Developing Applications for J2EE™ Servers

JBuilder® 2005

Borland®
Excellence Endures™

Borland Software Corporation
100 Enterprise Way
Scotts Valley, California 95066-3249
www.borland.com

Printed in the U.S.A.

JB2005j2eeapps 3E4R0804
0405060708-9 8 7 6 5 4 3 2 1
PDF

Contents

Chapter 1

Introduction

Developing Applications for J2EE Servers introduces you to the various technologies that make up the Java™ 2 Platform, Enterprise Edition (J2EE™), explains why they are important, and describes how you can use JBuilder to create J2EE applications that target your application server. You'll learn how to configure JBuilder to work with your server, and how to accomplish the essential programming tasks of running your applications, debugging them remotely, and deploying them to your server.

Documentation conventions

The Borland documentation for JBuilder uses the typefaces and symbols described in the following table to indicate special text.

Table 1.1 Typeface and symbol conventions

Typeface	Meaning
Bold	Bold is used for java tools, bmj (Borland Make for Java), bcj (Borland Compiler for Java), and compiler options. For example: **javac**, **bmj**, **-classpath**.
Italics	Italicized words are used for new terms being defined, for book titles, and occasionally for emphasis.
Keycaps	This typeface indicates a key on your keyboard, such as "Press *Esc* to exit a menu."
`Monospaced type`	Monospaced type represents the following: ■ text as it appears onscreen ■ anything you must type, such as "Type `Hello World` in the Title field of the Application wizard." ■ file names ■ path names ■ directory and folder names ■ commands, such as `SET PATH` ■ Java code ■ Java data types, such as `boolean`, `int`, and `long`. ■ Java identifiers, such as names of variables, classes, package names, interfaces, components, properties, methods, and events ■ argument names ■ field names ■ Java keywords, such as `void` and `static`

Table 1.1 Typeface and symbol conventions (continued)

Typeface	Meaning
[ ]	Square brackets in text or syntax listings enclose optional items. Do not type the brackets.
< >	Angle brackets are used to indicate variables in directory paths, command options, and code samples. JDK 5.0 uses angle brackets to denote generics. For example, `<filename>` may be used to indicate where you need to supply a file name (including file extension), and <username> typically indicates that you must provide your user name. When replacing variables in directory paths, command options, and code samples, replace the entire variable, including the angle brackets (< >). For example, you would replace `<filename>` with the name of a file, such as `employee.jds`, and omit the angle brackets. See “Using command-line tools” in *Building Applications with JBuilder* for more information. **Note:** Angle brackets are used in HTML, XML, JSP, and other tag-based files to demarcate document elements, such as <font color=red> and <ejb-jar>. The following convention describes how variable strings are specified within code samples that are already using angle brackets for delimiters.
Italics, serif	This formatting is used to indicate variable strings within code samples that are already using angle brackets as delimiters. For example, `<url="jdbc:borland:`*jbuilder*`\\samples\\guestbook.jds">`
...	In code examples, an ellipsis (…) indicates code that has been omitted from the example to save space and improve clarity. On a button, an ellipsis indicates that the button links to a selection dialog box.

JBuilder is available on multiple platforms. See the following table for a description of platform conventions used in the documentation.

Table 1.2 Platform conventions

Item	Meaning
Paths	Directory paths in the documentation are indicated with a forward slash (/). For Windows platforms, use a backslash (\).
Home directory	The location of the standard home directory varies by platform and is indicated with a variable, `<home>`. ■ For UNIX, Linux, and OS X, the home directory can vary. For example, it could be `/user/<username>` or `/home/<username>` ■ For Windows NT, the home directory is `C:\Winnt\Profiles\<username>` ■ For Windows 2000 and XP, the home directory is `C:\Documents and Settings\<username>`
Screen shots	Screen shots reflect the Borland Look & Feel on various platforms.

Developer support and resources

Borland provides a variety of support options and information resources to help developers get the most out of their Borland products. These options include a range of Borland Technical Support programs, as well as free services on the Internet, where you can search our extensive information base and connect with other users of Borland products.

Contacting Borland Developer Support

Borland offers several support programs for customers and prospective customers. You can choose from several categories of support, ranging from free support upon installation of the Borland product, to fee-based consultant-level support and extensive assistance.

For more information about Borland's developer support services, see our web site at `http://www.borland.com/devsupport/`, call Borland Assist at (800) 523-7070, or contact our Sales Department at (831) 431-1064.

When contacting support, be prepared to provide complete information about your environment, the version of the product you are using, and a detailed description of the problem.

For support on third-party tools or documentation, contact the vendor of the tool.

Online resources

You can get information from any of these online sources:

World Wide Web	`http://www.borland.com/` `http://info.borland.com/techpubs/jbuilder/`
Electronic newsletters	To subscribe to electronic newsletters, use the online form at: `http://www.borland.com/products/newsletters/index.html`

World Wide Web

Check the JBuilder page of the Borland website, `www.borland.com/jbuilder`, regularly. This is where the Java Products Development Team posts white papers, competitive analyses, answers to frequently asked questions, 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/jbuilder/` (updated software and other files)
- `http://info.borland.com/techpubs/jbuilder/` (updated documentation and other files)
- `http://bdn.borland.com/` (contains our web-based news magazine for developers)

Borland newsgroups

When you register JBuilder you can participate in many threaded discussion groups devoted to JBuilder. The Borland newsgroups provide a means for the global community of Borland customers to exchange tips and techniques about Borland products and related tools and technologies.

You can find user-supported newsgroups for JBuilder and other Borland products at `http://www.borland.com/newsgroups/`.

Usenet newsgroups

The following Usenet groups are devoted to Java and related programming issues:

- news:comp.lang.java.advocacy
- news:comp.lang.java.announce
- news:comp.lang.java.beans
- news:comp.lang.java.databases
- news:comp.lang.java.gui
- news:comp.lang.java.help
- news:comp.lang.java.machine
- news:comp.lang.java.programmer
- news:comp.lang.java.security
- news:comp.lang.java.softwaretools

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

Reporting bugs

If you find what you think may be a bug in the software, please report it to Borland at one of the following sites:

- Support Programs page at `http://www.borland.com/devsupport/namerica/`. Click the Information link under "Reporting Defects" to open the Welcome page of Quality Central, Borland's bug-tracking tool.
- Quality Central at `http://qc.borland.com`. Follow the instructions on the Quality Central page in the "Bugs Report" section.
- Quality Central menu command on the main Tools menu of JBuilder (Tools|Quality Central). Follow the instructions to create your QC user account and report the bug. See the Borland Quality Central documentation for more information.

When you report a bug, please include all the steps needed to reproduce the bug, including any special environmental settings you used and other programs you were using with JBuilder. Please be specific about the expected behavior versus what actually happened.

If you have comments (compliments, suggestions, or issues) for the JBuilder documentation team, you may email `jpgpubs@borland.com`. This is for documentation issues only. Please note that you must address support issues to developer support.

JBuilder is made by developers for developers. We really value your input.

Chapter 2

Programming for the Java 2 Platform, Enterprise Edition

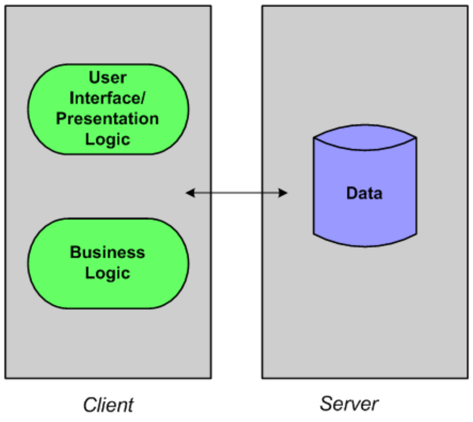
The Java™ 2 Platform, Enterprise Edition (J2EE™) is an architecture for a Java development platform for distributed enterprise applications. It was developed by Sun Microsystems, with input from the development community, including Borland. J2EE platform products, such as the Borland Enterprise Server, offer the developer the capability of building applications with these benefits:

- Reliability and scalability, so business transactions are processed quickly and accurately.
- Excellent security to protect users' privacy and the integrity of the enterprise's data.
- Ready availability, to meet the increasing demands of the global business environment.

JBuilder is a Java integrated development environment that, when coupled with a supported application server from companies such as Borland, BEA, and IBM greatly simplifies and speeds the development of J2EE applications.

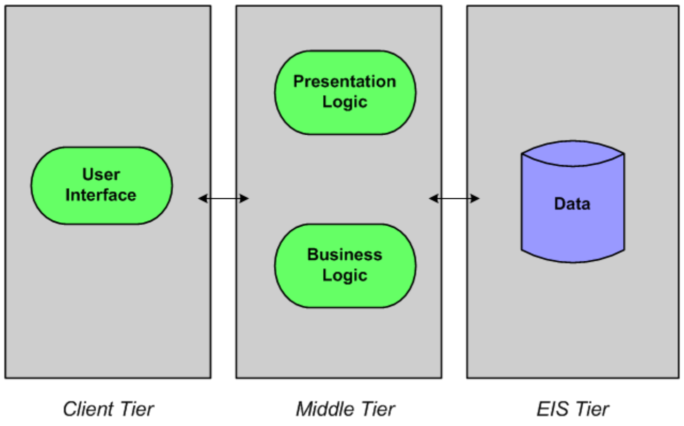
Why are J2EE applications desirable?

In the early 1990s, information systems frequently used a client-server architecture. The user interface to the application usually ran on a desktop computer. This was the client tier. The enterprise data being accessed by the client resided in a database and was “served up” by a server. This approach initially promised improved scalability and functionality.



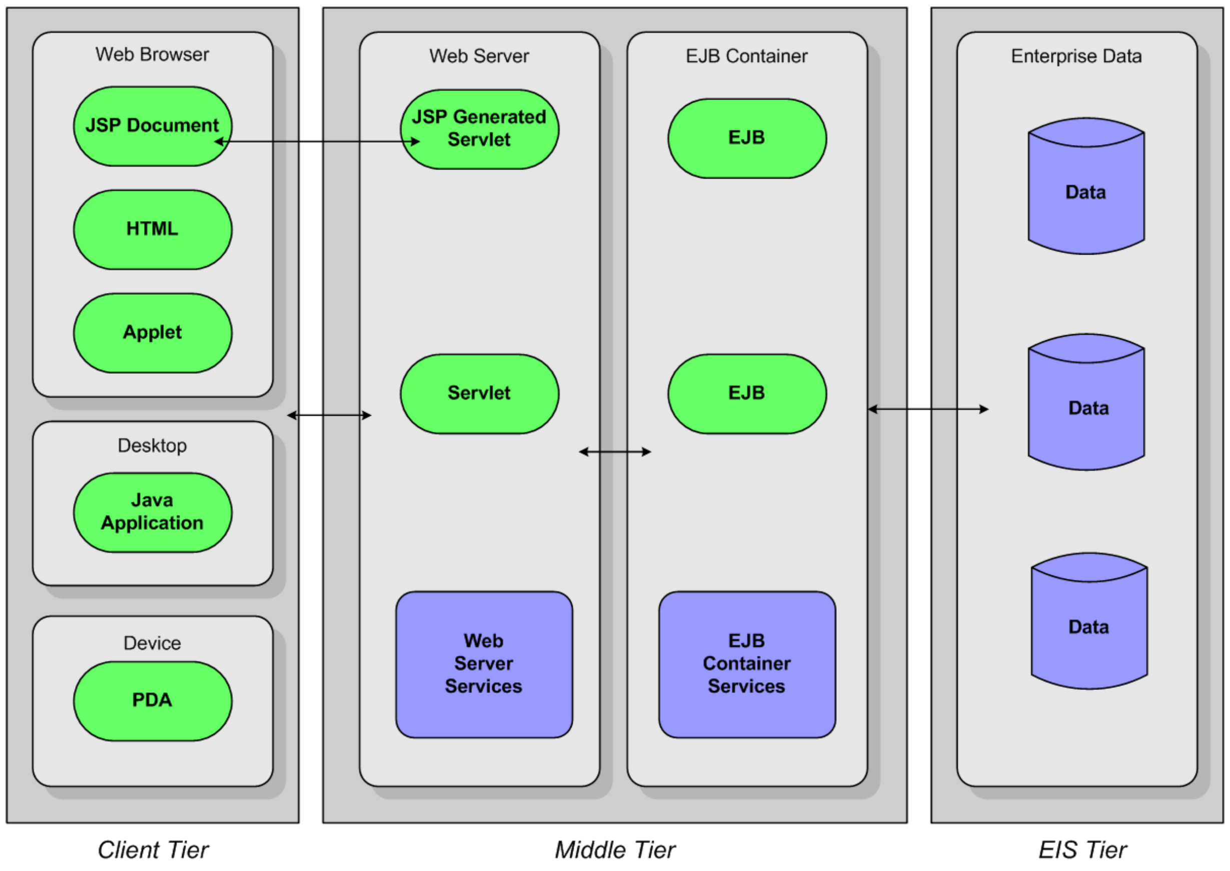
Through hard experience, however, the development community learned that building and maintaining a flexible distributed system is very difficult using the client-server model. For example, the business logic of the application was in the client application. Every time that logic needed modification, the revised application had to be installed on every client machine in the enterprise. Maintenance became a nightmare. These applications also had to manage transactions, be concerned with security, and process the data efficiently, all the while presenting an attractive, easy-to-understand interface to its users. Few developers have talents in all these areas. While a client-server architecture might be adequate for some environments, most of today’s global companies demand considerably more than the client-server model can deliver.

Once the limitations of the client-server approach became apparent, the development community began seeking a better way. The result is the multi-tier model.



In the multi-tier model, the logic involved in presenting the user interface of the application to the user lives on the middle tier. The business logic is now on the middle tier also. When changes are needed, they can be updated in one place instead of on each client machine.

This expanded diagram shows you the various components you might find running on the various tiers:



The client in a J2EE application can be a JavaServer Page (JSP), HTML page, or applet running in a browser; a Java application on a desktop machine; or even a Java client on some portable device, such as a personal digital assistant (PDA) or cell phone.

The middle-tier can have a servlet or a JSP-generated servlet running on a web server. These elements usually make up the server-side presentation logic. An EJB container provides a runtime environment for Enterprise JavaBeans™, which contain the business logic of application. Both a web server and an Enterprise JavaBeans (EJB) container provide services to the components that run on them. Because these services are always available, programmers don't have to include them in the components they write.

The Enterprise Information System (EIS) tier is a repository for the enterprise's data. Usually it consists of the data in a relational database system.

Few J2EE applications have all of these components. They can be mixed and matched in very flexible ways to meet the needs of the enterprise. See Chapter 3, "Creating applications with J2EE technologies" for more on this topic.

Benefits of the multi-tier model

The multi-tier approach adopted by the J2EE platform has several benefits:

- It reduces the complexity of distributed development with a simplified architecture and the sharing of the work load among roles.

  The business logic of the application runs in the middle tier inside an Enterprise JavaBean (EJB) container and/or on a web server. These containers and servers can handle many of the difficult tasks for developers. For example, an EJB container can handle transactions, instance pooling, and data persistence without requiring the EJB programmer to write the logic to perform these tasks. A web server can create and pool instances of servlet classes and handle multiple threads and socket connections. Instead of writing the code to do these things, a member of the development team specifies the desired behavior at deployment time.

  Members of the development team play different roles. Each is a specialist in one or more areas. For example, the content of an HTML page or stylesheet would likely be created by a graphic designer or webmaster. A senior developer might be responsible for the business logic of the application encapsulated within Enterprise JavaBean components. A web developer might develop the user interface and presentation logic using JavaServer Pages (JSPs) and servlets. An application assembler takes the various components of the application and puts it all together, often creating an Enterprise Archive (EAR) file and creating the deployment descriptor that explains how the application is to be deployed. The application deployer and administrator deploy the application. By partitioning the work this way, each step of the development/deployment process is handled by someone skilled in their area while no one has to be an expert in every area.

- It is highly scalable, allowing the development of systems to meet many different needs that can change quickly.

  When demands on the system increase, the logic can be updated easily in one place on the middle tier without having to load new logic on every client machine.

- New applications can integrate well with existing information systems.

  JDBC, a J2EE technology, is a Java API to SQL databases, permitting the access to any type of tabular data that might exist in the enterprise. The Java Naming and Directory Interface (JNDI) allows applications that use Java technology to access enterprise naming and directory services. The J2EE Connector architecture gives Java applications connections to heterogeneous legacy systems. The Java Message Service (JMS) is the Java API for sending and receiving messages though enterprise messaging systems. CORBA services are called using JavaIDL.

- Security is enhanced.

  J2EE technologies are designed with security in mind. For example, only users in assigned roles can access certain methods in enterprise beans. Who can access these methods isn't coded in the enterprise beans themselves. Instead this information is set in the enterprise beans' deployment descriptors, which are used by the deployer to establish their behavior after they are deployed. This type of security is called declarative security.

  For complex security requirements, however, J2EE also allows programmable security. In these cases, the security logic is placed within the code itself.

- Developers can choose from a variety of development tools, servers, and components to develop the applications they need.

  The development team can select the solutions that are best for their needs, without becoming locked into the offerings of a single vendor.

How JBuilder can help

JBuilder Enterprise Edition has many features to help your team develop J2EE applications. These are the technologies JBuilder has to help you develop the client tier:

Client tier technologies

- Applets

  Applets are a special kind of Java application that are downloaded and run by a web browser on a client machine. To begin developing an applet in JBuilder, start with the Applet wizard. For information about working with applets, see “Working with applets” in *Developing Web Applications*.

- Java user interface applications

  JBuilder has several features that can help you develop an application that runs on a client machine. Begin a Java application using the Application wizard. Continue designing your user interface by using JBuilder’s UI designer. You create your UI by adding UI components from JBuilder’s component palette. For information on working with JBuilder’s UI designer, see “Visual design in JBuilder” in *Designing Applications with JBuilder*.

  If you want to create your own JavaBean components to use in your user interface, BeansExpress can simplify the task for you. For information about using BeansExpress, see “Creating JavaBeans with BeansExpress” in *Designing Applications with JBuilder*. JBuilder’s DataExpress components for enterprise beans make it easier for you to build client applications using database-aware visual components such as dbSwing or InternetBeans Express. For more information about DataExpress for EJB, see “Using the DataExpress for Enterprise JavaBeans components” in *Developing Applications with Enterprise JavaBeans*.

  Both a web server and/or an EJB container can run on the middle tier. JBuilder ships with Tomcat, a servlet container that can be used as a web server, and with the Borland Enterprise Server 5.2.1, which contains the EJB container. You can build applications for these servers or you can set up JBuilder to enable you to develop applications for BEA WebLogic 7.x, and 8.1, IBM WebSphere 4.0 and 5.0, Sun-Netscape iPlanet 6.0 and 6.5, and Sybase 4.1 and 4.2.

Middle-tier technologies

These are the middle-tier J2EE technologies that use a web server:

- Servlets

  A servlet is a server-side Java application that can process requests from clients. The servlet responds to the request by generating dynamic output that is sent back to the client. You can begin developing servlets with JBuilder’s Standard Servlet wizard. To find out more about servlets and developing them, see “Developing servlets” in *Developing Web Applications*.

- JavaServer Pages (JSPs)

  An extension of servlet technology, JSPs offer a simplified way to develop servlets. Like servlets, they generate dynamic output that is sent back to the client’s web browser, thus bridging the client and middle tier. Begin developing JSPs with JBuilder’s JavaServer Page wizard. To find out more about JSPs and developing them, see “JavaServer Pages (JSP)” in *Developing Web Applications*.

  InternetBeans Express is a component library that supplements the servlet and JSP technology available in JBuilder. This library makes it easy to present and manipulate data from a database so you can build data-aware servlets and JSPs. For information about InternetBeans Express, see “Using InternetBeans Express” in the *Developing Web Applications.*

This is the middle-tier J2EE technology that uses an EJB container:

- Enterprise JavaBeans (EJBs)

  Enterprise JavaBeans are server-side components that contain the business logic of the application. JBuilder assists you in building EJB 1.x and EJB 2.0 components. Start building enterprise beans by using the EJB wizards on the Enterprise page of the object gallery (File|New|Enterprise). For building EJB 2.0 components, JBuilder offers the EJB designer, a Two-Way Tool™ that allows you to design your beans visually all the while keeping your code, deployment descriptors, and design synchronized. For more information about building, testing, and deploying enterprise beans, see “An introduction to EJB development” in *Developing Applications with Enterprise JavaBeans*.

