methods only to local clients; that is, clients located on the same EJB container.

Note

Deployment Descriptor files are regenerated every time you update the interface type for the bean.

Properties

This element is used to specify property values for various resources included in or referenced by the archive or its components. Each property entry specifies the property's name, type, and value using the appropriate sub-elements.

```
<!ELEMENT property (prop-name, prop-type, prop-value)>
```

Example

```
<property>
    <prop-name>ejb.security.transportType</prop-name>
    <prop-type>Enumerated</prop-type>
    <prop-value>CLEAR_ONLY</prop-value>
</property>
```

Message Bean Artifacts

EJB Local References

The ejb-local-ref element denotes an EJB reference that can be resolved by the EJB container locally.

```
<xsd: element name="ejb-local-ref" type="borl:ejb-local-refType" minOccurs="0"
maxOccurs="unbounded"/>

<complexType name="ejb-local-refType">
   <sequence>
      <element name="ejb-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string" minOccurs="0"/>
   </sequence>
</complexType>
```

Example

```
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>clerk</ejb-name>
            <bean-home-name>insurance/remote/clerk</bean-home-name>
            <timeout>5</timeout>
            <ejb-local-ref>
                 <ejb-ref-name>ejb/insurance/claim</ejb-ref-name>
            </ejb-local-ref>
            <resource-ref>
                 <res-ref-name>jms/insurance/ConnectionFactory</res-ref-name>
                 <jndi-name>jms/xacf</jndi-name>
            </resource-ref>
        </session>
        <entity>
              <ejb-name>claim</ejb-name>
              <bean-local-home-name>Claim</bean-local-home-name>
              ...
        </entity>
        ...
     </enterprise-beans>
</ejb-jar>
```

EJB References

This element is used to define EJB references used by the bean. Each EJB reference contains an ejb-ref-name used by the client application and its associated jndi-name.

```
<!ELEMENT ejb-ref (ejb-ref-name, jndi-name?)>
```

Example

```
<ejb-ref>
    <ejb-ref-name>ejb/Sort</ejb-ref-name>
    <jndi-name>sort</jndi-name>
</ejb-ref>
```

Message Destination Refs

This element is used to define a message destination reference, such as a JMS Queue or Topic within the context of an enterprise bean. Each message destination reference contains an message-destination-ref-name used by the bean and an associated jndi-name from which the desired object is resolved from a JNDI lookup.

```
<complexType name="message-destination-refType">
   <sequence>
      <element name="message-destination-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>

<xsd: element name="message-destination-ref" type="borl:message-destination-
refType" minOccurs="0" maxOccurs="unbounded"/>
```

Example

```
<ejb-jar>
        <enterprise-beans>
           <session>
              ...
              <message-destination-ref>
                  <message-destination-ref-name>jms/StockQueue</message-
destination-ref-name>
                  <jndi-name>jms/queues/Queue1</message-destination-type>
              </message-destination-ref>
              ...
           </session>
           ...
        </enterprise-beans>
    </ejb-jar>
```

Resource Environment Refs

This element is used to define resource environment references used by the bean. Each resource environment reference contains a res-env-ref-name used by the bean and its associated jndi-name.

```
<!ELEMENT resource-env-ref (res-env-ref-name, jndi-name?)>
```

Example

```
<resource-env-ref>
    <res-env-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-env-ref>
```

Resource Reference

This element is used to define resource references used by the bean. Each resource reference contains an res-ref-name used by the client application and its associated jndi-name.

```
<!ELEMENT resource-ref (res-ref-name, jndi-name?)>
```

Example

```
<resource-ref>
    <res-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-ref>
```

Tables

Specifies the name of a database table used by a CMP 2.x entity bean to populate its fields.

```
<!ELEMENT table (#PCDATA)>
```

Example

```
<table>Course</table>
```

Message Destinations

This element is used to define a message destination, such as a JMS Queue or Topic, that corresponds to a message-destination-link of one or more message-destination-ref or message-driven elements in the standard descriptor of application component. Each message destination contains an message-destination-name, that matches the message-destination-link value, and an associated jndi-name from which the destination object is resolved from a JNDI lookup.

```
<element name="message-destination" type="borl:message-destinationType"
minOccurs="0" maxOccurs="unbounded"/>

<complexType name="message-destinationType">
   <sequence>
      <element name="message-destination-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>
```

Example

```
<ejb-jar>
    ...
    <assembly-descriptor>
        ...
        <message-destination>
            <message-destination-name>myAppQueue</message-destination-name>
            <jndi-name>jms/queues/TibcoQueue1</jndi-name>
        </message-destination>
        ...
    </assembly-descriptor>
</ejb-jar>
```

Security Roles

Provides the name of a security role and (if applicable) a deployment role used by modules within the archive.

```
<!ELEMENT security-role (role-name, deployment-role?)>
```

Example

```
<security-role>
        <role-name>administrator</role-name>
        <deployment-role>administrator</deployment-role>
</security-role>
```

Properties

This element is used to specify property values for various resources included in or referenced by the archive or its components. Each property entry specifies the property's name, type, and value using the appropriate sub-elements.

```
<!ELEMENT property (prop-name, prop-type, prop-value)>
```

Example

```
<property>
    <prop-name>ejb.mdb.maxMessagesPerServerSession</prop-name>
    <prop-type>Integer</prop-type>
    <prop-value>1</prop-value>
</property>
```

Creating Instances of Artifacts

1. In the Outline view, right-click on the artifact and select New <artifact_name>, where artifact_name is the name of the artifact you are creating. For example, if you want to create a new EJB Reference, right-click on EJB Reference and select New EJB Reference.

   A dialog box appears.
2. Enter a name for the new artifact.
3. Click OK.

Renaming Instances of Artifacts

1. In the Outline view, expand the artifact node that contains the instance you want to rename. For example, if you want to rename an instance of the EJB References artifact, expand the EJB References node.
2. Right-click on the artifact instance and select Rename. For example, you have an instance named EJB_Reference_1 under the EJB References artifact node. You need to right-click on EJB_Reference_1 and select Rename.

   A dialog box appears.
3. Enter the new name.
4. Click OK.

Deleting Entity Beans

1. In the Outline view, right-click on the entity bean and select Delete.

   The Delete Artifact dialog box appears.
2. Click Yes.

Deleting Artifacts

1. In the Outline view, expand the artifact node that contains the instance you want to delete. For example, if you want to delete an instance of the EJB References artifact, expand the EJB References node.
2. Right-click on the artifact instance and select Delete. For example, you have an instance named EJB_Reference_1 under the EJB References artifact node. You need to right-click on EJB_Reference_1 and select Delete.

   The Delete Artifact message box appears.
3. Click Yes.

6

Web Modelling

This chapter explains how the Borland AppServer development plug-in for Eclipse enables you to work with Web projects, work with servlets, and model Web artifacts.

Understanding the Project Structure

The Web project contains the following:

- The WebContent\WEB-INF directory. This contains standard and vendor deployment descriptor files.
  - web.xml
  - web-borland.xml

Working with Web Projects

Creating Web Projects

You can create a new Web project or create a Web project using existing source code and deployment descriptor file.

Creating a New Web Project

1. Choose File|New|Project.
   The New Project dialog box appears.
2. Select Web|BAS Dynamic Web Project and click Next.
3. In the Project name field, enter a name for this Web project.
4. Select a Borland AppServer 6.7 runtime target from the Target Runtime drop-down list. For example, BAS v6.7.
5. Click Finish.

Creating a Web Project Using Existing Source Code and Deployment Descriptor Files

1. Choose File|New|Project.
   The New Project dialog box appears.
2. Select Web|BAS Dynamic Web Project and click Next.
3. In the Project name field, enter a name for this Web project.
4. Select a Borland AppServer 6.7 runtime target from the Target Runtime drop-down list. For example, BAS v6.7.
5. Click Finish.
6. Copy the existing descriptor files to the <Workspace_Home>\<projectname>\WebContent\WEB-INF directory, where <Workspace_Home> is the Eclipse workspace directory for this project and <projectname> is the name of the project you specified in step 3.
7. Copy the existing JSP and HTML source to the <Workspace_Home>\<projectname>\WebContent directory, where <Workspace_Home> is the Eclipse workspace directory for this project and <projectname> is the name of the project you specified in step 3.