  As you create your enterprise beans, JBuilder is building your EJB deployment descriptors. You can use JBuilder’s EJB DD Editor to modify them as you wish. For more information about the EJB DD Editor see Chapter 11, “Editing J2EE deployment descriptors” and “Editing EJB deployment descriptors” in *Developing Applications with Enterprise JavaBeans*.

All the web application components and enterprise bean components can be combined and delivered in an application module. JBuilder has an Application Module wizard. For more information, see “Creating an application module” in the “Deploying enterprise beans” chapter of the *Developing Applications with Enterprise JavaBeans* book.

Other J2EE technologies

While not confined to a particular architectural tier, these technologies are enablers that make things work:

- Java Database Connectivity (JDBC)

  JDBC is the standard used to access your database on the Enterprise Information Systems (EIS) tier. It defines a Java API you use to write SQL statements that are sent to your database.

  JBuilder includes DataExpress, a component library for accessing data in your database. It connects your application to your database using JDBC drivers.

  JBuilder also includes JDataStore, an all-Java embedded database and component library. You access JDataStore using JDBC.

  Entity beans that access rows in your database, also connect to your data using JDBC.

- Java Message Service (JMS)

  JMS is an enterprise messaging service that routes messages between components and processes in a distributed application.

  The Borland Enterprise Server includes SonicMQ, a JMS implementation. Also JBuilder supports EJB 2.0 components, which include message-driven beans. Message-driven beans integrate JMS into enterprise beans. JBuilder also includes a JMS wizard. See “Creating JMS producers and consumers” in *Developing Applications with Enterprise JavaBeans* for information on creating classes and applications that can create and consume JMS messages.

- Java Naming and Directory Interface (JNDI)

  All J2EE servers use JNDI, a Java naming service used to locate distributed objects.

- Extensible Markup Language (XML)

  Although not really a J2EE technology, XML is widely used by J2EE technologies. For example, web components and enterprise beans have their deployment descriptors written in XML. These deployment descriptors describe how the components behave once they are deployed.

  JBuilder has several XML features that help you accomplish common programming tasks you might encounter in your J2EE projects. For information about JBuilder's XML features, see "Introduction" in *Working with XML*.

Preparing to deploy J2EE applications

As you create and compile your web applications and enterprise beans, JBuilder can create the WAR (Web Archive) and EJB-JAR (EJB Archive) files for you automatically. You can choose to bundle the components of a J2EE application together into an EAR file. JBuilder provides an Application Module wizard to help you do this. For more information, see "Creating an application module" in *Developing Applications with Enterprise JavaBeans*.

Learning about J2EE

If you've read this far, you've been exposed very briefly to many concepts and, with all the acronyms to identify J2EE technologies, an alphabet soup. To develop a deeper understanding of J2EE benefits and concepts, begin your explorations on Sun's `www.java.sun.com` web site. This `http://java.sun.com/j2ee/docs.html` link takes you to Sun's J2EE documentation home page where you can find an abundance of useful information.

If you're new to J2EE programming, look at the J2EE tutorial at `http://java.sun.com/j2ee/tutorial/index.html`. For an in-depth discussion of J2EE programming and the recommended programming practices to use in your J2EE applications, don't miss the very detailed J2EE Blueprints, found at `http://java.sun.com/j2ee/blueprints/index.html`. J2EE Blueprints is an integral part of J2EE itself. You'll find it useful when you need to understand deeper concepts and are looking for the best ways to approach J2EE development. This material is also available in book form as *Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition* written by Nicholas Kassem and the Enterprise Team of Sun. `http://java.sun.com/j2ee/blueprints/aboutthebook.html` links you to information about the book.

You'll find additional documentation and specifications for the various J2EE technologies on the Sun site. There are also excellent third-party books, but because J2EE is a developing product, be aware which versions of the various technologies they address.

Chapter

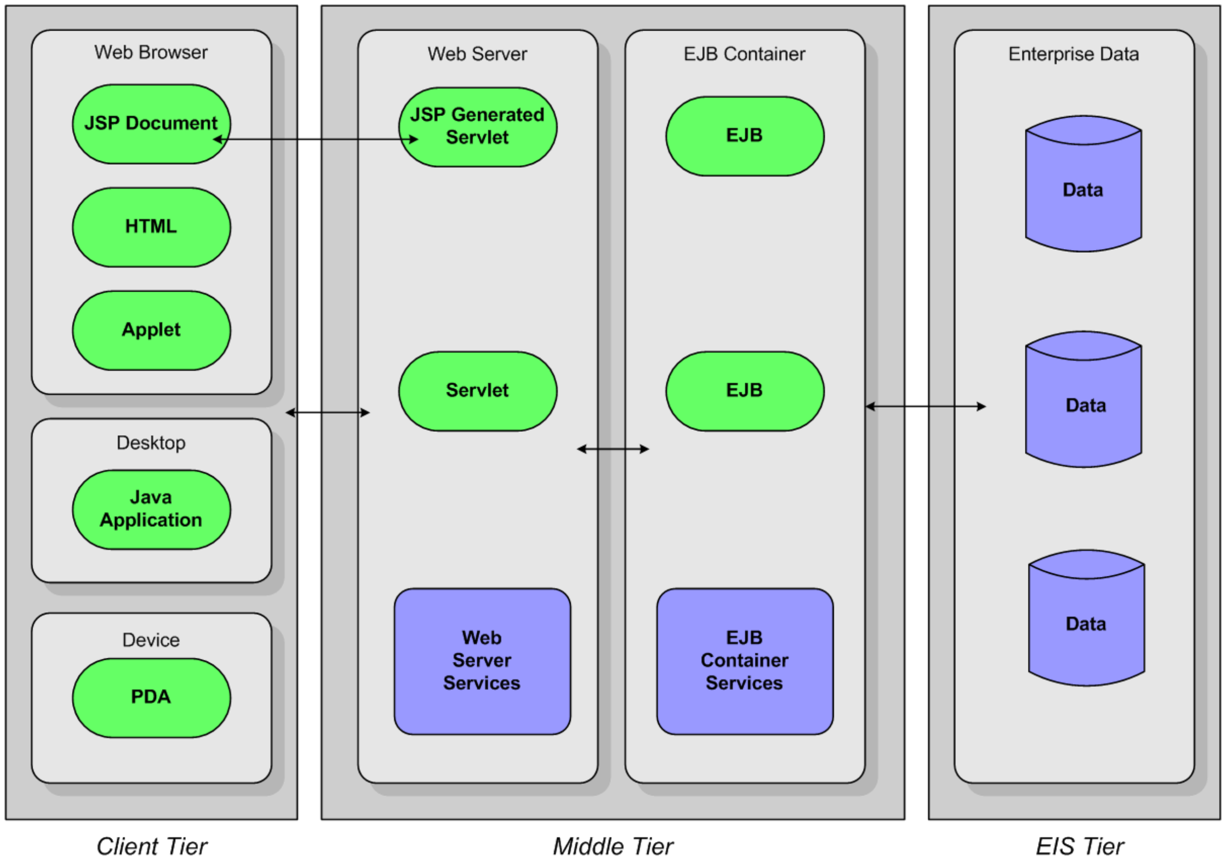
3

Creating applications with J2EE technologies

The *Developing Web Applications*, *Developing Applications with Enterprise JavaBeans*, and other parts of JBuilder's documentation explain how to use J2EE technologies. They describe in depth how to use JBuilder features to develop web applications, work with XML, develop Enterprise JavaBeans, access and work with your data using DataExpress, and create web services. Within each of these areas of the documentation, you should find the information to understand these technologies and work productively with them.

But today's web-based enterprise applications usually use several J2EE technologies. How do these disparate technologies fit together in applications you and your team might develop? Which technologies are right for which kind of applications? This chapter intends to help you see how the pieces fit together, while highlighting the flexibility of J2EE applications. It borrows heavily from the Introduction chapter of *Designing Enterprise Applications with the J2EE Platform* by Inderjeet Singh, Beth Stearns, Mike Johnson, and the Sun Enterprise Team. You can find the book online on the Sun web site at `http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/#chapters`, and it is also available for purchase.

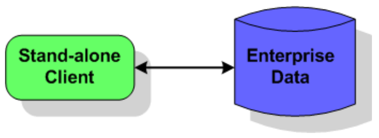
Here's the diagram from the previous chapter showing the multiple tiers used in most J2EE applications:



The diagram shows you all the different types of entities found on each tier, but it doesn't show you the common ways J2EE applications combine these technologies to create reliable, scalable, and easily distributed applications.

Client-server applications

Before looking at multi-tier J2EE models, consider the old standard, the two-tier client-server model. Using J2EE technologies does not preclude you from creating client-server applications. Client-server applications omit the middle tier shown in the multi-tier application diagram. While the client-server model scales poorly and maintaining and distributing such applications is more difficult, sometimes a Java client on a desktop that accesses the data of the company directly is all you really need.

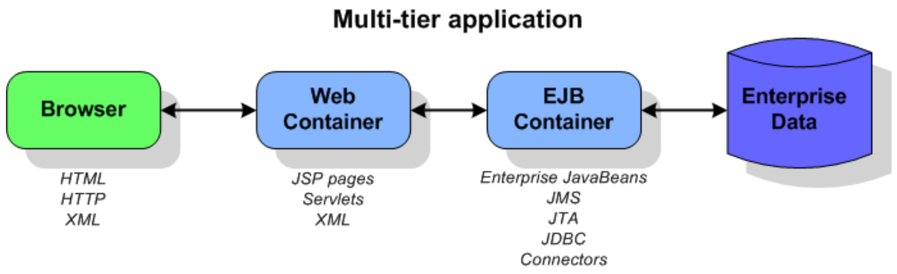


The Java client can contain both the presentation and business logic, although it's possible one application could handle the presentation tasks while another could handle business logic, all on the client tier. You would use JDBC or Connectors to access the data on the Enterprise Information System (EIS) tier.

Consider using the client-server model only when the number of desktops running clients is quite small and will remain so. Once the demand for more than a few client instances increases, you should probably consider another type of application architecture.

Multi-tier applications

Here you see the classic multi-tier, distributed architecture that uses three tiers: the browser on the client tier, the web container and EJB container on the middle tier that runs on a J2EE server, and the Enterprise Information Systems (EIS) tier that is managed by a database system. Such an application uses web components such as JavaServer Pages (JSPs), servlets, and XML to manage the application's presentation logic. The EJB container contains enterprise beans that respond to requests from the web tier components and access the data in the EIS tier.

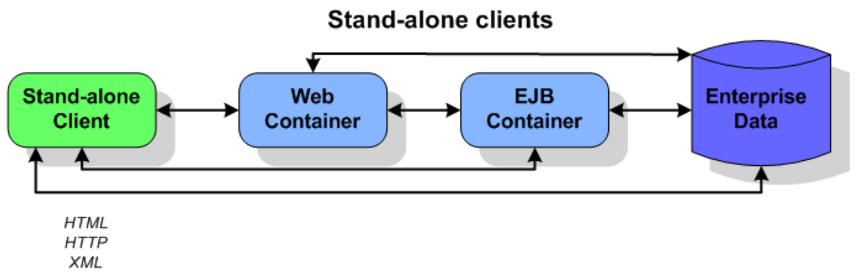


When should you use JavaServer Pages and when should you use servlets? JavaServer Pages are intended as user interface components. Servlets are usually used for processing requests and controlling application logic.

XML can be very important in multi-tier applications. XML data messages use HTTP as their transport protocol. Such data messages can encapsulate all types of data and respond to a variety of client types, including XML-enabled browsers.

Stand-alone clients

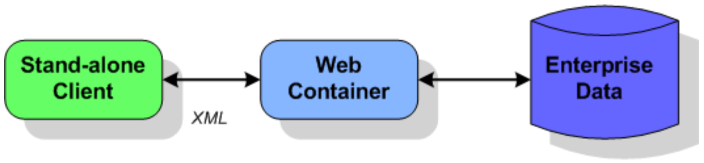
Not all multi-tier applications use a browser on the client tier. Often the client is a Java client or a client written in another language that consumes dynamic web content or an application that interacts directly with enterprise beans running in an EJB container. This diagram shows you the possibilities of this type of application:



Note that this diagram includes the client-server application also as such an application includes a stand-alone client, too.

Consumers of dynamic web content

One type of stand-alone client is a Java application or another programming language that consumes dynamic web content. This web content is usually XML data messages:



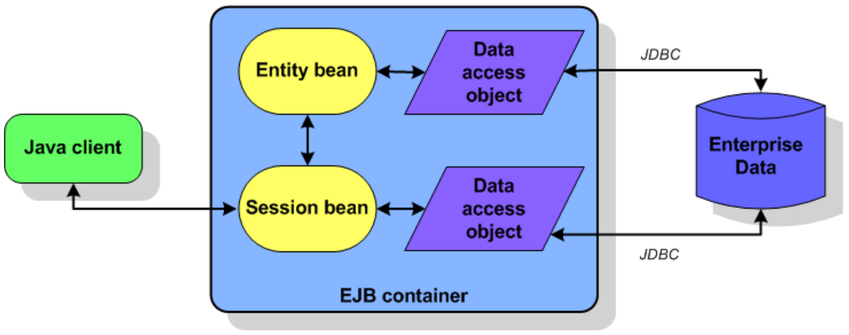
The web container performs the XML transformations and provides web connectivity to clients. The client handles the presentation logic while the web tier manages the business logic and accesses the data on the EIS tier if necessary. Business logic might be implemented as enterprise beans. The J2EE technologies likely to be used are XML, servlets, web services, and possibly enterprise beans and JDBC to access enterprise data.

Java client calling EJBs

Enterprise JavaBeans (EJBs) are server-side components that run in a middle-tier EJB container. The container provides services to the beans, such as transaction management and security. You, as the developer, don't have to implement such low-level and complex services, but instead can concentrate on developing the business logic for the beans, knowing that the services are there when they are needed. Developers often choose EJBs when building applications that will be distributed over several servers. They also like to use EJBs because of the transparent way they handle transactions.

The following diagram depicts a Java client that calls upon the business methods encapsulated with the enterprise beans that are running in an EJB container. If the beans are entity beans, they access company data on the EIS tier through JDBC and J2EE connectors. Ideally, the client should access company data through a session bean which interacts with the entity beans that model company data. A session bean that interacts with an entity bean in this way is usually using the Session Facade design pattern; see "Creating session facades for entity beans" in *Developing Applications with Enterprise JavaBeans* for more information. XML is used in the deployment descriptors for enterprise beans.

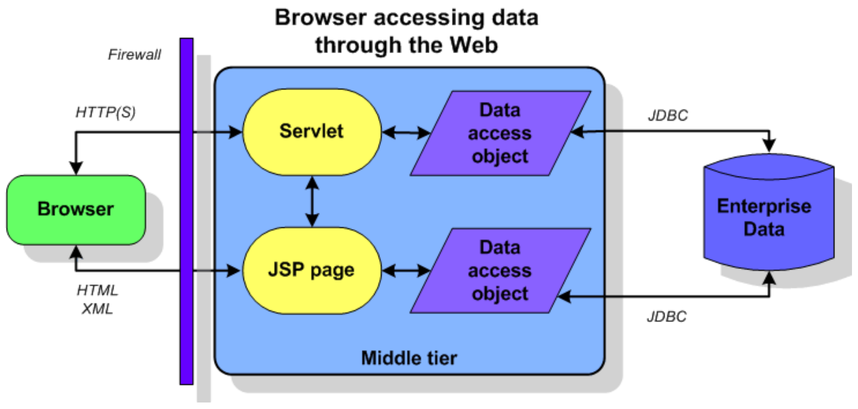
Java client calling EJBs



If the application you need requires secure transactions over a distributed system, consider using enterprise beans.

Web-centric applications

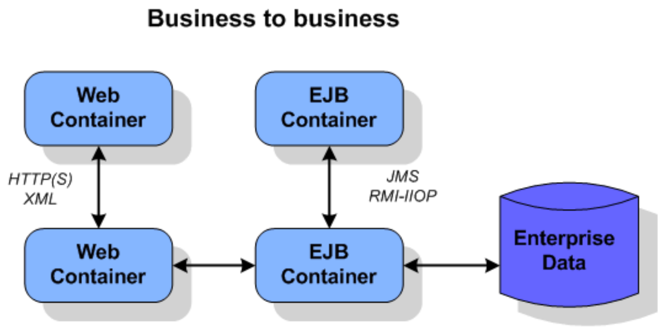
There are times when your application just doesn't need to use enterprise beans and if you do so, you add a layer of complexity. If your application doesn't require transactions on a distributed system, consider a web-centric application model:



This scenario puts both the presentation and business logic on the web tier. The web container can host JavaServer Pages and servlets. The client is simply a browser and data is transferred using XML, HTTP protocol, and/or HTML pages. Servlets can access the EIS tier, when necessary, using JDBC or connectors. Applications using this type of architecture are quite common.

Business to business

When you need a web-based commerce solution, consider the business-to-business model. This might include multiple web containers and multiple EJB containers.



Web container to web container architecture is suitable for ecommerce solutions. Communication takes place through XML data message over HTTP. Such applications are very loosely coupled.

Peer-to-peer communication between EJB containers offers a more tightly coupled model that works well for intranet solutions. In these types of applications, seriously consider using message-driven enterprise beans and Java Message Service (JMS), which enable you to develop loosely coupled applications.

Chapter

4

Configuring the target server settings

Before you begin creating enterprise applications, you must configure the settings for the application server and/or web server to which you are going to deploy.

Tip Be sure to see the server-specific chapters for the various servers to find additional configuration information for each server. These are the chapters:

- Chapter 5, "Using JBuilder with Borland servers"
- Chapter 7, "Using JBuilder with BEA WebLogic servers"
- Chapter 8, "Using JBuilder with IBM WebSphere servers"
- Chapter 9, "Using JBuilder with JBoss servers"
- Chapter 10, "Using JBuilder with Tomcat"

Supported servers

JBuilder supports the following servers:

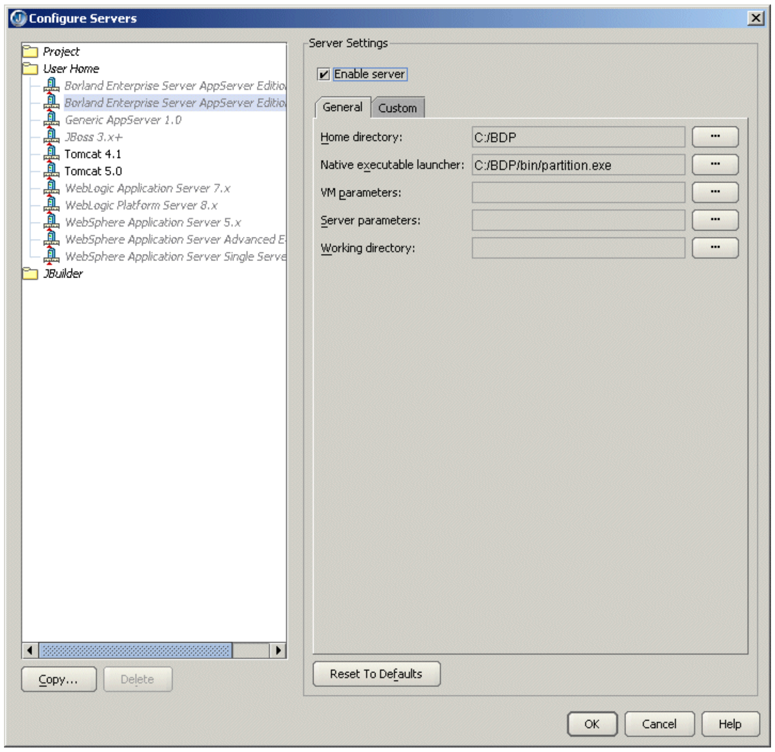
- Borland Enterprise Server 5.2.1
- Borland Enterprise Server 6.0.1
- BEA WebLogic Server 7.0 SP 5
- BEA WebLogic Platform 8.1 SP 3
- IBM WebSphere Application Server 4.0.7 Single Server
- IBM WebSphere Application Server 4.0.7 Advanced Edition
- IBM WebSphere Application Server 5.0.2.4, 5.1.0.4
- JBoss Application Server 3.0.8
- JBoss Application Server 3.2.5
- Tomcat 4.1.30
- Tomcat 5.0.27

Setting up servers within JBuilder

This is a feature of JBuilder Developer and Enterprise

To configure the settings to target one or more application servers,

1 Choose Enterprise|Configure Servers. The Configure Servers dialog box is displayed:



Configure Servers

- Project
- User Home
  - Borland Enterprise Server AppServer Editio
  - Borland Enterprise Server AppServer Editio
  - Generic AppServer 1.0
  - JBoss 3.x+
  - Tomcat 4.1
  - Tomcat 5.0
  - WebLogic Application Server 7.x
  - WebLogic Platform Server 8.x
  - WebSphere Application Server 5.x
  - WebSphere Application Server Advanced E
  - WebSphere Application Server Single Serve
- JBuilder

Server Settings

☑ Enable server

Home directory:	C:/BDP
Native executable launcher:	C:/BDP/bin/partition.exe
VM parameters:	
Server parameters:	
Working directory:	

The left side of the dialog box lists the servers that can be configured in JBuilder and for which JBuilder finds a registered OpenTool.