   Copy the existing Java source to the <Workspace_Home>\<projectname>\Src directory, where <Workspace_Home> is the Eclipse workspace directory for this project and <projectname> is the name of the project you specified in step 3.
8. To reload and build the project, select the project and choose File|Refresh.
9. In the Project Explorer view, double-click on the project you created.
10. In the Project Explorer view, double-click on the deployment descriptor that belongs to the project.
11. In the Outline view, expand the root node.
12. Right-click on the root node and select Save to ensure that the deployment decriptor information is saved into the merge file.

Working with Servlets

Creating Servlets

1. In the Project Explorer view, double-click the Deployment Descriptor node.
2. In the Outline view, right-click on the project name and select New Servlet.
3. From the Project drop-down list, select the project that will contain the new servlet.
4. In the Folder field, enter the folder where the servlet class will be placed.
5. In the Java package field, enter the package name for the new servlet.

   Note

   By convention, package names should begin with a lowercase letter.
6. In the Class name field, type the class name for the servlet.

   Note

   By convention, class names should begin with an uppercase.
7. In the Superclass field, enter the super class for the servlet class.
8. Click Next.
9. Enter the initialization parameters of the servlet as name-value pairs.
10. In the URL Mappings box, enter the URL string to be mapped with the servlet.
11. Click Next.
12. Click Finish.

Adding Browser JSP File

1. In the Project Explorer view, right-click on the WebContent folder and select New| JSP.

   The New Java Server Page window appears with your folder selected
2. In the File name field, enter the file name of the JSP file.

   Note

   Ensure that you include the JSP extension in the file name.
3. To accept the defaults associated with a new JSP file, click Finish.

   To link to a file in the file system and specify path variables, click Advanced.

   To use a template file for the initial content of your JSP page, perform steps 4 to 6.
4. Click Next.

   The Select JSP Template window appears.
5. Select the Use JSP Template check box, and then select one of the sample templates.
6. Click Finish.

Renaming Servlets

1 In the Outline view, right-click on the servlet and select Rename.

2 In the New name field, enter the new name for the bean.

3 Click OK.

Deleting Servlets

1 In the Outline view, right-click on the servlet and select Delete.

The Delete Artifact dialog box appears.

2 Click Yes.

Modelling Web Artifacts

Understanding Artifacts

EJB References

This element is used to define EJB references used by the web application. Each EJB reference contains an ejb-ref-name used by the application and its associated jndi-name.

```
<!ELEMENT ejb-ref (ejb-ref-name, jndi-name?)>
```

Example

```
<ejb-ref>
    <ejb-ref-name>ejb/Sort</ejb-ref-name>
    <jndi-name>sort</jndi-name>
</ejb-ref>
```

Resource Environment Refs

This element is used to map a resource environment reference used by the web application to a name in JNDI. Each resource environment reference contains a res-env-ref-name used by the bean and its associated jndi-name.

```
<!ELEMENT resource-env-ref (res-env-ref-name, jndi-name?)>
```

Example

```
<resource-env-ref>
    <res-env-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-env-ref>
```

Resource References

This element is used to define resource references used by the web application. Each resource reference contains an res-ref-name used by the application and its associated jndi-name.

```
<!ELEMENT resource-ref (res-ref-name, jndi-name?)>
```

Example

```
<resource-ref>
    <res-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-ref>
```

Message Destination Refs

This element is used to define a message destination reference, such as a JMS Queue or Topic. Each message destination reference contains an message-destination-ref-name used by the web application and an associated jndi-name.

```
<element name="message-destination-ref" type="borl:message-destination-refType"
minOccurs="0" maxOccurs="unbounded"/>

<complexType name="message-destination-refType">
   <sequence>
      <element name="message-destination-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>
```

Example

```
<web-app>
        ...
        <message-destination-ref>
           <message-destination-ref-name>jms/StockQueue</message-destination-
ref-name>
           <jndi-name>jms/queues/Queue1</message-destination-type>
        </message-destination-ref>
        ...
</web-app>
```

Message Destinations

This element is used to define a message destination, such as a JMS Queue or Topic, that corresponds to message-destination-link of one or more message-destination-ref elements in the web application. Each message destination contains an message-destination-name, that matches the message-destination-link value, and an associated jndi-name.

```
<element name="message-destination" type="borl:message-destinationType"
minOccurs="0" maxOccurs="unbounded"/>

<complexType name="message-destinationType">
   <sequence>
      <element name="message-destination-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>
```

Example

```
<web-app>
    ...
    <message-destination>
        <message-destination-name>myAppQueue</message-destination-name>
        <jndi-name>jms/queues/TibcoQueue1</jndi-name>
    </message-destination>
    ...
</web-app>
```

Web Deploy Paths

Tomcat's server.xml file can define one or more hosts under one or more engines, which themselves are under a given service. If you would like specify exactly where to deploy the web application under the Tomcat container, use this element.

```
<web-deploy-path> <!ELEMENT web-deploy-path (service, engine, host)>
```

Example

```
<web-deploy-path>
    <service>tomcatX</service>
    <engine>cyrpi</engine>
    <host>it3</host>
</web-deploy-path>
```

Security Roles

Maps a role for the web application (found in web.xml) to a deployment-role in the Borland Enterprise Server.

```
<!ELEMENT security-role (role-name, deployment-role?)>
```

Example

```
<security-role>
        <role-name>administrator</role-name>
        <deployment-role>administrator</deployment-role>
</security-role>
```

Creating Instances of Artifacts

1. In the Outline view, right-click on the artifact and select New <artifact_name>, where artifact_name is the name of the artifact you are creating. For example, if you want to create a new EJB Reference, right-click on EJB Reference and select New EJB Reference.

   A dialog box appears.

2. Enter a name for the new artifact.
3. Click OK.

Renaming Instances of Artifacts

1. In the Outline view, expand the artifact node that contains the instance you want to rename. For example, if you want to rename an instance of the EJB References artifact, expand the EJB References node.
2. Right-click on the artifact instance and select Rename. For example, you have an instance named EJB_Reference_1 under the EJB References artifact node. You need to right-click on EJB_Reference_1 and select Rename.

   A dialog box appears.

3. Enter the new name.
4. Click OK.

Deleting Artifacts

1. In the Outline view, expand the artifact node that contains the instance you want to delete. For example, if you want to delete an instance of the EJB References artifact, expand the EJB References node.
2. Right-click on the artifact instance and select Delete. For example, you have an instance named EJB_Reference_1 under the EJB References artifact node. You need to right-click on EJB_Reference_1 and select Delete.

   The Delete Artifact message box appears.

3. Click Yes.

7

Configuring the Borland AppServer Development Plug-in in Eclipse

This chapter explains how to configure the Borland AppServer (BAS) development plug-in in Eclipse.

Adding Borland AppServer as an Installed Server Runtime Environment

After you install the Borland AppServer development plug-in for Eclipse, you need to add Borland AppServer as an installed server runtime environment and then define a new server in Eclipse.

To add Borland AppServer as an installed server runtime environment:

1. In Eclipse, choose Window|Preferences.
   The Preferences dialog box appears.
2. In the left pane, select Server|Installed Runtimes.
3. Click Add.
   The New Server Runtime dialog box.
4. Select Borland|BAS v6.7 and click Next.
5. In the Borland AppServer Install Directory field, enter the path where you installed Borland AppServer or click Browse to browse for the path.
6. Ensure that the values specified for all the fields are correct and then click Finish.
7. BAS v6.7 is listed in the Installed Server Runtime Environments list. Click OK.

Defining a New Server in Eclipse

Defining Borland AppServer as a server in Eclipse enables you to start and stop Borland AppServer from Eclipse.

To define a new server in Eclipse:

1. Choose File|New|Other.
   The New dialog box appears.
2. Select Server|Server and click Next.
3. Select Borland|BAS v6.7 and click Finish.

Starting and Stopping Local Borland AppServer in Eclipse

1. Switch to the J2EE perpective view.
2. Switch to the Servers view.
3. To start the Borland AppServer server, in the Servers view, right-click the Borland AppServer server and select Start.
   To stop the Borland AppServer server, in the Servers view, right-click the Borland AppServer server and select Stop.

8

EAR Modelling

This chapter explains how the Borland AppServer development plug-in for Eclipse enables you to work with EAR projects, work with module dependencies, and model EAR artifacts.

Working with EAR Projects

Creating EAR Projects

1. Choose File|New|Project.
   The New Project dialog box appears.
2. Select J2EE|BAS Enterprise Application Project and click Next.
3. In the Project name field, enter a name for this EAR project.
4. Select a Borland AppServer 6.7 runtime target from the Target Runtime drop-down list. For example, BAS v6.7.
5. Click Next.
   The Project Facets screen appears.
6. Click Next.
7. Select, deselect, or add J2EE modules for this new EAR project.
8. Click Finish.