2 Select the server you want to configure by clicking it in the pane on the left.

The right side of the dialog box lists the default settings for the selected server. Each application server except Generic App Server 1.0 and all versions of Tomcat has both a General and a Custom page for editing settings. The General page has fields that all the application servers have in common, while the Custom page has fields that are specific to the selected application server. In some cases, modifying a Custom setting will update a setting on the General page and vice versa.

The Generic AppServer 1.0 server option is a generic option. It represents a basic application server that supports EJB 1.1 and/or EJB 2.0 development. Select it if the application server you use is not currently supported by JBuilder. You will probably want to edit the resulting deployment descriptor with tools supplied with that application server to get the exact settings you want. You could also choose this option if the you aren't targeting a specific application server.

3 Check the Enable Server check box at the top of the dialog box.

Checking this option enables the fields for the selected application server. You won't be able to edit any fields until it is checked. The Enable Server check box also determines whether this server will appear in the list of servers when you select a server for your project using the Server page of the Project|Project Properties dialog box.

JBuilder provides default settings for the selected server. If JBuilder's default settings are not appropriate, edit any of the settings as needed. Some servers require you to enter settings in addition to the defaults. Clicking the OK button in this dialog box when not all required values are set or selecting another server while the current one is enabled displays a message dialog box that informs you about the missing settings and usually selects the control representing the missing setting.

4 If you want to change any of the settings, click the ellipsis (…) button next to the field and make your changes.

   You can add any necessary application dependencies to the server's classpath by clicking the Required Libraries tab and using the Add button to add the libraries the server needs.

5 Click the Custom tab to view settings unique to the server. Most application servers allow you to specify the location of the JDK the server uses; some require you to specify the JDK location for successful configuration completion. Updates you make on the Custom page can update settings on the General page. (Note that Tomcat configurations do not have a Custom tab.)

   Following are the JDKs and their installed locations for the application servers supported by JBuilder:

   - Borland Enterprise Server AppServer Edition 5.2.1: `<bes home>/jdk/jdk1.3.1` or `<bes home>/jdk/jdk1.4.1` (the default value)
   - Borland Enterprise Server AppServer Edition 6.0: `<bes home>/jdk/jdk1.4.2`
   - WebLogic Server 7.x: `<bea home>/jdk131_10`
   - WebLogic Platform Server 8.1: `<bea home>/jdk141_04`. WebLogic 8.1 also supports JRockit (a JDK optimized for server applications), so the JDK directory could be <bea home>/jrockit81sp3_142_04
   - WebSphere 4.x (either Advanced Edition or Single Server): `<websphere home>/java`
   - WebSphere 5.x: `<websphere home>/java`

   As you make your edits, JBuilder attempts to validate your settings. Once you've made your edits, click the OK button to accept your changes and close the dialog box. If you prefer that JBuilder not try to validate your changes, uncheck the Enable Server dialog box. Selecting OK saves changes for all servers you've enabled, disabled, or modified.

6 Click OK when you are through configuring the application server.

   When you click OK and the Enable Server check box is checked, JBuilder attempts to verify your settings. If errors are detected, a message box containing information about the errors appears and focus is given to the control appropriate for fixing the error. If the Enable Server option is unchecked, the dialog box simply closes and no setting validation is performed.

Once a server is properly configured and validated, the server you configured is listed in black type in the pane on the left. Gray type indicates the server hasn't been configured yet. Red type indicates an error condition; specifically it means the server's library containing its settings has been read, but the OpenTool representing the server can't be found. You can delete a server shown in red by selecting it and clicking the Delete button.

The Reset To Defaults button sets the settings back to the values they originally had when JBuilder was first installed and the server is disabled.

If a dialog box appears with a message that you must restart JBuilder, you must close and then restart JBuilder for changes to become effective; otherwise, restarting JBuilder is not necessary.

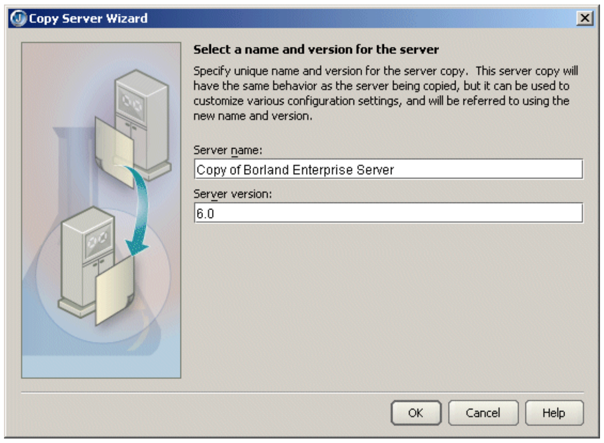
Creating a duplicate configuration to edit

Once you've created a server configuration, you might find you'd like to have a similar one, but with some slight differences. Follow these steps to create a duplicate configuration you can then edit:

1 Choose Enterprise|Configure Servers to display the Configure Servers dialog box.

2 Select the server configuration you would like to duplicate from the left pane.

3 Click the Copy button at the bottom of the left pane to display the Copy Server wizard:



Select a name and version for the server

Specify unique name and version for the server copy. This server copy will have the same behavior as the server being copied, but it can be used to customize various configuration settings, and will be referred to using the new name and version.

Server name: Copy of Borland Enterprise Server

Server version: 6.0

4 Specify the name you want to use to identify this server configuration in the Name For This Server field.

5 Specify the server version in the Version For This Server field.

6 Click OK.

7 Select your new configuration in the left pane of the Configure Servers dialog box and make the changes to the settings you need using the General and Custom pages.

Copying a server configuration also creates copies of that server's tools. You could, therefore, create a duplicate server configuration and modify just the server's tools as the only difference between the two configurations. As another example, you could have the copied configuration refer to a different home directory if you have slightly different versions of that server installed. You can also delete a server configuration in the Configure Servers dialog box.

Adding a service pack

If you want to add a service pack to your application server configuration, follow these steps:

1 Choose Enterprise|Configure Servers.

2 Select your server from the list of servers in the pane on the left side of the dialog box.

3 Click the General tab, then the Class tab, if it isn't already selected.

4 Click Add.

5 Navigate to the location of the service pack JAR file.

6 Click OK twice to close all dialog boxes.

The created libraries

When you configure the server settings, one or more libraries are created for you automatically in your JBuilder home directory that contain all the application server files you will need for enterprise bean development with the application server of your choice. These are the libraries created for you, listed by application server:

Borland Enterprise Server 6.0.1

- Borland Enterprise Server 6.0 Client: All JARs needed to run a client.
- Borland Enterprise Server 6.0 Servlet: Used for web applications.

Borland Enterprise Server 5.2.1

- Borland Enterprise Server 5.2.1 Client: All JARs needed to run a client.
- Borland Enterprise Server 5.2.1 Servlet: Used for web applications.

WebLogic 8.1

- WebLogic 8.1 Client: All JARs needed to run a client.
- WebLogic 8.1 Deploy: JARs needed to run the WebLogic 8.1 deploy tool.
- WebLogic 8.1 Servlet: Used for web applications.

WebLogic 7.x

- WebLogic 7.x Client: All JARs needed to run a client.
- WebLogic 7.x Deploy: JARs needed to run the WebLogic 7.x deploy tool.
- WebLogic 7.x Servlet: Used for web applications.

WebSphere 5.x

- WebSphere 5.x Client: JARs needed to run a client.
- WebSphere Application Server 5.x Ext Dirs: JARs used during server startup.
- WebSphere Application Server 5.x EJB Deploy: JARs used to compile enterprise beans and create stubs.
- WebSphere Application Server 5.x Servlet: Used for web applications.

WebSphere 4.0 Single Server

- WebSphere AES 4.0 Client: JARs needed to run a client.
- WebSphere AES 4.0 Ext Dirs: JARs used during server startup.
- WebSphere AES 4.0 EjbDeploy: JARs used to compile enterprise beans and create stubs.
- WebSphere AES 4.0 SeAppInstaller: JARs used to run the WebSphere 4.0 deploy tool (SeAppInstaller).
- WebSphere AES 4.0 Servlet: Used for web applications.

WebSphere 4.0 Advanced Edition

- WebSphere AE 4.0 Client: JARs needed to run a client.
- WebSphere AE 4.0 Ext Dirs: JARs used during server startup.
- WebSphere AE 4.0 XmlConfig: JARs used to run the WebSphere 4.0 AE deploy tool.
- WebSphere AE 4.0 EjbDeploy: JARs used to compile enterprise beans can create stubs.
- WebSphere AE 4.0 Servlet: Used for web applications.

JBoss 3.x+

- JBoss 3.x+ Client: JARs needed to run a client.
- JBoss 3.x+ Servlet: Used for web applications.

Tomcat

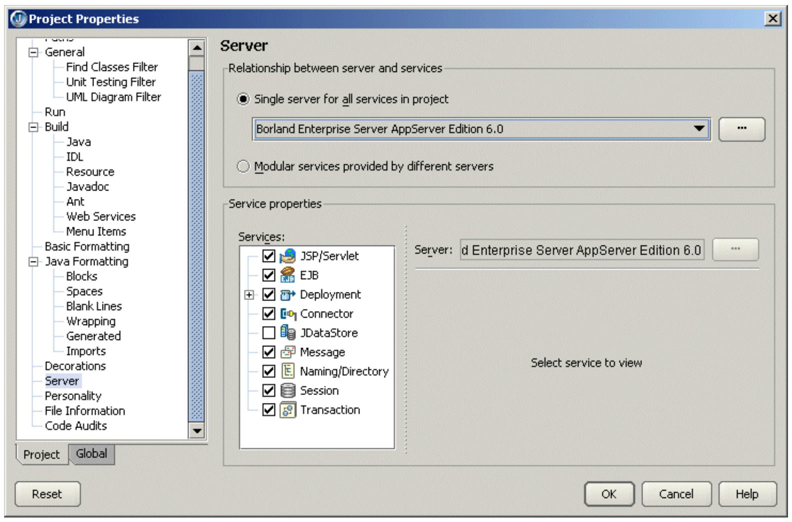
- Tomcat 4.1 Servlet: Used for web applications.
- Tomcat 5.0 Servlet: Used for web applications.

Selecting a server

JBuilder can target multiple servers. You can choose a single application server for all stages of EJB and web application development, or you can choose different servers for different aspects of development. For example, you can select one server to use to develop enterprise beans and another to develop web applications.

To select one or more servers to use for your project,

1. Choose Project|Project Properties.
2. Select Server in the dialog box tree. The Server page is displayed:



Project Properties

- General
  - Find Classes Filter
  - Unit Testing Filter
  - UML Diagram Filter
- Run
- Build
  - Java
  - IDL
  - Resource
  - Javadoc
  - Ant
  - Web Services
  - Menu Items
- Basic Formatting
- Java Formatting
  - Blocks
  - Spaces
  - Blank Lines
  - Wrapping
  - Generated
  - Imports
- Decorations
- Server
- Personality
- File Information
- Code Audits

Server

Relationship between server and services

[✓ Single server for all services in project]

Borland Enterprise Server AppServer Edition 6.0

Modular services provided by different servers

Service properties

Services:

- ✓ JSP/Servlet
- ✓ EJB
- ✓ Deployment
- ✓ Connector
- ☐ JDataStore
- ✓ Message
- ✓ Naming/Directory
- ✓ Session
- ✓ Transaction

Server: d Enterprise Server AppServer Edition 6.0

Select service to view

3. Decide whether you want to use a single server for all aspects of development or different servers to handle different areas of development.

   - To use a single server,

     1. Select the Single Server For All Services In Project option and select the server from the drop-down list.
     2. If you want to make changes to the configuration settings for the server, click the ellipsis (…) button and edit the settings you want on the General and Custom pages. Click OK.

        You can also use this dialog box to select a different server.

     3. If you don't want a particular service started when the server starts up, uncheck the check box next to the service in the Services list. This feature currently applies only to the Borland Enterprise Server as it is the only one which allows you to start and stop selected services.

        If you uncheck the Deployment or Client JAR Creation services, the corresponding menu items will be disabled on the JBuilder Enterprise menu. If you uncheck the JSP/Servlet service, web server support is disabled. If you uncheck the Naming/Directory service for Tomcat, naming service support is disabled. If you uncheck the Deployment service, vendor-specific descriptors won't be generated and all deploy options will be disabled.

- To use different servers for different services,
  1. Select the Modular Services Provided By Different Servers option.
  2. Check the check boxes next to all the services for which you want to specify an application server in the Services list.
  3. Click one of the checked services to select it in the Services list.
  4. In the Service Properties For Project group box, use the Server drop-down list to select the server you want to use for this service.

     If server-specific properties are available for the server you selected for the particular service you are working with, they will appear below the Server drop-down list. If the Deployment service is selected, a Build Target For Deploying To Server drop-down list appears on the Service Properties For Project panel. Select the type of build you want to occur. If the EJB service is selected, you'll see read-only information appear that is intended to help you decide which server is appropriate for your needs.

     If you select the JSP/Servlet service, the Default Runtime Host Name and Default Runtime Port Number fields are available to you. The default values refer to the global server configuration. If you change these values, they are used for all server run configurations created after you made the change.

     If you expand the Deployment service node, you can select from several subnodes that have accompanying properties. The properties vary depending on which is your selected server. All servers have a Build Target For Deploying To Server field, however. Select your desired build option from the drop-down list.

     For Borland Enterprise Server only

     The Deployment jndi-definitions.xml option determines whether the `jndi-definitions.xml` file is included in any of the deployment actions you select. Uncheck it if you don't want it included. `jndi-definitions.xml` is an XML deployment descriptor for EJB 2.0 resources.

     For WebLogic 7.x – 8.1

     If you expand the WebLogic Deployment service node and select the WARs node, the service properties include a Map Project Webapps At Runtime option. When this option is selected (the default value) all web applications in the project deploy from the web application directory and not as a WAR when the server starts up.

     If you expand the WebLogic Deployment service node and select the EARs node, the service properties include a Map Project EJB Modules At Runtime option. When this option is selected (the default value) all EJBs in the project deploy from the EJB directory and not as a EAR when the server starts up.

     All the WebLogic Deployment service nodes have Deploy As Archive and Deploy As Exploded Directory options. Checking the Deploy As Archive option enables the archive to be deployed. Checking the Deploy As Exploded Directory option means the archive is exploded into a directory as its contents are deployed instead.

     If you select the EJB service, the service properties include a combo box with options to deploy data sources for all EJB modules in the project. These are the options:

     - Map Project Data Sources: Maps data sources for all EJB modules in `config.xml` for the domain specified in the server configuration. DataSource entries are removed when the server shuts down.
     - Map And Persist Project Data Sources: Maps data sources for all EJB modules in `config.xml` for the domain specified in the server configuration.

Data Source information is persisted in `config.xml`. DataSource entries are removed when the server shuts down.

- Do Not Map Project Data Sources: Does not map any data sources.

Note that JBuilder also adds the JDBC driver (if it can be located in the project's required libraries) to the server's classpath at start up time. All data sources created by JBuilder are transactional data sources.

5. If you want to make changes to the configuration settings for the server selected for the service, click the ellipsis (…) button and edit the settings you want on the General and Custom pages. Click OK.
6. Repeat these steps for each service you want access to.

4 Click OK.

If you neglect to select one ore more servers for a project, the Server page of the Project Properties dialog box appears when you start an EJB wizard from the object gallery. All EJB projects must have one or more selected servers.

Setting up JDBC drivers

To enable JBuilder's EJB wizards and EJB designer to access a database, you must install the JDBC driver supplied by the database vendor and set up the driver in JBuilder. Install a JDBC driver following the vendor's instructions.

To begin setting up the driver in JBuilder, select Enterprise|Enterprise Setup to display the Enterprise Setup dialog box. Select Database Drivers in the tree to display the Database Drivers page. Use this page to add a new database driver to JBuilder. See the following section for step-by-step instructions.

Creating the .library and .config files

There are three steps to adding a database driver to JBuilder:

1. Creating a library file which contains the driver's classes, typically a JAR file, and any other auxiliary files such as documentation and source.
2. Deriving a .config file from the library file which JBuilder adds to its classpath at start-up.
3. Adding the new library to your project, or to the Default project if you want it available for all new projects.

The first two steps can be accomplished from the Database Drivers page:

1. Open JBuilder and choose Enterprise|Enterprise Setup. Click the Database Drivers tab which displays .config files for all the currently known database drivers.
2. Click Add to add a new driver, then New to create a new library file for the driver. The library file is used to add the driver to the required libraries list for projects.

Note You can also create a new library under Tools|Configure|Libraries, but since you would then have to use Enterprise Setup to derive the .config file, it is simpler to do it all here.

3. Type a name and select a location for the new file in the Create New Library dialog box.
4. Click Add, and browse to the location of the driver. You can select the directory containing the driver and all it's support files, or you can select just the archive file for the driver. Either will work. JBuilder will extract the information it needs.

5 Click OK to close the file browser. This displays the new library at the bottom of the library list and selects it.

6 Click OK. JBuilder creates a new .library file in the JBuilder `/lib` directory with the name you specified (for example, `InterClient.library`). It also returns you to the Database Drivers page which displays the name of the corresponding .config file in the list which will be derived from the library file (for example, `InterClient.config`).

7 Select the new .config file in the database driver list and click OK. This places the .config file in the JBuilder `/lib/ext` directory.

8 Close and restart JBuilder so the changes to the database drivers will take effect, and the new driver will be put on the JBuilder classpath.

Important If you make changes to the .library file after the .config file has been derived, you must re-generate the .config file using Enterprise Setup, then restart JBuilder.

Now that JBuilder can see the database driver, you must add the database driver library to the Required Libraries list in Project|Properties, or Project|Default Properties.

Adding the JDBC driver to projects

Projects run from within JBuilder use only the classpath defined for that project. Therefore, to make sure the JDBC driver is available for all new projects that will need it, define the library and add it to your default list of required libraries. This is done from within JBuilder using the following steps:

1 Start JBuilder and close any open projects.

2 Choose Project|Default Project Properties.

3 Select the Required Libraries tab on the Paths page, then click the Add button.

4 Select the new JDBC driver from the library list and click OK.

5 Click OK to close the Default Project Properties dialog box.

Note You can also add the JDBC driver to an existing project. Just open the project, then choose Project|Properties and use the same process as above.