Adding J2EE Module Dependencies

1. In the Project Explorer view, right-click the EAR Project and select Properties.
2. From the left pane, select J2EE Module Dependencies.
3. Select, deselect, or add module dependencies.
4. Click Apply.
5. Click OK.

Modelling EAR Artifacts

Understanding Artifacts

Module

This element represents a collection of one or more components that execute in the same container type or deployment descriptors of that type. The module element must contain one of the sub-elements ejb, java, or web; and hosts sub-element.

```
<xsd:element name="module" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:choice>
            <xsd:element name="connector" type="xsd:string"/>
            <xsd:element name="ejb" type="xsd:string"/>
            <xsd:element name="java" type="xsd:string"/>
            <xsd:element name="web">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="web-uri" type="xsd:string"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:choice>
            <xsd:element name="hosts" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Example

```
<application>
   <module>
      <ejb>my-ejb.jar</ejb>
   </module>
   <module>
      <web>
         <web-uri>myweb.war</web-uri>
      </web>
   </module>
   ...
</application>
```

Property

This element is used to specify property values necessary for the application at runtime. Each property entry specifies the property's name, type, and value using the appropriate sub-elements.

```
<xsd:element name="property" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="prop-name" type="xsd:string"/>
         <xsd:element name="prop-type" type="xsd:string" minOccurs="0"/>
         <xsd:element name="prop-value" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Example

```
<application>
   <module>
      <ejb>my-ejb.jar</ejb>
   </module>
   <module>
      <web>
         <web-uri>myweb.war</web-uri>
      </web>
   </module>
   <property>
      <prop-name>vbroker.security.disable</prop-name>
      <prop-type>security</prop-type>
      <prop-value>false</prop-value>
   </property>
</application>
```

Security Roles

Maps a role for the application (found in application.xml) to a deployment-role in the Borland AppServer.

```
<xsd:element name="security-role" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="role-name" type="xsd:string"/>
         <xsd:element name="deployment-role" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Example

```
<security-role>
   <role-name>administrator</role-name>
   <deployment-role>administrator</deployment-role>
</security-role>
```

Creating Instances of Artifacts

1. In the Outline view, right-click on the artifact and select New <artifact_name>, where artifact_name is the name of the artifact you are creating. For example, if you want to create a new Security Role, right-click on Security Role and select New Security Role.

   A dialog box appears.
2. Enter a name for the new artifact.
3. Click OK.

Renaming Instances of Artifacts

1. In the Outline view, expand the artifact node that contains the instance you want to rename. For example, if you want to rename an instance of the Security Role artifact, expand the Security Role node.
2. Right-click on the artifact instance and select Rename. For example, you have an instance named Security_Role_1 under the Security Role artifact node. You need to right-click on Security_Role_1 and select Rename.

   A dialog box appears.
3. Enter the new name.
4. Click OK.

Deleting Entity Beans

1. In the Outline view, right-click on the entity bean and select Delete.

   The Delete Artifact dialog box appears.
2. Click Yes.

Deleting Artifacts

1. In the Outline view, expand the artifact node that contains the instance you want to delete. For example, if you want to delete an instance of the Security Role artifact, expand the Security Role node.
2. Right-click on the artifact instance and select Delete. For example, you have an instance named Security_Role_1 under the Security Role artifact node. You need to right-click on Security_Role_1 and select Delete.

   The Delete Artifact message box appears.
3. Click Yes.

9

Application Client Modelling

This chapter explains how the Borland AppServer development plug-in for Eclipse enables you to work with Application Client projects and model Application Client artifacts.

Working with Application Client Projects

Creating Client Projects

1. Choose File|New|Project.
   The New Project dialog box appears.
2. Select J2EE|BAS Application Client Project and click Next.
3. In the Project name field, enter a name for this Application Client project.
4. Select a Borland AppServer 6.7 runtime target from the Target Runtime drop-down list. For example, BAS v6.7.
5. Click Next.
   The Project Facets screen appears.
6. Click Next.
7. In the Source Folder field, enter the source folder name.
8. Click Finish.

Note

If you want to run the client application, for example, a simple EJB client application, ensure that you include the server runtime stubs, for example server.jar, in the client project.

Creating a Client Project Using Existing Source Code and Deployment Descriptor Files

1. Choose File|New|Other.
   The New Project dialog box appears.
2. Select J2EE|BAS Application Client Project and click Next.
3. In the Project name field, enter a name for this EJB project.
4. Select BAS v6.7 from the Target Runtime drop-down list.
5. If you want to add this EJB project to an EAR project, check the Add project to an EAR and select the EAR project name from the EAR Project Name drop-down list or create a new EAR project by clicking New. For more information on creating an EAR project, see “Creating EAR Projects”.
6. Click Finish.
7. Copy the existing descriptor files to the <Workspace_Home>\<projectname>\<clientModule>\META-INF directory, where <Workspace_Home> is the Eclipse workspace directory for this project and <projectname> is the name of the project you specified in step 3, and <clientModule> is the source directory.
8. Copy the existing source code to the <Workspace_Home>\<projectname>\<clientModule> directory, where <Workspace_Home> is the Eclipse workspace directory for this project, <projectname> is the name of the project you specified in step 3, and <clientModule> is the source directory.
9. To reload and build the project, select the project and choose File|Refresh.
10. In the Project Explorer view, double-click on the project you created.
11. In the Project Explorer view, double-click on the deployment descriptor that belongs to the project.
12. In the Outline view, expand the root node.
13. Right-click on the root node and select Save to ensure that the deployment decriptor information is saved into the merge file.

Modelling Application Client Artifacts

Understanding Artifacts

EJB References

This element is used to define EJB references used by the client. Each EJB reference contains an ejb-ref-name used by the client application and its associated jndi-name (if applicable).