Updating projects with the latest server settings

If a server configuration has been modified since a project was last opened, JBuilder updates it with the latest server settings for the project's selected server(s). So, for example, if you have modified server settings for one project and you have others that aren't open that use the same server configuration, the next time you open those other projects, your modified server settings will be in force automatically. If you want to use a similar but unique server configuration for a particular project, create a duplicate configuration using the Copy button, and use the Copy Servers wizard to edit it to your needs.

Be aware that this automatic updating of projects with the latest server settings can occur only with server configurations modified in JBuilder 9 or higher. If you attempt to transfer server libraries from a previous JBuilder version, your projects won't be updated with them. You can still tell JBuilder to update the project manually:

1 Right-click the project node of the project you want to update in the project pane and choose Properties.

2 Select the Servers page.

3 Click the ellipsis (…) button next to your selected single server or the separate service servers and make your changes.

4 Click OK to close the dialog box.

Updating from an earlier version of JBuilder

Previous versions of JBuilder supported an older module format. Your old modules will be automatically converted to the new module format in JBuilder X. Once the conversion has taken place, you will not be able to share modules between JBuilder X and above and an earlier version of JBuilder.

Chapter

5

Using JBuilder with Borland servers

This chapter explains how to set up and use Borland servers with JBuilder. JBuilder supports Borland Enterprise Server AppServer Editions 5.2.1 and 6.0.1.

Configuring Borland servers in JBuilder

To configure JBuilder settings to target your preferred Borland server, you use the Configure Servers dialog box (Enterprise|Configure Servers). See the instructions below for either the Borland Enterprise Server AppServer Edition 6.0 or the Borland Enterprise Server AppServer Edition 5.2.1.

Borland Enterprise Server AppServer Edition 6.0

To configure JBuilder settings to target Borland Enterprise Server AppServer Edition 6.0,

1 Choose Enterprise|Configure Servers to display the Configure Servers dialog box.

2 Select Borland Enterprise Server AppServer Edition 6.0 from the User Home folder in the left pane.

   The right side of the dialog box displays the default settings for the server. The General page has fields that all servers have in common, while the Custom page has fields that are specific to the selected server. In some cases, modifying a Custom setting will update a setting on the General page.

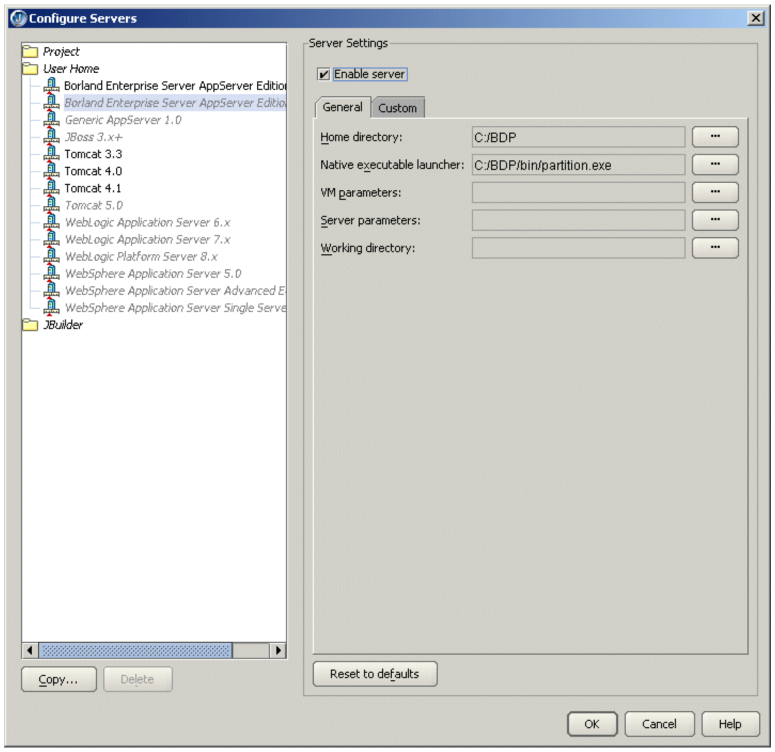
3 Select the Enable Server option at the top of the dialog box.

   Checking this option enables the fields for the selected application server. You won't be able to edit any fields until it is checked. The Enable Server check box also determines whether this server will appear in the list of servers when you select a server for your project using Project|Project Properties|Servers.

   - Check the General tab to view and change (if required) default directories and parameter settings for the server. If JBuilder's default settings are not appropriate, edit any of the settings as needed. To change settings, click the ellipsis (…) button next to the field and make your changes.

- Home Directory: The directory where Borland Enterprise Server AppServer Edition 6.0 is installed. The default is `Borland/BDP`. If the default directory is not accurate, use the ellipsis (…) button to select the correct directory.
- Native Executable Launcher: The native executable used to run this server. The default is `partition.exe` in the Home Directory's `bin` folder.
- VM Parameters: The parameters you want to pass to the virtual machine.
- Server Parameters: The parameters you want to pass to the server.
- Working Directory: The name and location of a working directory.

The General tab will look similar to this:



Project
User Home
Borland Enterprise Server AppServer Editio
Borland Enterprise Server AppServer Editio
Generic AppServer 1.0
JBoss 3.x+
Tomcat 3.3
Tomcat 4.0
Tomcat 4.1
Tomcat 5.0
WebLogic Application Server 6.x
WebLogic Application Server 7.x
WebLogic Platform Server 8.x
WebSphere Application Server 5.0
WebSphere Application Server Advanced E
WebSphere Application Server Single Serve
JBuilder

✓ Enable server

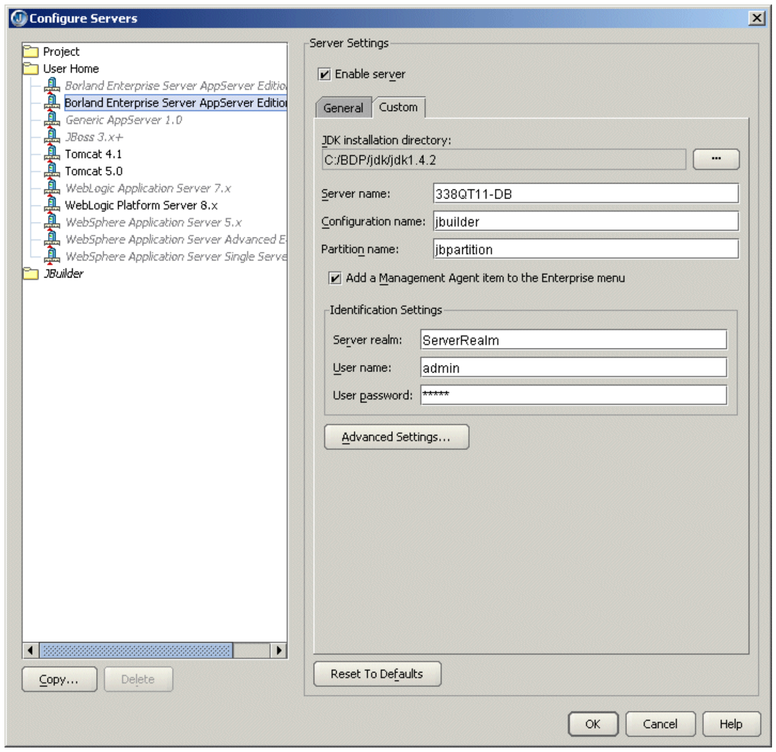
Home directory:	C:/BDP
Native executable launcher:	C:/BDP/bin/partition.exe
VM parameters:	
Server parameters:	
Working directory:	

4 Click the Custom tab to view and change (if required) fields unique to the server. Change or fill in these fields:

- JDK Installation Directory: The directory where the JDK version required by the server is located. JBuilder fills this field in automatically, depending on the value you specified for the Home Directory on the General page. For the Borland Enterprise Server AppServer Edition 6.0, this is the `jdk/1.4.2` folder. Note that your project will automatically be updated to use JDK 1.4.2.
- Server Name: The identifier of your machine.
- Configuration Name: The name of the configuration that manages the partition. For more information see "Configurations, partitions, partition services, and J2EE APIs" on page 34.
- Partition Name: The name of the partition in which the module is run. For more information see "Configurations, partitions, partition services, and J2EE APIs" on page 34.
- Add A Management Agent Item To The Enterprise Menu: Adds a Management Agent item to the JBuilder Enterprise menu, so you can start the Management Agent quickly from within JBuilder IDE.
- Server Realm: The realm of the server. For more information about the server realm, see your Borland Enterprise Server documentation.

- User Name: The name you use to identify yourself to the server. The default value is `admin`.
- User Password: The password you use to identify yourself to the server. The default fault is `admin`.
- Advanced Settings: Click this button to display the Advanced Settings dialog box. Use this dialog box to change the number of the port used by the Management Agent and to select the Use Security option. The management port is used in JBuilder to detect the server during startup and deployment. Change the management port only if you are deploying to a remote server with a management port that is different from the default. When you change the port number, ensure that you entered the correct port number as configured in the server, as the server won't start up without the correct port number. The port and security settings you choose must match the settings of your server. JBuilder updates the values in this dialog box whenever the Home Directory changes. The values are read from the Borland Enterprise Server property files. Changing the port number while you have the Management Agent started in JBuilder automatically shuts down the Management Agent.

The Custom tab will look similar to this:


Configure Servers
Project
User Home
Borland Enterprise Server AppServer Editio
Borland Enterprise Server AppServer Editio
Generic AppServer 1.0
JBoss 3.x+
Tomcat 4.1
Tomcat 5.0
WebLogic Application Server 7.x
WebLogic Platform Server 8.x
WebSphere Application Server 5.x
WebSphere Application Server Advanced E
WebSphere Application Server Single Serve
JBuilder
Copy...
Delete
Server Settings
Enable server
General
Custom
Identification Settings
Advanced Settings...
Reset To Defaults
OK
Cancel
Help

[✓] Enable server

JDK installation directory: C:/BDP/jdk/jdk1.4.2

Server name: 338QT11-DB

Configuration name: jbuilder

Partition name: jbpartition

[✓] Add a Management Agent item to the Enterprise menu

Server realm: ServerRealm

User name: admin

User password: *****

Borland Enterprise Server AppServer Edition 5.2.1

To configure JBuilder settings to target Borland Enterprise Server AppServer Edition 5.2.1,

1. Choose Enterprise|Configure Servers to display the Configure Servers dialog box.
2. Select Borland Enterprise Server AppServer Edition 5.2.1 from the User Home folder in the left pane.

   The right side of the dialog box displays the default settings for the selected server. The General page has fields that all servers have in common, while the Custom page has fields that are specific to the selected server. In some cases, modifying a Custom setting will update a setting on the General page.

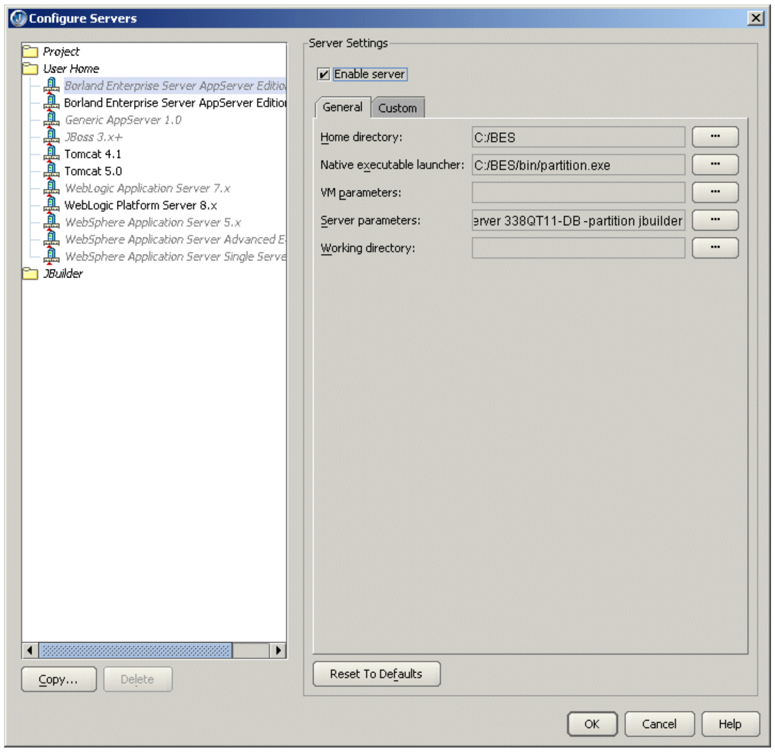
3 Select the Enable Server option at the top of the dialog box.

Checking this option enables the fields for the selected application server. You won't be able to edit any fields until it is checked. The Enable Server check box also determines whether this server will appear in the list of servers when you select a server for your project using Project|Project Properties|Servers.

4 Check the General tab to view and change (if required) default directories and parameter settings for the server. If JBuilder's default settings are not appropriate, edit any of the settings as needed. To change settings, click the ellipsis (…) button next to the field and make your changes.

- Home Directory: The directory where Borland Enterprise Server AppServer Edition 5.2.1 is installed. The default is `BES`. If the default directory is not accurate, use the ellipsis (…) button to select the correct directory.
- Native Executable Launcher: The native executable used to run this partition. The default is `partition.exe` in the Home Directory's `bin` folder.
- VM Parameters: The parameters you want to pass to the virtual machine.
- Server Parameters: The parameters you want to pass to the server. The default is `-partition jbuilder -server <server_name>`.
- Working Directory: The name and location of a working directory.

The General tab will look similar to this:


Configure Servers
Project
User Home
Borland Enterprise Server AppServer Editio
Borland Enterprise Server AppServer Editio
Generic AppServer 1.0
JBoss 3.x+
Tomcat 4.1
Tomcat 5.0
WebLogic Application Server 7.x
WebLogic Platform Server 8.x
WebSphere Application Server 5.x
WebSphere Application Server Advanced E
WebSphere Application Server Single Serve
JBuilder
Copy...
Delete
Server Settings
General
Custom
Reset To Defaults
OK
Cancel
Help

☑ Enable server

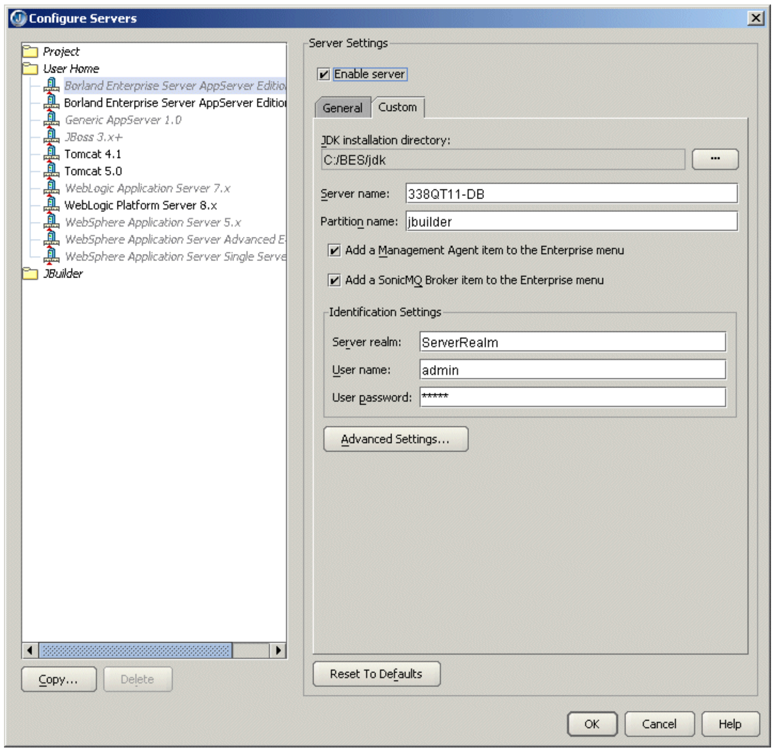
Home directory: C:/BES
Native executable launcher: C:/BES/bin/partition.exe
VM parameters:
Server parameters: erver 338QT11-DB -partition jbuilder
Working directory:

5 Click the Custom tab to view and change (if required) fields unique to the server. Change or fill in these fields:

- JDK Installation Directory: The directory where the JDK version required by the server is located. JBuilder fills this field in automatically, depending on the value you specified for the Home Directory on the General page. For the Borland Enterprise AppServer Edition 5.2.1, the directory is `jdk/jdk1.4.1` directory in the Home Directory. Note that this version of the Borland Enterprise Server can use either JDK 1.3 or JDK 1.4.
- Server Name: The name of the server you are configuring. By default, JBuilder specifies the identifier of your machine.

- Partition Name: The name of the partition in which the server is run. For more information about partitions, see "Configurations, partitions, partition services, and J2EE APIs" on page 34.
- Add A Management Agent Item To The Enterprise Menu: Adds a Management Agent item to the JBuilder Enterprise menu, so you can start the Management Agent quickly from within JBuilder IDE.
- Add A SonicMQ Broker Agent Item To The Enterprise Menu: Adds the SonicMQ Broker Agent item to the JBuilder Enterprise menu, so you can start the SonicMQ Broker Agent quickly from within JBuilder IDE.
- Server Realm: The realm of the server. For more information about the server realm, see your Borland Enterprise Server documentation.
- User Name: The name you use to identify yourself to the server. The default value is `admin`.
- User Password: The password you use to identify yourself to the server. The default fault is `admin`.
- Advanced Settings: Click this button to display the Advanced Settings dialog box. Use this dialog box to change the number of the port used by the Management Agent and to select the Use Security option. The management port is used in JBuilder to detect the server during startup and deployment. Change the management port only if you are deploying to a remote server with a management port that is different from the default. When you change the port number, please ensure that you entered the correct port number as configured in the server, as the server won't start up without the correct port number. The port and security settings you choose must match the settings of your server. JBuilder updates the values in this dialog box whenever the Home Directory changes. The values are read from the Borland Enterprise Server property files. Changing the port number while you have the Management Agent started in JBuilder automatically shuts down the Management Agent.

The Custom tab will look similar to this:


Configure Servers
Project
User Home
Borland Enterprise Server AppServer Editio
Borland Enterprise Server AppServer Editio
Generic AppServer 1.0
JBoss 3.x+
Tomcat 4.1
Tomcat 5.0
WebLogic Application Server 7.x
WebLogic Platform Server 8.x
WebSphere Application Server 5.x
WebSphere Application Server Advanced E
WebSphere Application Server Single Serve
JBuilder
Server Settings
General
Custom
Advanced Settings...
Copy...
Delete
Reset To Defaults
OK
Cancel
Help

[✓] Enable server

JDK installation directory: C:/BES/jdk

Server name: 338QT11-DB

Partition name: jbuilder

[✓] Add a Management Agent item to the Enterprise menu

[✓] Add a SonicMQ Broker item to the Enterprise menu

Identification Settings

Server realm: ServerRealm

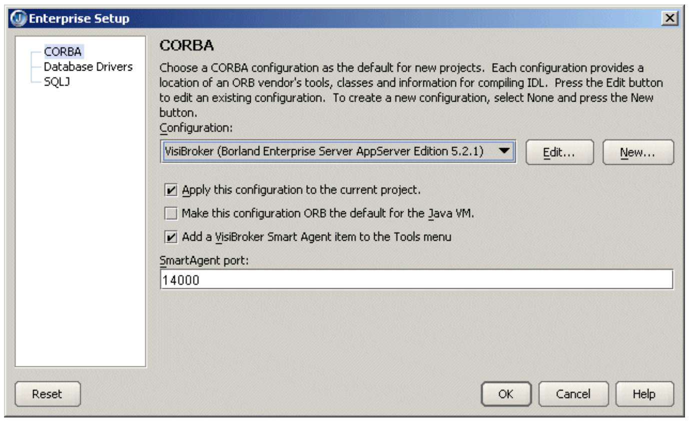
User name: admin

User password: *****

Making the ORB available to JBuilder

You use the CORBA node of the Enterprise Setup dialog box (Enterprise|Enterprise Setup) to set up the Borland Enterprise Server for use with VisiBroker. For Borland Enterprise Server AppServer Editions 6.0 and 5.2.1, the server will create its own VisiBroker configuration called “VisiBroker <server name and version>.” The default VisiBroker configuration won’t be changed.