```
<xsd: element name="ejb-ref" type="borl:ejb-refType" minOccurs="0"
maxOccurs="unbounded"/>

<xsd:complexType name="ejb-refType">
  <xsd:sequence>
   <xsd:element name="ejb-ref-name" type="xsd:string"/>
   <xsd:element name="jndi-name" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

Example

```
<ejb-ref>
    <ejb-ref-name>ejb/Sort</ejb-ref-name>
    <jndi-name>sort</jndi-name>
</ejb-ref>
```

Message Destination Refs

This element is used to define a message destination reference, such as a JMS Queue or Topic. Each message destination reference contains an message-destination-ref-name used by the client application and an associated jndi-name.

```
<xsd:element name="message-destination-ref" type="borl:message-destination-
refType" minOccurs="0" maxOccurs="unbounded"/>

<xsd:complexType name="message-destination-refType">
  <xsd:sequence>
   <xsd:element name="message-destination-ref-name" type="xsd:string"/>
   <xsd:element name="jndi-name" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

Example

```
<application-client>
        ...
        <message-destination-ref>
           <message-destination-ref-name>jms/StockQueue</message-destination-
ref-name>
           <jndi-name>jms/queues/Queue1</message-destination-type>
        </message-destination-ref>
        ...
</application-client>
```

Message Destinations

This element is used to define a message destination, such as a JMS Queue or Topic, that corresponds to message-destination-link of one or more message-destination-ref elements in the application client. Each message destination contains an message-destination-name, that matches the message-destination-link value, and an associated jndi-name.

```
<xsd:element name="message-destination" type="borl:message-destinationType"
minOccurs="0" maxOccurs="unbounded"/>

<xsd:complexType name="message-destinationType">
  <xsd:sequence>
   <xsd:element name="message-destination-name" type="xsd:string"/>
   <xsd:element name="jndi-name" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

Example

```
<application-client>
        ...
        <message-destination>
           <message-destination-name>myAppQueue</message-destination-name>
           <jndi-name>jms/queues/TibcoQueue</jndi-name>
        </message-destination>
        ...
</application-client>
```

Resource Environment Refs

This element is used to define resource environment references used by the client. Each resource environment reference contains a resource-env-ref-name used by the client application and its associated jndi-name (if applicable). The resource-env-ref-name element uniquely identifies a resource environment reference from the standard deployment descriptor.

```
<element name="resource-env-ref" type="borl:resource-env-refType" minOccurs="0"
maxOccurs="unbounded"/>

<complexType name="resource-env-refType">
   <sequence>
      <element name="resource-env-ref-name" type="xsd:string"/>
      <element ref="borl:jndi-name"/>
   </sequence>
</complexType>
```