You also use this dialog box to set the VisiBroker Smart Agent port to a unique number. All client/server communication takes through the Smart Agent port. To start the Smart Agent from the Tools menu, check the Add The VisiBroker Smart Agent Item To The Tools Menu option. This option is selected by default. The CORBA node of the Enterprise Setup dialog box looks like this:



CORBA

Choose a CORBA configuration as the default for new projects. Each configuration provides a location of an ORB vendor's tools, classes and information for compiling IDL. Press the Edit button to edit an existing configuration. To create a new configuration, select None and press the New button.

Configuration: VisiBroker (Borland Enterprise Server AppServer Edition 5.2.1)

[✓] Apply this configuration to the current project.

[ ] Make this configuration ORB the default for the Java VM.

[✓] Add a VisiBroker Smart Agent item to the Tools menu

SmartAgent port: 14000

To make the ORB available to JBuilder, you must start the VisiBroker Smart Agent with the Enterprise|VisiBroker Smart Agent command. This handles the initial bootstrap issues such as how the client locates the naming service and so on.

For Borland Enterprise Server AppServer Edition 5.2.1/6.0: If you choose Enterprise|Borland Enterprise Server Management Agent, the VisiBroker Smart Agent is started as part of the process. When you are working with the Borland Enterprise Server AppServer Edition 5.2.1/6.0, you must start the Borland Enterprise Server Management Agent instead of the VisiBroker Smart Agent. If you don’t start it, it is automatically started for you when you start the server within JBuilder. Note that starting the server automatically might take longer because the server searches for any running agents. The most efficient way to start the Management Agent is to start it yourself.

For more information on setting up the ORB for use within JBuilder, see “Setting up JBuilder for CORBA applications” on page 47.

Configurations, partitions, partition services, and J2EE APIs

The architecture of the Borland Enterprise Server AppServer Edition 6.0 and 5.2.1 varies significantly. The Borland Enterprise Server AppServer Edition 6.0 introduces the concept of *management hubs* and *configurations*. A management hub, also known as the *server control unit,* controls, by server (or machine identifier), one or more configurations. A configuration, in turn, manages one or more running *partitions,* or processes. A configuration is based on an XML configuration file that resides in the configuration’s root directory. This file determines the order of partition startup. A configuration can have multiple partitions. Because each partition is a separate process, an application’s functions can be distributed across multiple processes.

In the Borland Enterprise Server AppServer Edition 5.2.1, a server, also a machine identifier, manages a partition. A server can also have multiple partitions. And, as with

Borland Enterprise Server AppServer Edition 6.0, because each partition is a separate process, an application's functions can be distributed across multiple processes. For either edition, when you use the Configure Servers dialog box to configure the server, all these pieces are seamlessly set up for use in JBuilder.

For more information on the Borland Enterprise Server architecture, refer to the documentation, available from the Help menu of the Borland Management Console.

In both editions, partitions provide containers and services for hosting your applications. Any or all of a partition's containers and services can be enabled or disabled. Additionally, a partition can be cloned (copied) on the same configuration or server or on another enterprise configuration or server located on the same local area network. Cloning a partition can save significant time in the deployment process because it not only copies all of the deployed modules, but also copies all the associated service configurations settings which can require a considerable amount of time and effort to complete.

Application components are hosted in either the web container or EJB container, or simply in the partition itself (in the case of connectors). You can deploy application EARs or smaller archives to partitions. The partitions are "smart" and will place your application components in the proper containers. Since you may create as many partitions as you wish, you can also have as many container instances as you wish. Borland provides up to two different containers per partition instance:

- Web container (Tomcat 4.x or 5.0): for hosting JSPs and servlets, and
- EJB container (Borland): for hosting EJB components

Each partition also provides Partition Services and J2EE APIs as Partition Services. These services and APIs are:

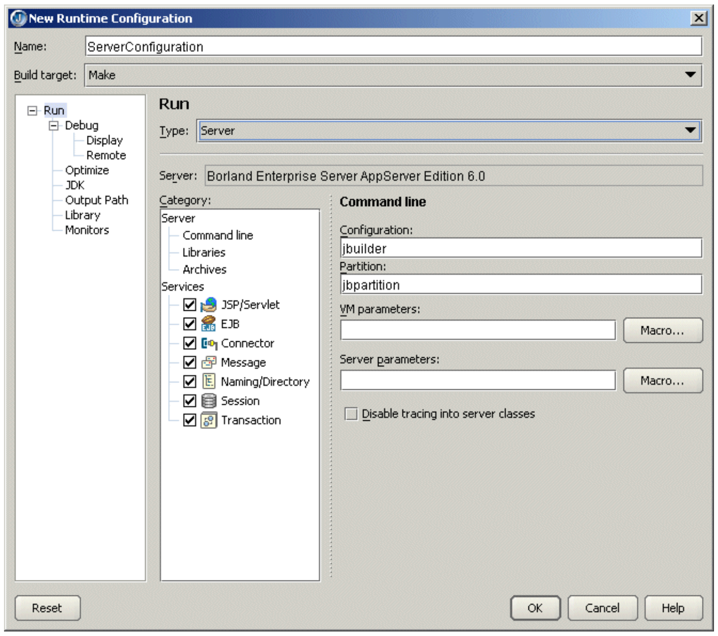
- Naming Service: In Borland Enterprise Server, the Naming Service is managed by the VisiBroker ORB.
- JDataStore: Borland's all-Java database.
- JDBC: For getting connections to and modeling data from databases.
- Java Mail: A Java email service.
- JTA: The Java transactional API.
- JAXP: The Java API for XML parsing.
- JNDI: The Java Naming and Directory interface.
- RMI-IIOP: Remote method invocation (RMI) carried out via internet inter-ORB protocol (IIOP).

JBuilder, by default, uses a partition named `jbpartition` and a configuration named `jbuilder` as a deployment target for Borland Enterprise Server AppServer Edition 6.0. For Borland Enterprise Server AppServer Edition 5.2.1, the partition name is `jbuilder`. If the configuration or partition doesn't exist, one will be created for you automatically. For both editions, the default partition shares the same port number with Tomcat and JDataStore services. Additionally, in both editions, the server name is the same as the machine identifier.

You can start multiple partitions using multiple JBuilder run configurations. To create multiple run configurations, follow these steps:

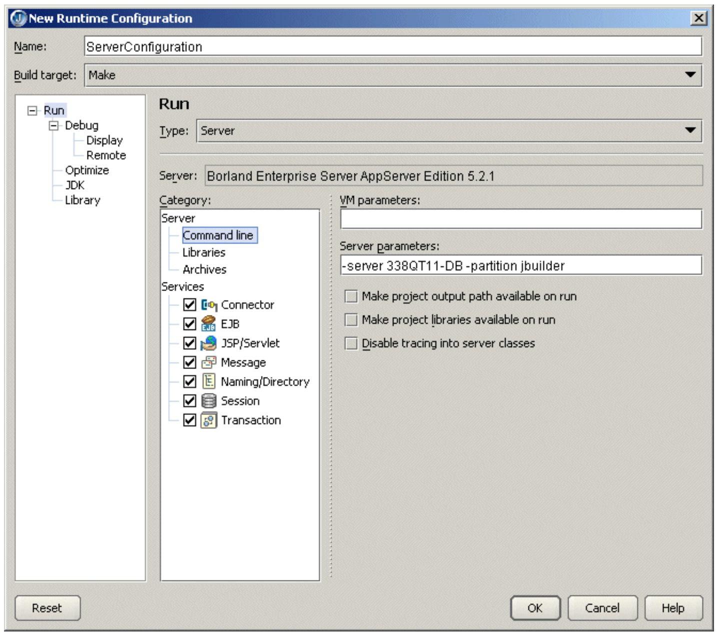
1. Choose Run|Configurations, then click New.
2. Change the Run Type to Server. The displayed server is the server selected for the project in Project Properties|Server.
3. In the Category tree, select Server|Command Line.

4 For Borland Enterprise Server 6.0, change the Partition And Configuration Name to the one you want to use. The Command Line page looks like this:



Name: ServerConfiguration
Build target: Make
Type: Server
Server: Borland Enterprise Server AppServer Edition 6.0
Configuration: jbuilder
Partition: jbpartition
VM parameters:
Server parameters:
☐ Disable tracing into server classes
☑ JSP/Servlet ☑ EJB ☑ Connector ☑ Message ☑ Naming/Directory ☑ Session ☑ Transaction

For Borland Enterprise Server 5.2.1, edit the Server Command Line parameters and change the partition name to the one you want to use, as show in the following figure:



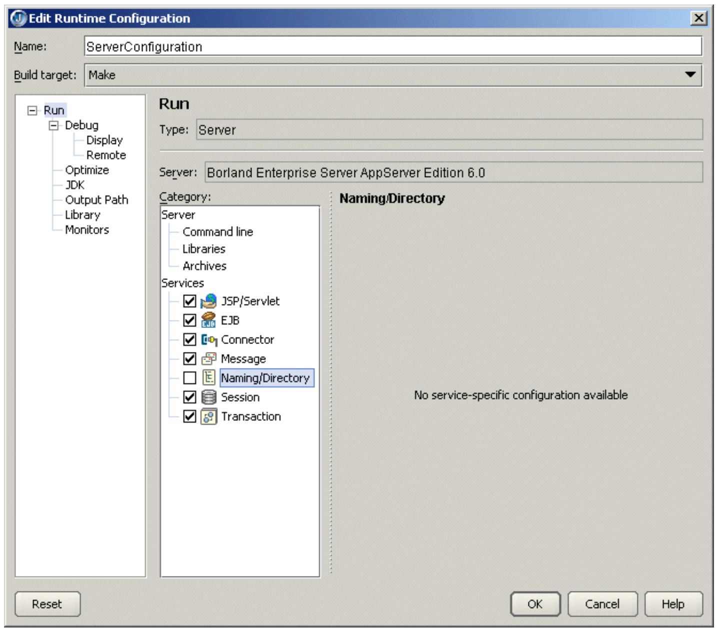
Name: ServerConfiguration
Build target: Make
Type: Server
Server: Borland Enterprise Server AppServer Edition 5.2.1
VM parameters:
Server parameters: -server 338QT11-DB -partition jbuilder
☐ Make project output path available on run
☐ Make project libraries available on run
☐ Disable tracing into server classes
☑ Connector ☑ EJB ☑ JSP/Servlet ☑ Message ☑ Naming/Directory ☑ Session ☑ Transaction

5 Click OK to close the New Runtime Configuration dialog box and save the configuration.

6 Repeat these steps to create additional run configurations to run other partitions.

Before starting up the partition, make sure you have configured unique port numbers for Tomcat and JDataStore services.

Also ensure that you have the naming service enabled for only one of the partitions to avoid naming service conflicts. To disable the naming service for a partition, edit the run configuration for the partition (Edit Runtime Configuration dialog box) and uncheck the naming service option, as shown in the following figure:



Name: ServerConfiguration
Build target: Make
Run
Type: Server
Server: Borland Enterprise Server AppServer Edition 6.0
Category:
Server
Command line
Libraries
Archives
Services
☑ JSP/Servlet
☑ EJB
☑ Connector
☑ Message
☐ Naming/Directory
☑ Session
☑ Transaction

Naming/Directory

No service-specific configuration available

Changing the management port

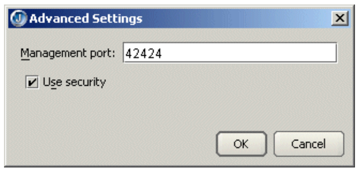
The management agent manages all partitions. The default management port, set on the Advanced Settings dialog box for both Borland Enterprise Server AppServer Editions 6.0 and 5.2.1, is 42424. You can change the management port used by JBuilder, or the one used by the server.

To change the port used by JBuilder,

1 Choose Enterprise|Configure Servers and chose Borland Enterprise Server AppServer Edition 6.0 or 5.2.1 from the User Home Folder on the left.

2 Click the Custom tab, then the Advanced Settings button.

3 Change the port in the Management Port field, as shown in the following figure:



Management port: 42424
☑ Use security

4 Click OK two times.

To change the management port used by the server,

1 Open the Borland Management Console.

2 Double-click Installations.

3 For Borland Enterprise Server 6.0, choose Wizards|Configure Agents. For Borland Enterprise Server 5.2.1, right-click the server you want to change the port for.

4 Select Update Ports.

5 Use the Port Configuration wizard to change the Management Port number. The User Port value is equal to the Smart Agent Port in JBuilder.

After you change the management port, the management agent will shutdown if it was started in JBuilder.

Starting the configuration or server

Before any server related actions (server startup, deployment, and so on) are performed, you must start the management agent with Enterprise|Borland Enterprise Server Management Agent (use the version of this command that matches the version of the Borland Enterprise Server AppServer Edition your project is targeted for.). For the Borland Enterprise Server 6.0, this will start the management hub. For Borland Enterprise Server 5.2.1, this starts the server.

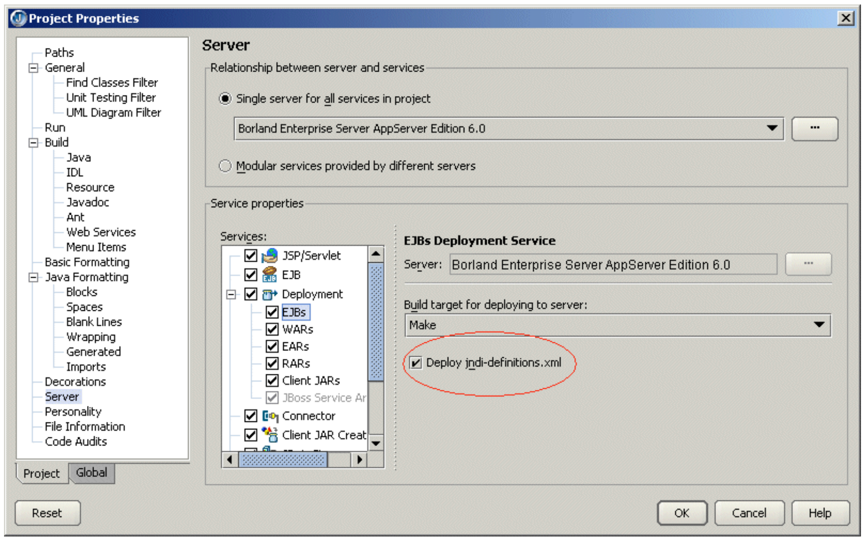
For both editions, after the management agent has started up, you can launch the partition by right-clicking the EJB or web module in the project pane and selecting Run Using Defaults. You can also create a Server run configuration and run the project using that configuration.

Running the partition in JBuilder will

- Create the partition (and configuration for Borland Enterprise Server 6.0), if not already present. The partition name (and the configuration name for Borland Enterprise Server 6.0) used in this step is derived from the run configuration used to start up the server. If you are starting the configuration or server using the default configuration, the partition name as configured in the application server properties will be used.
- Deploy any resources defined in `jndi-definitions.xml` (if present) to the root of the partition directory. For the Borland Enterprise Server AppServer Edition 6.0, this directory is: `PARTITION_NAME\dars\jbuilder.dar`. For Borland Enterprise Server 6.0 only, the `jndi-definitions.xml` file is packaged into a `.dar` file and deployed. For the Borland Enterprise Server AppServer Edition 5.2.1, it is: `APPSERVER_HOME\var\servers\SERVER_NAME\partitions\PARTITION_NAME\`.

  This file will be created if your project contains an EJB 2.0 module with data sources/messaging resources defined.

Note This action can be turned off by unchecking the Deploy jndi-definitions.xml option in the Deployment EJBs Services properties on the Server node of the Project Properties dialog box.



Server

Relationship between server and services

(●) Single server for all services in project

Borland Enterprise Server AppServer Edition 6.0

( ) Modular services provided by different servers

Service properties

Services:
- ✓ JSP/Servlet
- ✓ EJB
- ✓ Deployment
  - ✓ EJBs
  - ✓ WARs
  - ✓ EARs
  - ✓ RARs
  - ✓ Client JARs
  - ✓ JBoss Service Ar
- ✓ Connector
- ✓ Client JAR Creat

EJBs Deployment Service

Server: Borland Enterprise Server AppServer Edition 6.0

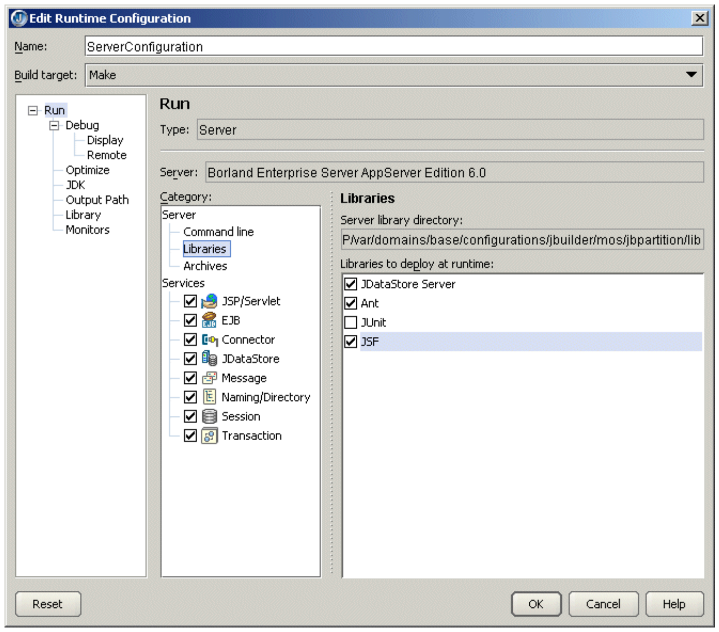
Build target for deploying to server: Make

✓ Deploy jndi-definitions.xml

- Copy selected libraries to the partition's `lib` directory. All archives in this directory will be added to the partition's classpath when the partition is started up.

  The directory on the server for Borland Enterprise Server AppServer Edition 6.0 is `APPSERVER_HOME\var\domains\base\configurations\CONFIGURATION_NAME\mos\PARTITION_NAME\lib`. For the Borland Enterprise Server AppServer Edition 5.2.1, the directory is `APPSERVER_HOME\var\servers\SERVER_NAME\partitions\PARTITION_NAME\lib`.

The libraries to be copied can be set with the Libraries To Deploy At Runtime list on the Edit Runtime Configuration dialog box for the server run configuration:



Edit Runtime Configuration

Name: ServerConfiguration

Build target: Make

- Run
  - Debug
    - Display
    - Remote
  - Optimize
  - JDK
  - Output Path
  - Library
  - Monitors

Run

Type: Server

Server: Borland Enterprise Server AppServer Edition 6.0

Category:

- Server
  - Command line
  - Libraries
  - Archives
- Services
  - ☑ JSP/Servlet
  - ☑ EJB
  - ☑ Connector
  - ☑ JDataStore
  - ☑ Message
  - ☑ Naming/Directory
  - ☑ Session
  - ☑ Transaction

Libraries

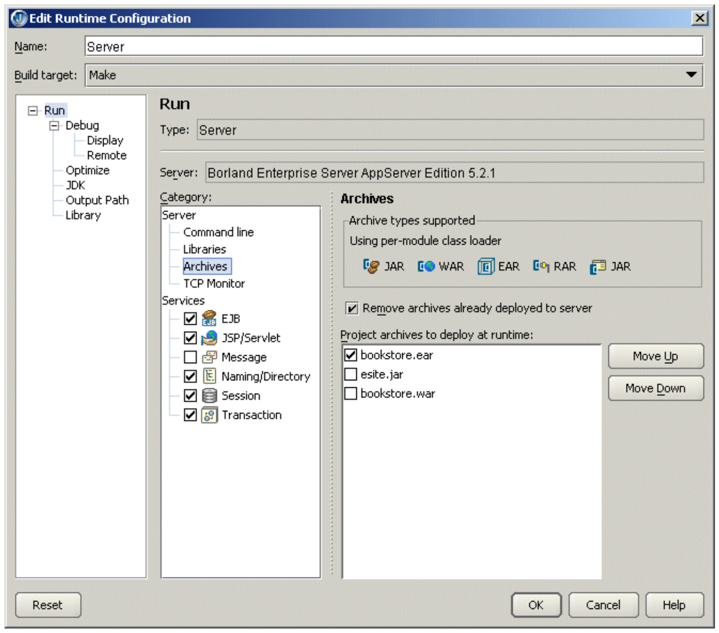
Server library directory: P:/var/domains/base/configurations/jbuilder/mos/jbpartition/lib

Libraries to deploy at runtime:

- ☑ JDataStore Server
- ☑ Ant
- ☐ JUnit
- ☑ JSF

Reset | OK | Cancel | Help

- Remove any archives deployed to the partition, if this option is selected. This can be set from the Archives Server category on the Edit Runtime Configuration dialog box for the server run configuration. If you want to have a clean partition, select the Remove Archives Already Deployed To Server option.



Edit Runtime Configuration

Name: Server

Build target: Make

- Run
  - Debug
    - Display
    - Remote
  - Optimize
  - JDK
  - Output Path
  - Library

Run

Type: Server

Server: Borland Enterprise Server AppServer Edition 5.2.1

Category:

- Server
  - Command line
  - Libraries
  - Archives
  - TCP Monitor
- Services
  - ☑ EJB
  - ☑ JSP/Servlet
  - ☐ Message
  - ☑ Naming/Directory
  - ☑ Session
  - ☑ Transaction

Archives

Archive types supported

Using per-module class loader

JAR WAR EAR RAR JAR

☑ Remove archives already deployed to server

Project archives to deploy at runtime:

- ☑ bookstore.ear
- ☐ esite.jar
- ☐ bookstore.war

Move Up | Move Down

Reset | OK | Cancel | Help

- Deploy archives if selected. By default all deployable archives in the project are selected. To change this, click the Archives Server category in the Server run configuration (Edit Runtime Configuration dialog box) and unselect any archives that you do not wish to deploy.
- Start the partition. When partition startup is complete, you should see the partitions listed in the message pane for Borland Enterprise Server AppServer Edition 6.0 or the message 'Partition <PARTITION_NAME> has been started' for Borland Enterprise Server AppServer Edition 5.2.1. Archives deployed at startup should be loaded and accessible.

Note By default, all services associated with a partition are started. This could cause the partition to take a longer time to start. To reduce startup time, uncheck the services that you don't require in the Server run configuration.

Remote deploying

To prepare to remote deploy EJBs, WARs, and EAR modules for the Borland Enterprise Server AppServer Edition 6.0, follow these steps.

1. Choose Enterprise|Configure Servers.
2. Select Borland Enterprise Server AppServer Edition 6.0 from the left side of the dialog box.
3. Click the Custom tab and set the Server, Configuration, and Partition names to match that of the remote server.
4. Click the Advanced Settings button and make sure the Management Port matches your server configuration.
5. Click OK two times.
6. Choose Enterprise|Enterprise Setup.
7. Select the CORBA node to display the CORBA page, and set the Smart Agent Port to match that of your server configuration.
8. Click OK.

To prepare to remote deploy EJBs, WARs, and EAR modules for the Borland Enterprise Server AppServer Edition 5.2.1, follow these steps.

1. Choose Enterprise|Configure Servers.
2. Select Borland Enterprise AppServer Edition 5.2.1 from the left side of the dialog box.
3. Click the Custom tab and set the Server and Partition names to match that of the remote server.
4. Click Advanced Settings and make sure the Management Port matches your server configuration.
5. Click OK two times.
6. Choose Enterprise|Enterprise Setup.
7. Select the CORBA node to display the CORBA page, and set the Smart Agent Port to match that of your server configuration.
8. Click OK.

Now you are ready to deploy. There are two ways to deploy EJBs, WARs, and EAR modules using JBuilder:

- Deploying to the server using the Borland Enterprise Server Deployment Wizard (Enterprise|Server Deployment). This is a Borland Enterprise Server tool and requires that the management agent is running. On the first page of the wizard, select the modules to deploy. The wizard then detects all partitions on the server, even if they have not been started. Select the appropriate partition from the list and click Finish to deploy the archive. If the partition has already been started, restart the partition to access the deployed archive.
- Deploying to the server using the context menu option on any deployable node. Right-click the deployable node to see the Deploy commands. Choose Deploy. You can also select multiple deployable nodes, right-click them, and use the context menu that appears to deploy more than one module.

Remote debugging

Before you can debug your application remotely, you need to configure the partition. You can do this before you start the server or while the server and partition are already running. See these sections for more information:

- "Preparing to remote debug partitions that are not managed in JBuilder" on page 41
- "Preparing to remote debug partitions that are managed in JBuilder" on page 41
- "Preparing to remote debug before the Borland Enterprise Server AppServer Edition 5.2.1 is running" on page 42
- "Preparing to remote debug when the Borland Enterprise Server AppServer Edition 5.2.1 is already running" on page 42

Once the configuration is complete and the server and partition are started, follow the instructions in "Remote debugging in JBuilder" on page 43.

Remote debugging for Borland Enterprise Server 6.0

Preparing to remote debug partitions that are not managed in JBuilder

To prepare to remote debug partitions that are not managed in JBuilder,

1. Start the Borland Management Console.
2. Start the server.
3. Locate the partition you wish to debug.
4. Right-click the partition and choose Properties.
5. Select the Partition Settings tab.
6. Check the Enable JPDA Remote Debugging option.
7. Set the transport address field to 3999.
8. Uncheck the Suspend Partition Until Debugger Attaches option.
9. Click OK.

Preparing to remote debug partitions that are managed in JBuilder

To prepare to remote debug partitions that are managed in JBuilder, you have two options:

Starting the server/configuration/partition outside of JBuilder

1. Shut down the server.
2. Open the file `<APPSERVER_HOME>/var/domains/base/configurations/configuration.xml`.
3. Look for the JPDA element and edit attribute values as follows:

```
enable-jpda-debug="true"
jpda-transport-address="3999"
jpda-suspend="false"
```

4. Start the server/configuration/partition outside of JBuilder.

Starting the management agent/partition in JBuilder

1. Shut down the server.
2. Open the file `<APPSERVER_HOME>/var/domains/base/configurations/configuration.xml`.

3 Look for the JPDA element and edit attribute values as follows:

```
enable-jpda-debug="true"
jpda-transport-address="3999"
jpda-suspend="false"
```

4 Choose Run|Configurations, select the server run configuration, and click the Edit button.

5 Set the following server parameters: `-jpda:address=3999`

6 Start the management agent/partition in JBuilder.

Remote debugging for Borland Enterprise Server 5.2.1

Preparing to remote debug before the Borland Enterprise Server AppServer Edition 5.2.1 is running

To set up for remote debugging before you start the Borland Enterprise Server AppServer Edition 5.2.1,

1 Start the Borland Management Console.

2 Click the Installations button.

3 Locate the server you will start. Expand the node and locate the partition you wish to debug in JBuilder.

4 Click the partition node in the structure pane.

5 Search for and set the debug parameters as follows in the content pane (set the values that are in bold-faced type):

```
partition.enable_jpda_debug=true
partition.jpda.transport_address=3999
partition.jpda.suspend=false
```

6 Click Apply Changes.

Preparing to remote debug when the Borland Enterprise Server AppServer Edition 5.2.1 is already running

To set up for remote debugging when the Borland Enterprise Server AppServer Edition 5.2.1 is already running,

1 Start the Borland Management Console.

2 Click the Servers button.

3 Expand the Enterprise Servers node and locate the partition you want to debug.

4 Right-click and select Configure.

5 Choose the JPDA tab.

6 Check the Enable JPDA Remote Debugging option.

7 Set the transport address field to 3999.

8 Uncheck the Suspend Partition Until Debugger Attaches option.

9 Click Apply Changes.

Remote debugging in JBuilder

Once either edition of the Borland Enterprise Server AppServer and the partition have been started, follow these steps from within JBuilder:

1 In the project from which you want to launch the remote debug session, choose Run|Configurations. If you have not yet created a Server type run configuration, click New. In the New Runtime Configuration dialog box, set the Type to Server. If you have already created a Server run configuration, select it and choose Edit.

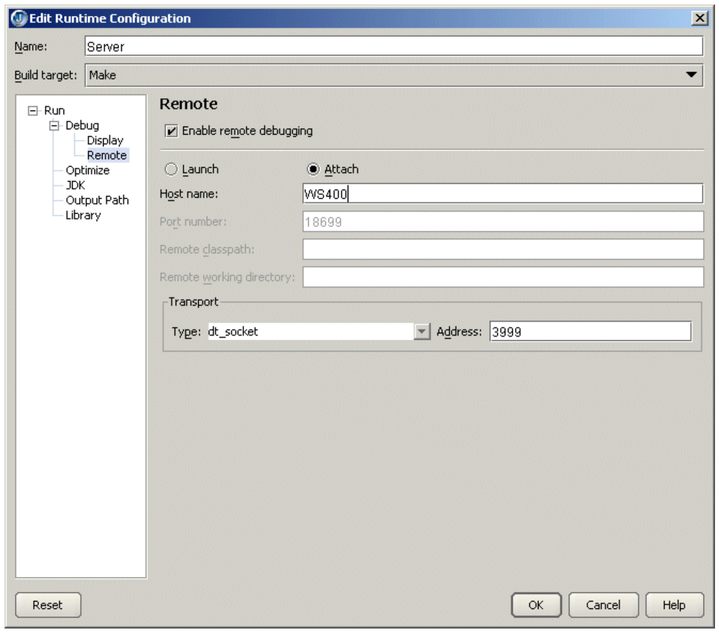
2 Select the Debug|Remote node.

3 Check the Enable Remote Debugging option.

4 Select the Attach option.

5 Enter the name of the machine on which the server is running in the Host Name field.

6 Make sure the Transport Type is set to `dt_socket` and the Address is set to `3999`. The dialog box will look similar to this:



Name: Server
Build target: Make
[✓] Enable remote debugging
( ) Launch (●) Attach
Host name: WS400
Port number: 18699
Remote classpath:
Remote working directory:
Transport
Type: dt_socket Address: 3999

7 Click OK to close the dialog box.

8 Set a breakpoint in the process you want to debug.

9 Deploy that process to the running partition.

10 Click the down arrow next to the Debug Project button on the toolbar and select the Server configuration you just created or edited. The debugger launches, attaches to the partition running remotely, and stops at the breakpoint.

Web module workarounds

The default application server install includes `ROOT.war`, which contains the default context. If you deploy an EAR that contains a default context, you must delete `ROOT.war` (or rename its extension) so that it does not load and cause a conflict. If deploying WARs (with the container classloader policy), the resulting WAR is automatically copied to the partition as `ROOT.war`, in which case there is no conflict. Please note that it is the `<context-root>!ROOT!</context-root>` element in the `web-borland.xml` file that designates the context as root, not the file name.

If you are working on a named context but have the root context in an already deployed archive, the internal web browser in JBuilder will attempt to load once that archive is loaded by the container, because the root context can potentially match any URL. You'll then probably get a "Document not found" error. To prevent this, remove `ROOT.war` if you're not using it or overwriting it.

Displaying the Borland Management Console in JBuilder

You can display the Borland Management Console in JBuilder. Choose View|Panes|BES Console.

International issues

The Borland Enterprise Server AppServer Editions 6.0 and 5.2.1 do support Japanese characters in the JNDI name.

Borland servers and JDataStore 7.0

The Borland Enterprise Servers 5.2.1 and 6.0 do not support JDataStore 7.0.1, although JBuilder does. When you are running applications that access JDataStore databases, be sure you are using the JDataStore 6.x or older file format and JDataStore libraries that ship with the server. Databases created using JDataStore 7.0 cannot be used with Borland Enterprise Server 5.2.1 and 6.0.

Chapter

6

Using JBuilder’s CORBA tools

This chapter explains the CORBA (Common Object Request Broker Architecture) tools available in JBuilder, including the VisiBroker ORB and IDL compilers.

The Common Object Request Broker Architecture (CORBA) allows distributed applications to interoperate (application to application communication), regardless of what language they are written in or where these applications reside.

The CORBA specification was adopted by the Object Management Group to address the complexity and high cost of developing distributed object applications. CORBA uses an object-oriented approach for creating software components that can be reused and shared between applications. Each object encapsulates the details of its inner workings and presents a well defined interface, which reduces application complexity. The cost of developing applications is reduced, because once an object is implemented and tested, it can be used over and over again.

The Object Request Broker (ORB) connects a client application with the objects it wants to use. The client program does not need to know whether the object implementation it is in communication with resides on the same computer or is located on a remote computer somewhere on the network. The client program only needs to know the object’s name and understand how to use the object’s interface. The ORB takes care of the details of locating the object, routing the request, and returning the result.

Note The ORB itself is not a separate process. It is a collection of libraries and network resources that integrates within end-user applications, and allows your client applications to locate and use objects.

JBuilder supports the VisiBroker ORB and the OrbixWeb ORB. JBuilder also provides tools to configure other third-party ORBs.

How JBuilder and the VisiBroker ORB work together

The VisiBroker ORB provides a complete CORBA ORB runtime and supporting development environment for building, deploying, and managing distributed Java applications that are open, flexible, and inter-operable. The VisiBroker ORB is part of the Borland Enterprise Server. Objects built with the VisiBroker ORB are easily accessed by Web-based applications that communicate using OMG’s Internet Inter-ORB Protocol (IIOP) standard for communication between distributed objects through the Internet or through local intranets. The VisiBroker ORB has a native implementation of IIOP to ensure high-performance and inter-operability.

The VisiBroker ORB incorporates the following compilers for working in a Java environment:

- `java2iiop`

  The `java2iiop` compiler allows you to stay in an all-Java environment. The `java2iiop` compiler takes your Java interfaces and generates IIOP-compliant stubs and skeletons. Part of the advantage of using the `java2iiop` compiler is that, through the use of extensible structs, you can pass Java serialized objects by value.

- `java2idl`

  The `java2idl` compiler turns your Java code into IDL, allowing you to generate client stubs in the programming language of your choice. In addition, because this compiler maps your Java interfaces to IDL, you can re-implement Java objects in another programming language that supports the same IDL.

When you develop distributed applications with JBuilder and the VisiBroker ORB, you must first identify the objects required by the application. You will then usually follow these steps:

1. Write a specification for each object using the IDL wizard to generate the Interface Definition Language (IDL) file. IDL is the language an implementer uses to specify the operations an object will provide and how they should be invoked.
2. Use the IDL compiler to generate the client stub code and server Portable Object Adapter (POA) servant code.
3. Write the client program code.
4. Write the server object code.
5. Compile the client and server code.
6. Start the server.
7. Run the client program.

For in-depth information on using the VisiBroker ORB to create distributed applications, see the VisiBroker documentation in the `doc` directory of your Borland Enterprise Server installation.

For more information about Common Object Request Broker Architecture (CORBA), some useful links are:

- `http://www.borland.com/books` — Books on Borland products and related technologies
- *"A White Paper: Java, RMI, and CORBA"* at `http://www.omg.org/news/whitepapers/wpjava.htm`
- *CORBA/IIOP Specification* in *Introduction to OMG's Specifications* at `http://www.omg.org/gettingstarted/specintro.htm`
- *CORBA Basics* at `http://www.omg.org/gettingstarted/corbafaq.htm`

RMI

Another way to invoke methods on remote Java objects is to use Remote Method Invocation (RMI). The Borland Enterprise Server provides capabilities equivalent to RMI, but, in addition, allows you to create Java objects that communicate with all other objects that are CORBA compliant even if they are not written in Java. For an example of a Java interface that describes an RMI remote interface, see the sample file `Account.java` in the `examples/vbe/rmi-iiop/Bank` directory of your Borland Enterprise Server installation.

Setting up JBuilder for CORBA applications

This section explains how to set up JBuilder with the VisiBroker ORB or OrbixWeb so that you can create, run, and deploy CORBA applications. You can also configure other, third-party ORBs. More information on the VisiBroker ORB is available in the `doc` directory of your Borland Enterprise Server installation. Consult the OrbixWeb documentation for information on particular features of that ORB.

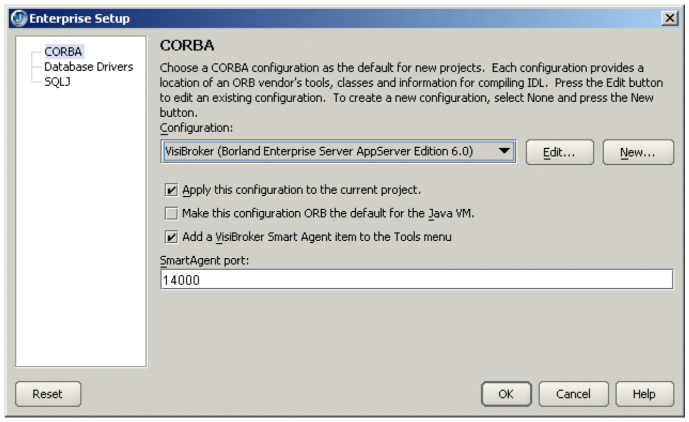
Configuring VisiBroker when the Borland Enterprise Server is installed

Follow the steps in this section to set up JBuilder for use with VisiBroker when the Borland Enterprise Server AppServer Edition 6.0 is already installed, configured and the target server for your project. Note that the server creates its own VisiBroker configuration. For information on configuring Borland Enterprise Server, see Chapter 5, "Using JBuilder with Borland servers."

To configure VisiBroker when the Borland Enterprise Server is installed,

1. Choose Enterprise|Enterprise Setup to display the Enterprise Setup dialog box. Select the CORBA page. The parameters in this dialog box allow JBuilder to see the ORB.
2. Select the VisiBroker (Borland Enterprise Server AppServer Edition 6.0) option from the Configuration drop-down list.

   The Enterprise Setup dialog box will look similar to this:



Enterprise Setup

CORBA
Database Drivers
SQLJ

CORBA

Choose a CORBA configuration as the default for new projects. Each configuration provides a location of an ORB vendor's tools, classes and information for compiling IDL. Press the Edit button to edit an existing configuration. To create a new configuration, select None and press the New button.

Configuration: VisiBroker (Borland Enterprise Server AppServer Edition 6.0)

Edit... New...

[✓] Apply this configuration to the current project.

[ ] Make this configuration ORB the default for the Java VM.

[✓] Add a VisiBroker Smart Agent item to the Tools menu

SmartAgent port: 14000

Reset OK Cancel Help

3. Click OK to close the dialog box. You do not have to complete other configuration steps; JBuilder is now configured for use with VisiBroker.
4. Click OK to save the configuration.

Important In the runtime configuration for your CORBA application, you need to add the following parameters to the VM Parameters field in the Edit Runtime Configuration dialog box (Run|Configurations|Edit):

```
-Dvbroker.agent.port=14000
-Dborland.enterprise.licenseDir=C:/<BES_install>/var
-Dborland.enterprise.licenseDefaultDir=C:/<BES_install>/license
```

You've now completed setting up your system to use the VisiBroker version installed with the Borland Enterprise Server AppServer Edition 6.0. Before running your application, choose Tools|VisiBroker Smart Agent to start the Smart Agent. (Note that it is also automatically started when you start the Management Agent with the Enterprise| Borland Enterprise Server Management Agent command.)

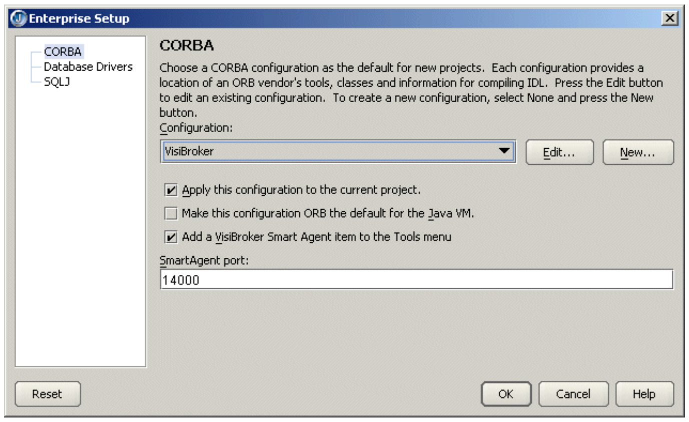
Configuring VisiBroker stand-alone

Follow the steps in this section to set up JBuilder for use with VisiBroker stand-alone; that is, without targeting your project for the Borland Enterprise Server AppServer Edition 6.0.