Example

```
<application-client>
        ...
        <resource-env-ref>
            <resource-env-ref-name>jms/StockQueue</resource-env-ref-name>
            <jndi-name>jms/Queue1</jndi-name>
        </resource-env-ref>
        ...
</application-client>
```

Resource References

This element is used to define resource references used by the client. Each resource reference contains an res-ref-name used by the client application and its associated jndi-name (if applicable). The res-ref-name element uniquely identifies a resource reference from the standard deployment descriptor.

```
<xsd:element name="resource-ref" type="borl:resource-refType" minOccurs="0"
maxOccurs="unbounded"/>

<xsd:complexType name="resource-refType">
  <xsd:sequence>
   <xsd:element name="res-ref-name" type="xsd:string"/>
   <xsd:element name="jndi-name" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

Example

```
<application-client>
     ...
        <resource-ref>
           <res-ref-name>jdbc/SavingsDataSource</res-ref-name>
           <jndi-name>jdbc/datasources/Oracle</jndi-name>
        </resource-ref>
  ...
</application-client>
```

Creating Instances of Artifacts

1. In the Outline view, right-click on the artifact and select New <artifact_name>, where artifact_name is the name of the artifact you are creating. For example, if you want to create a new EJB Reference, right-click on EJB Reference and select New EJB Reference.

   A dialog box appears.

2. Enter a name for the new artifact.
3. Click OK.

Renaming Instances of Artifacts

1. In the Outline view, expand the artifact node that contains the instance you want to rename. For example, if you want to rename an instance of the EJB References artifact, expand the EJB References node.
2. Right-click on the artifact instance and select Rename. For example, you have an instance named EJB_Reference_1 under the EJB References artifact node. You need to right-click on EJB_Reference_1 and select Rename.

   A dialog box appears.

3. Enter the new name.
4. Click OK.

Deleting Entity Beans

1. In the Outline view, right-click on the entity bean and select Delete.

   The Delete Artifact dialog box appears.

2. Click Yes.

Deleting Artifacts

1. In the Outline view, expand the artifact node that contains the instance you want to delete. For example, if you want to delete an instance of the EJB References artifact, expand the EJB References node.
2. Right-click on the artifact instance and select Delete. For example, you have an instance named EJB_Reference_1 under the EJB References artifact node. You need to right-click on EJB_Reference_1 and select Delete.

   The Delete Artifact message box appears.

3. Click Yes.

10

Deploying Borland AppServer Projects from Eclipse

This chapter explains how to deploy Borland AppServer projects from Eclipse.

Deploying Projects to Borland AppServer

The Borland AppServer development plug-in enables you to deploy Borland AppServer projects, which you create in Eclipse, to the local and the remote Borland AppServer.

Configuring Deployments Settings

1. Right-click on the Borland AppServer project you want to deploy and select Properties.
   The Properties dialog box appears.
2. In the left pane, select Borland AppServer Server Settings.
3. Set the server name , deploy configuration name, and deploy partition name to match that of the server. The server can be running remotely or on the local machine.
4. Set the management port to the port that is set for the server.
5. Click OK.

Deploying Projects to Borland AppServer

1. Save the project so that the information is saved in the appropriate descriptor file.
2. Right-click on the Borland AppServer project you want to deploy and select Borland| Deploy.
   The Deploy to Borland AppServer dialog box appears.
3. In the left pane, double-click on the projects that you want to deploy to the Borland AppServer.
4. Click Next.
5. Click Deploy.

11

Debbuging in WTP

This chapter describes the steps to set up debugging environments for Java clients talking to J2EE modules deployed in Borland AppServer. Note that debugging server code, for example, servlets or beans, are not supported in this release.

Setting Up Client Debugging in Eclipse

1. Select Run|Debug.
2. Right-click on Remote Java Application and select New.
3. Click the Arguments tab.
4. Add the following line in VM Arguments:
   `Dvbroker.agent.port=<your_osagent_port> -Djava.endorsed.dirs="<path_to_bas/lib/endorsed>"`

Running the Client in Debug Mode

1. Select Run|Debug.
2. Double-click the client profile.

Index

Symbols

A

B

C

D

E

I

J

M

P

R

S

T

U

V

W