To configure VisiBroker stand-alone,

1 Choose Enterprise|Enterprise Setup to display the Enterprise Setup dialog box. Select the CORBA page. The parameters in this dialog box allow JBuilder to see the ORB.

2 Select the VisiBroker option from the Configuration drop-down list.



CORBA

Choose a CORBA configuration as the default for new projects. Each configuration provides a location of an ORB vendor's tools, classes and information for compiling IDL. Press the Edit button to edit an existing configuration. To create a new configuration, select None and press the New button.

Configuration: VisiBroker

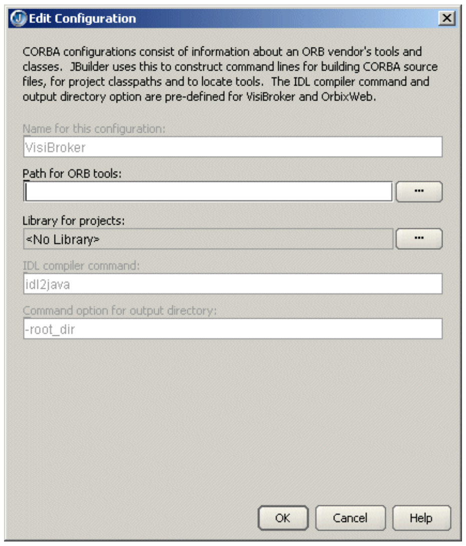
☑ Apply this configuration to the current project.

☐ Make this configuration ORB the default for the Java VM.

☑ Add a VisiBroker Smart Agent item to the Tools menu

SmartAgent port: 14000

3 Click the Edit button to display the Edit Configuration dialog box.



CORBA configurations consist of information about an ORB vendor's tools and classes. JBuilder uses this to construct command lines for building CORBA source files, for project classpaths and to locate tools. The IDL compiler command and output directory option are pre-defined for VisiBroker and OrbixWeb.

Name for this configuration: VisiBroker

Path for ORB tools:

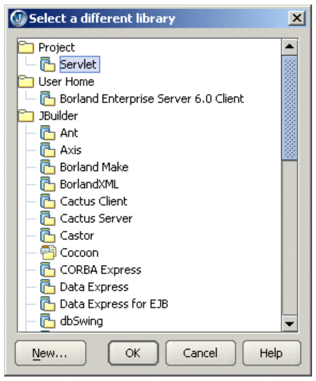
Library for projects: <No Library>

IDL compiler command: idl2java

Command option for output directory: -root_dir

4 The Path For ORB Tools field allows JBuilder to access ORB tools. Click the ellipsis (...) button next to the Path For ORB Tools field to browse to the directory containing the ORB tools. Point to the directory that contains the `osagent.exe` file. This is the `bin` directory of your Borland Enterprise Server installation.

5 The Library For Projects field defines the library location of the ORB tools. To define the library location, click the ellipsis (…) button next to the Library For Projects field. This displays the Select A Different Library dialog box.



The library is necessary for compiling the generated stubs and skeletons and for executing an application. For VisiBroker, browse to the Borland Enterprise Server 6.0 Client library, then click OK two times.

6 In the Enterprise Setup dialog box, leave the Apply This Configuration To The Current Project option selected in order to set this ORB configuration for the current project, as well as the default project.

7 Set the Make This Configuration's ORB The Default For The Java VM option to use the selected ORB as the default ORB for the Java VM.

8 Select the Add A VisiBroker Smart Agent Item To The Tools Menu option to enable the Smart Agent to run as a JBuilder service during development. This option will be disabled if the Smart Agent is not necessary.

9 Select a port on which to run the Smart Agent. The default port of 14000 will work on most systems.

Note When running the project, you can use a JBuilder macro in the VM Parameters field on the Edit Runtime Configuration dialog box for the application. This macro allows the project to be used on another machine where the Smart Agent is configured for a different port. The syntax is:

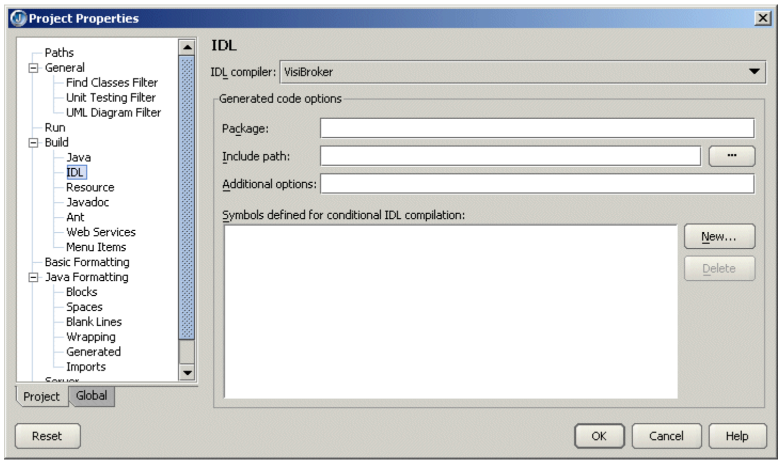
```
-Dvbroker.agent.port=($SmartAgentPort)
```

10 Click OK to save the configuration.

You also need to pass options to the IDL compiler.

To pass options to the IDL compiler when configuring VisiBroker stand-alone,

1 Choose Project|Project Properties|Build|IDL to display the IDL Build page, which will look similar to this:



2 Make sure VisiBroker is selected as your IDL compiler.

3 Enter the following options into the Additional Options field:

```
-VBJprop borland enterprise.licenseDir=<visibroker license directory>
-VBJjavavm<jdk1.4.1 java or javaw path>
```

The license directory is `<BES_install>\var\servers\SERVER_NAME\adm`. The `<server directory>` is same as the server name assigned when you configured the server using Enterprise|Configure Servers. You can find the server name on the Custom page of the Configure Servers dialog box.

4 Click OK to close the Project Properties dialog box.

You must still perform one step: starting the VisiBroker Smart Agent. This handles the initial bootstrap issues such as how the client locates the naming service and so on. To start the Smart Agent, choose Tools|VisiBroker Smart Agent.

You've now completed setting up your system to use the JBuilder CORBA features.

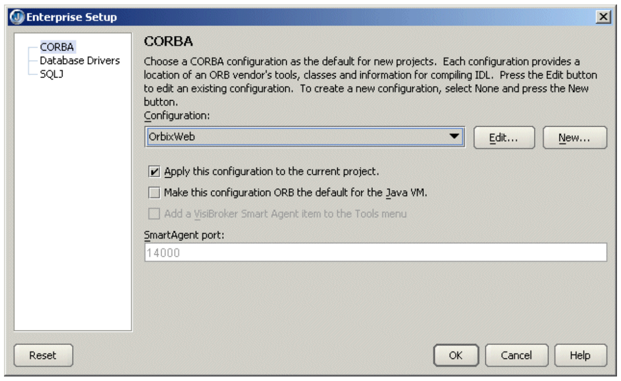
Configuring the OrbixWeb ORB

Follow the steps in this section to set up JBuilder for use with the OrbixWeb ORB.

To configure the OrbixWeb ORB,

1 Choose Enterprise|Enterprise Setup to display the Enterprise Setup dialog box. Select the CORBA page. The parameters in this dialog box allow JBuilder to see the ORB.

2 Select the OrbixWeb option from the Configuration drop-down list.



CORBA

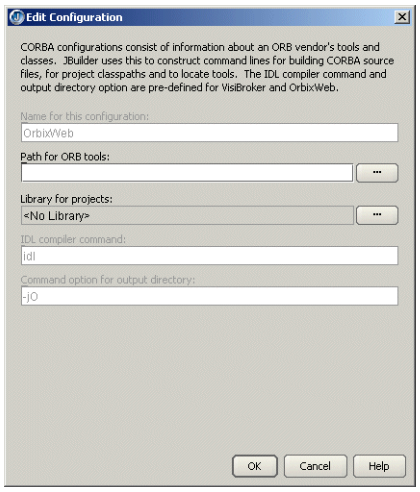
Choose a CORBA configuration as the default for new projects. Each configuration provides a location of an ORB vendor's tools, classes and information for compiling IDL. Press the Edit button to edit an existing configuration. To create a new configuration, select None and press the New button.

Configuration: OrbixWeb

[✓] Apply this configuration to the current project.
[ ] Make this configuration ORB the default for the Java VM.
[ ] Add a VisiBroker Smart Agent item to the Tools menu

SmartAgent port: 14000

3 Click the Edit button to display the Edit Configuration dialog box.



CORBA configurations consist of information about an ORB vendor's tools and classes. JBuilder uses this to construct command lines for building CORBA source files, for project classpaths and to locate tools. The IDL compiler command and output directory option are pre-defined for VisiBroker and OrbixWeb.

Name for this configuration: OrbixWeb

Path for ORB tools:

Library for projects: <No Library>

IDL compiler command: idl

Command option for output directory: -jO

4 The Path For ORB Tools field allows JBuilder to access ORB tools. Click the ellipsis (…) button next to the Path For ORB Tools field to browse to the directory containing the ORB tools.

5 The Library For Projects field defines the library location of the ORB tools. To define the library location, click the ellipsis (…) button next to the Library For Projects field. This displays the Select A Different Library dialog box. The library is necessary for compiling the generated stubs and skeletons and for executing an application. Choose the library and click OK two times.

6 In the Enterprise Setup dialog box, leave the Apply This Configuration To The Current Project option selected if this ORB configuration should be applied to the current project as well as the default project.

7 Set the Make This Configuration's ORB The Default For The Java VM option to use the selected ORB as the default ORB for the Java VM.

8 Click OK to save the configuration.

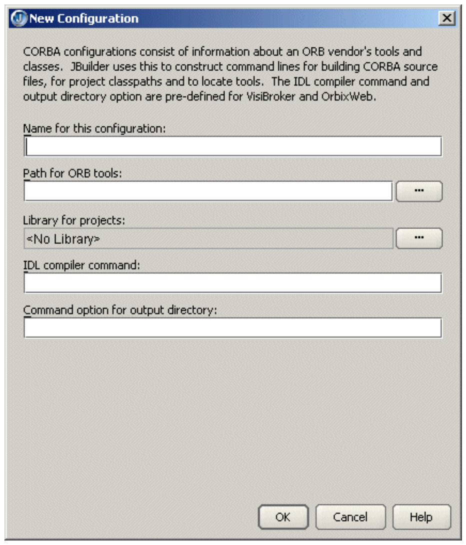
Adding a new configuration

Follow the steps in this section to create a new configuration for a third-party ORB.

To configure a third-party ORB,

1 Choose Enterprise|Enterprise Setup to display the Enterprise Setup dialog box. Select the CORBA page. The parameters in this dialog box allow JBuilder to see the ORB.

2 Select the None option from the Configuration drop-down list, then click the New button. The New Configuration dialog box is displayed.


New Configuration

CORBA configurations consist of information about an ORB vendor's tools and classes. JBuilder uses this to construct command lines for building CORBA source files, for project classpaths and to locate tools. The IDL compiler command and output directory option are pre-defined for VisiBroker and OrbixWeb.

Name for this configuration:

Path for ORB tools:

Library for projects: <No Library>

IDL compiler command:

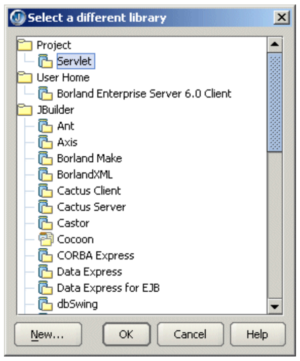
Command option for output directory:

3 Enter a name for the configuration in the Name For This Configuration field.

4 Enter the path for the ORB tools in the Path For ORB Tools field. Click the ellipsis (…) button to browse to the directory containing the ORB tools.

5 Click the ellipsis (…) button next to the Library For Projects field to display the Select A Different Library dialog box. You use this dialog box to browse to the library

location of the ORB tools. The library is necessary for compiling the generated stubs and skeletons and for executing an application.



a To add a library that is not displayed in the Select A Different Library dialog box, click the New button.

b In the New Library wizard, enter a name for the new library in the Name field.

c Select a location for the library: the JBuilder directory, the project directory, or your home directory.

d Click Add to browse to the library (JAR) file.

e Click OK until you return to the New Configuration dialog.

6 Enter the command for the IDL compiler in the IDL Compiler Command field.

7 Enter the IDL compiler command option that specifies the output directory in the Command Option For Output Directory field.

8 Click OK to close the New Configuration dialog box.

9 In the Enterprise Setup dialog box, leave the Apply This Configuration To The Current Project option selected in order to set this ORB configuration for the current project, as well as the default project.

10 Click OK to save the new configuration.

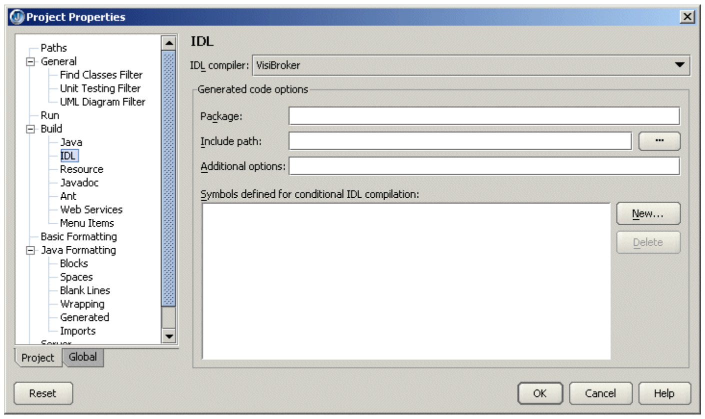
Setting and viewing ORB build properties

This section explains how to set and view build properties for the installed ORB.

Setting VisiBroker ORB properties

To set and view properties for the `java2iiop` and `java2idl` compilers in the JBuilder development environment,

1 Choose Project|Project Properties|Build|IDL. The IDL page is displayed.



IDL compiler: VisiBroker

Make sure the IDL Compiler field is set to VisiBroker or VisiBroker (Borland Enterprise Server AppServer Edition 6.0 and 5.2.1).

2 Set the Generated Code Options as needed:

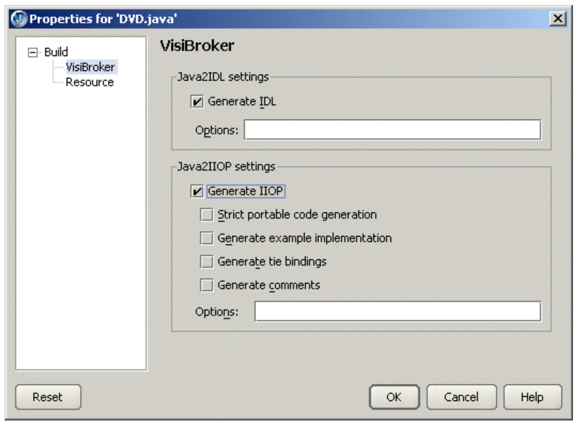
- In the Package field, enter the name of the package to generate code in. The package name for definitions is prepended with the specified package name. If a directory with the specified package name does not exist, it will be created. If the package directory exists, its contents will be updated. Code is generated that uses CORBA package resolution rules.
- In the Include Path field, enter the location where other files that are referenced from IDL files reside. Click the ellipsis (…) button beside this field to open a dialog box that lets you browse the directory structure for the location of the files.
- In the Additional Options field, enter any additional options for your IDL compiler, as you would enter them when running the compiler from the command line.
- The Symbols Defined For Conditional IDL Compilation list displays symbols that have been defined for conditional IDL compilation. Click the New button to display the Define New Symbol dialog box where you can define a new conditional symbol. To remove a symbol from the list, select it then choose the Delete button.

3 Choose the Paths node of the Project Properties dialog box. Make sure the Required Libraries list includes the Borland Enterprise Server 6.0 Client.

4 Click OK to close the Project Properties dialog box.

5 Once you've created a Java interface file in your JBuilder project, right-click the file in the project pane and choose Properties.

6 On the Build|VisiBroker node, select the Generate IDL and Generate IIOP options. The dialog box should look like this:



VisiBroker

Java2IDL settings
[✓] Generate IDL
Options:

Java2IIOP settings
[✓] Generate IIOP
[ ] Strict portable code generation
[ ] Generate example implementation
[ ] Generate tie bindings
[ ] Generate comments
Options:

- Select the Java2IDL Generate IDL option to generate an Interface Definition Language (IDL) file from a Java file or a class file. To support the distribution of objects implemented in a variety of programming languages, an IDL file is used to define the services offered by a particular distributed object.
- Enter any Java2IDL command line options in the Options field. See the Borland Enterprise Server VisiBroker documentation for command line options.
- The Java2IIOP Generate IIOP Interface option generates an Internet InterORB (Object Request Broker) Protocol (IIOP) compatible Java interface file from a Java interface or class file. This is done by creating a new Java interface that extends `org.omg.Corba.object`. This Java interface can be used to describe CORBA interfaces in place of using an Interface Definition Language (IDL) file to describe the interfaces. The IIOP protocol maintains a basic set of functionality to ensure inter-operability between client applications and server-based objects in a Common Object Request Broker Architecture (CORBA).

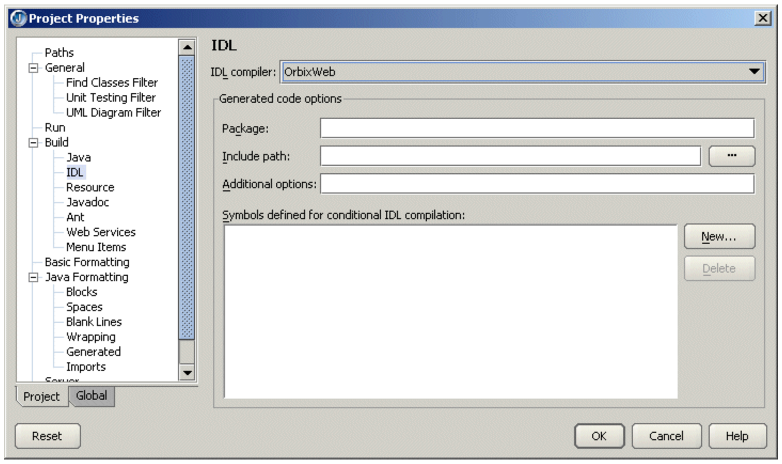
- Set the Java2IIOP Strict Portable Code Generation (Java2IIOP) option to enable generation of portable stubs, that is, stubs that contain code not specific to VisiBroker.
- Set the Java2IIOP Generate Example Implementation option to enable generation of example implementation code.
- Set the Java2IIOP Generate Tie Bindings option to enable generation of tie classes.
- Set the Java2IIOP Generate Comments option to enable generation of comments in the source code.
- Enter any Java2IIOP command line options in the Options field. See the Borland Enterprise Server VisiBroker documentation for command line options.

7 Click OK to close the dialog box.

Setting build properties for the OrbixWeb or other third-party ORBs

To set and view properties for a configured ORB,

1 Choose Project|Project Properties|Build|IDL. The IDL page is displayed.



IDL compiler: OrbixWeb

Make sure the IDL Compiler field matches the ORB you selected in the Enterprise Setup dialog box.

2 Set the Generated Code Options as needed:

- In the Package field, enter the name of the package to generate code in. The package name for definitions is prepended with the specified package name. If a directory with the specified package name does not exist, it will be created. If the package directory exists, its contents will be updated. code is generated that uses CORBA package resolution rules.
- In the Include Path field, enter the location where other files that are referenced from IDL files reside. Click the ellipsis (…) button beside this field to open a dialog box that lets you browse the directory structure for the location of the files.
- In the Additional Options field, enter any additional options for your IDL compiler, as you would enter them when running the compiler from the command line.
- The Symbols Defined For Conditional IDL Compilation list displays symbols that have been defined for conditional IDL compilation. Click the New button to display the Define New Symbol dialog box where you can define a new conditional symbol. To remove a symbol from the list, select it then choose the Delete button.

3 Choose the Paths node in the Project Properties dialog box. Make sure the Required Libraries list includes the libraries specific to the ORB you're working with.

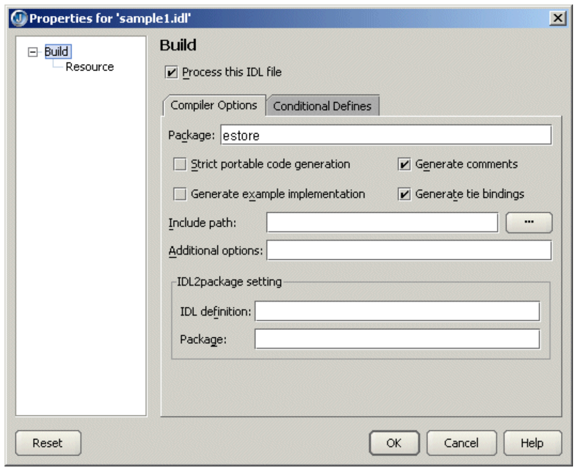
4 Click OK to close the Project Properties dialog box.

Setting and viewing IDL build properties

This section explains how to set and view build properties for an IDL file. You must first create an IDL file. You can use the Sample IDL wizard, available on the Enterprisel CORBA page of the object gallery (FilelNew).

To set and view IDL build properties,

1 Right-click an IDL file in the project pane and choose Properties. The Compiler Options page of the Build node (Properties dialog box) is displayed:



[✓] Process this IDL file

Package: estore

[ ] Strict portable code generation
[✓] Generate comments
[ ] Generate example implementation
[✓] Generate tie bindings

Include path:

Additional options:

IDL2package setting

IDL definition:

Package:

- Set the Process This IDL File option to set compiler options, add the compiled files and/or packages to the current project, and view compiler output.
- Enter the package for generated code in the Package field. The package name for definitions is prepended with the specified package name. If a directory with the specified package name does not exist, it will be created. If the package directory exists, its contents will be updated. When this option is not set, `idl2java` generates code using the CORBA package resolution rules.
- Select the Strict Portable Code Generation option to generate code that can run on any CORBA-compliant ORB. This means that any VisiBroker-specific code will not be generated, including code that generates smart stubs. Selecting this option also suppresses the generation of `toString()` and `bind()` methods. When this option is selected, the compiler generates skeleton code using the Dynamic Skeleton Interface (DSI).

  If not selected, the portable stubs will not be generated. VisiBroker-specific extensions and optimizations will be present in the code that is generated. Generated stubs will specifically be for use in VisiBroker ORB environments.
- Select the Generate Example Implementation option to generate example classes when the IDL file is compiled. If not selected, the generation of example classes will be suppressed.
- Select the Generate Comments option to generate comments in the code. If not selected, comments will be suppressed in the generated code.
- Select the Generate Tie Bindings option to generate `_tie` classes when the IDL file is compiled. If this option is not selected, the generation of `_tie` classes will be suppressed.

The tie mechanism offers an alternative when it is not convenient or possible to have your implementation class inherit from the VisiBroker skeleton class. Object implementation classes must inherit from the VisiBroker skeleton class. Java does not allow multiple class inheritance and you may want your implementation class to inherit from a different class.

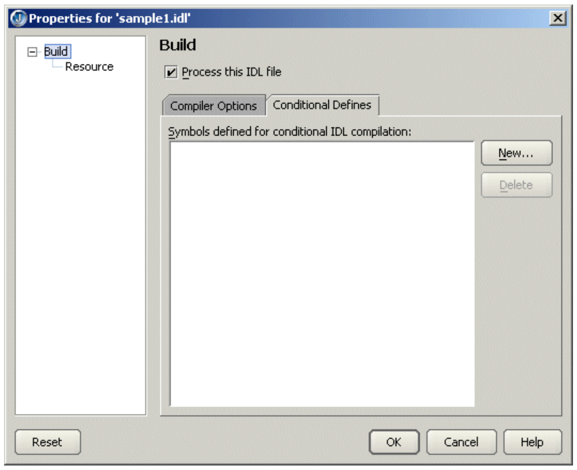
Inheritance is easier to use than the tie mechanism because implementation objects look and behave just like object references. If an implementation object happens to be in the same process as a client using it, operation invocations involve less overhead because no transport, indirection, or delegation of any kind is required.

The tie mechanism provides a `delegator implementation` class that inherits from `org.omg.CORBA.Object`. The delegator implementation class does not provide any semantics of its own. It simply delegates every call to the real implementation class, which can be implemented separately and can inherit from whichever class it wishes.

The `Operations` class defines all of the methods that must be implemented by the object implementation. This class acts as the delegate object for the associated `_tie` class when the tie mechanism is used.

- Click the ellipsis (…) button next to the Include Path field to display the Select Directory dialog box, where you can select a directory to search for the list of files to include in the build. You must select a directory that contains an `.idl` file.
- Enter options not available in this dialog box in the Additional Options field.
- The IDL2package Setting options are the equivalent of the command line parameter `-idl2package idl pkg`. If any part of the IDL file is in the scope `::CORBA,`the generated is code is placed in the `org.omg.CORBA` package. When this option is not set, the generated code uses the CORBA package resolution rules.
  - IDL Definition — The definition in scope of IDL.
  - Package — The Java package into which to place the IDL definition.

2 Click the Conditional Defines tab to set symbols defined for conditional compilation.



- The Symbols Defined for Conditional IDL Compilation list displays a list of symbols defined for conditional IDL compilation.
- Click the New button to display the Define New Symbol dialog box, where you to define a symbol name for conditional IDL compilation, for example `#define name def`.
- Choose a symbol and click the Delete button to delete the selected symbol from the list of symbols defined for conditional compilation.

3 Click the OK button to set the options to be used by the VisiBroker `idl2java` compiler, when the IDL file is compiled. Set the correct options so that when compiled, Java interface definitions and Java client and server stubs and skeletons are generated in a subdirectory of the project with the same name as the project.

When the Process This IDL File option is not checked, selecting OK saves the selected options, but does not call the `idl2java` compiler when the IDL file is compiled. This means that the client stubs and server skeletons are not generated when the IDL file is compiled.

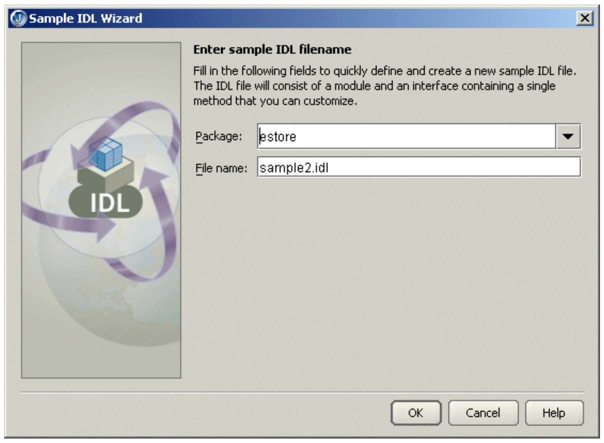
Using the CORBA wizards

This section describes the JBuilder wizards that are available for CORBA development. Wizards provide quick and easy development of IDL files and CORBA client and server interfaces. These wizards are available only if your project is configured for use with CORBA (Enterprise|Enterprise Setup|CORBA).

Sample IDL wizard

The Sample IDL wizard creates a simple IDL file that may be used to experiment with the various CORBA tools that take an IDL file as input. The generated IDL file will consist of a module and an interface containing a single method that can be customized. This wizard is available on the Enterprise|CORBA page of the object gallery (File|New).

The Sample IDL wizard looks like this:



Enter sample IDL filename

Fill in the following fields to quickly define and create a new sample IDL file. The IDL file will consist of a module and an interface containing a single method that you can customize.

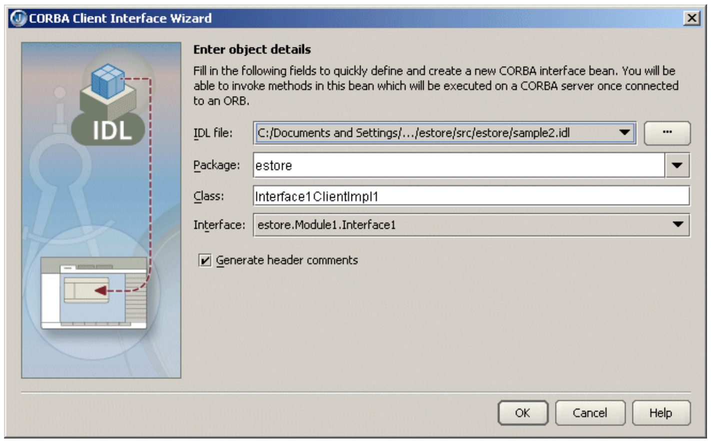
Package: estore

File name: sample2.idl

CORBA Client Interface wizard

The CORBA Client Interface wizard produces a layer over `idl2java`-generated code allowing a client application to easily invoke the interface methods of a running CORBA server. You will be able to invoke methods in the interface that will be executed on a CORBA server when you are connected to an Object Request Broker (ORB). This wizard is only available if you have an IDL file in your project. It is available on the Enterprise|CORBA page of the object gallery (File|New).

The CORBA Client Interface wizard looks like this:



Enter object details

Fill in the following fields to quickly define and create a new CORBA interface bean. You will be able to invoke methods in this bean which will be executed on a CORBA server once connected to an ORB.

IDL file: C:/Documents and Settings/.../estore/src/estore/sample2.idl

Package: estore

Class: Interface1ClientImpl1

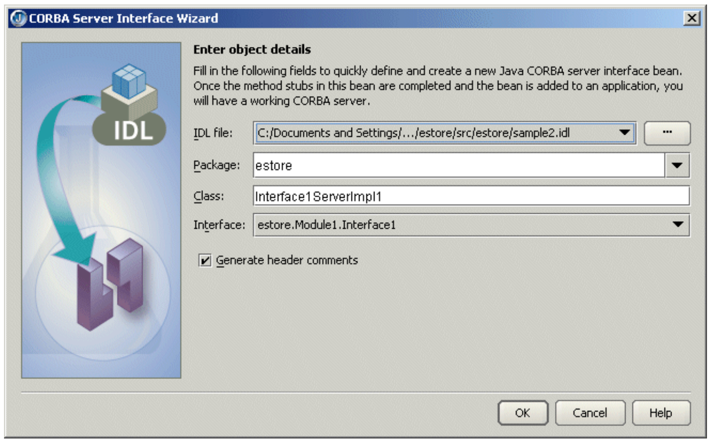
Interface: estore.Module1.Interface1

[✓] Generate header comments

CORBA Server Interface wizard

The CORBA Server Interface wizard lets you define and create a new Java CORBA server interface bean. Complete the method stubs and add the bean to an application to create a working CORBA server for that interface. This wizard is only available if you have an IDL file in your project. It is available on the Enterprise|CORBA page of the object gallery (File|New).

The CORBA Server Interface wizard looks like this:



Enter object details

Fill in the following fields to quickly define and create a new Java CORBA server interface bean. Once the method stubs in this bean are completed and the bean is added to an application, you will have a working CORBA server.

IDL file: C:/Documents and Settings/.../estore/src/estore/sample2.idl

Package: estore

Class: Interface1ServerImpl1

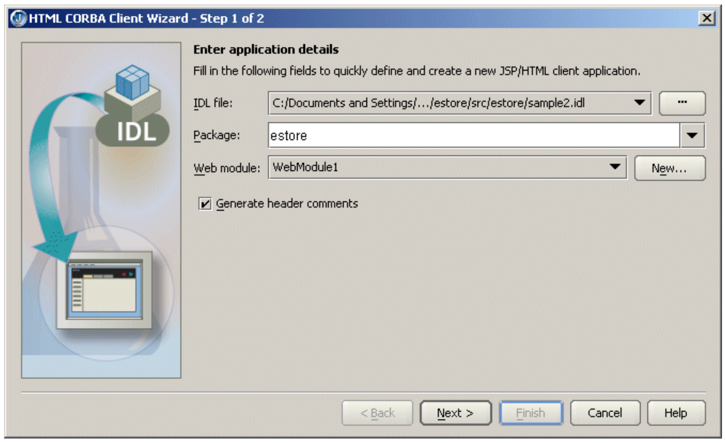
Interface: estore.Module1.Interface1

[✓] Generate header comments

HTML CORBA Client wizard

The HTML CORBA Client wizard produces a JSP or HTML client application that provides input to a CORBA server. This wizard is only available if you have an IDL file in your project and a web server is selected. It is available on the Enterprise|CORBA page of the object gallery (File|New).

The Enter Application Details page of the HTML CORBA Client wizard looks like this. This page of the wizard allows you to select an existing web module or create a new one.


HTML CORBA Client Wizard - Step 1 of 2

Enter application details

Fill in the following fields to quickly define and create a new JSP/HTML client application.

IDL file: C:/Documents and Settings/.../estore/src/estore/sample2.idl

Package: estore

Web module: WebModule1

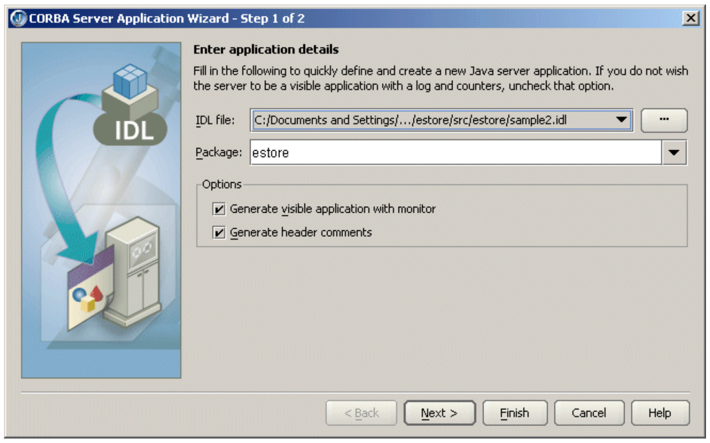
☑ Generate header comments

The final page of the wizard is where you create a runtime configuration.

CORBA Server Application wizard

The CORBA Server Application wizard produces an application that provides a complete default implementation for a CORBA server. This wizard is only available if you have an IDL file in your project. It is available on the Enterprise|CORBA page of the object gallery (File|New).

The Enter Application Details page of the CORBA Server Application wizard looks like this:


CORBA Server Application Wizard - Step 1 of 2

Enter application details

Fill in the following to quickly define and create a new Java server application. If you do not wish the server to be a visible application with a log and counters, uncheck that option.

IDL file: C:/Documents and Settings/.../estore/src/estore/sample2.idl

Package: estore

Options

☑ Generate visible application with monitor

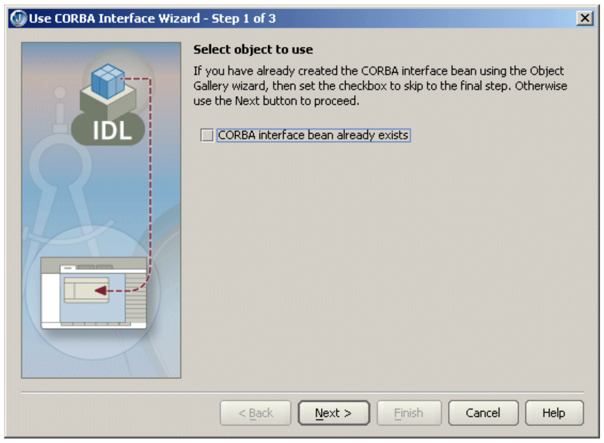
☑ Generate header comments

The final page of the wizard is where you create a runtime configuration.

Use CORBA Interface wizard

This Use CORBA Interface wizard creates a client CORBA application. It will create a reference to a CORBA client interface that was built earlier, or it will create one. This wizard is only available if you have an IDL file in your project and a `.java` file open in the editor. It is available from the Edit menu.

The first page of the Use CORBA Interface wizard, where you choose whether to use an existing client interface, looks like this:

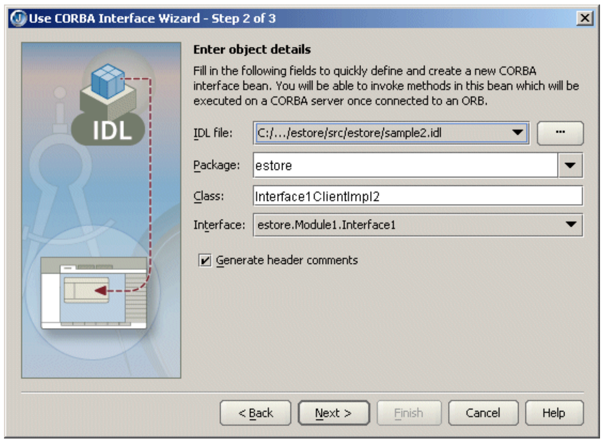


Select object to use

If you have already created the CORBA interface bean using the Object Gallery wizard, then set the checkbox to skip to the final step. Otherwise use the Next button to proceed.

☐ CORBA interface bean already exists

The second page of the wizard, where you define and create the new CORBA interface bean looks like this:



Enter object details

Fill in the following fields to quickly define and create a new CORBA interface bean. You will be able to invoke methods in this bean which will be executed on a CORBA server once connected to an ORB.

IDL file: C:/.../estore/src/estore/sample2.idl

Package: estore

Class: Interface1ClientImpl2

Interface: estore.Module1.Interface1

☑ Generate header comments

The final page of the wizard is where you create a runtime configuration.

Chapter

7

Using JBuilder with BEA WebLogic servers

This chapter explains how to use to set up and use BEA WebLogic servers with JBuilder. JBuilder supports WebLogic Server 7.x and WebLogic Platform 8.1.

Service packs

JBuilder requires WebLogic 7.0 with Service Pack 5 for proper operation. For WebLogic 8.1, you will need Service Pack 3.

Configuring JBuilder for WebLogic servers

To configure JBuilder settings to target WebLogic servers,

1. Choose Enterprise|Configure Servers.

   The Configure Servers dialog box appears.

2. Select the WebLogic server you want to configure by clicking it in the pane on the left.

   The right side of the dialog box lists the default settings for the selected server. The General page has fields that all WebLogic servers have in common, while the Custom page has fields that are specific to the selected server. In some cases, modifying a Custom setting will update a setting on the General page.

3. Check the Enable Server check box at the top of the dialog box.

   Checking this option enables the fields for the selected application server. You won't be able to edit any fields until it is checked. The Enable Server check box also determines whether this server will appear in the list of servers when you select a server for your project using the Servers page of the Project|Project Properties dialog box.

   JBuilder provides default settings for the selected server. If JBuilder's default settings are not appropriate, edit any of the settings as needed. Some servers require you to enter settings in addition to the defaults. Clicking the OK button in this dialog box when not all required values are set or selecting another server while the current one is enabled displays a message dialog box that informs you about the missing settings.

4 If you want to change any of the settings, click the ellipsis (…) button next to the field and make your changes. Assuming you installed WebLogic Platform 8.1 into a BEA_HOME directory with the name `bea`, your home directory should be `[drive]:/bea/weblogic81/server`. For WebLogic Server 7.x, you home directory should be `[drive]:/bea/weblogic71/server`.

   For WebLogic Platform 8.1 and WebLogic Server 7.x, the working directory is set when you set the domain directory using the Custom page. For now, you can leave it unchanged.

5 Click the Custom tab to view fields unique to the server. Change or fill in these fields:

   - BEA Home Directory: The BEA Home Directory is the directory into which you installed the WebLogic Server and that contains the file `registry.xml`. JBuilder assigns a default value for the BEA Home Directory that varies depending on the version of the WebLogic Server installed.
   - JDK Installation Directory: For WebLogic Platform 8.1, the directory is `<bea home>/jdk141_04`. For WebLogic 7.x, the directory is `<bea home>/jdk131_010`.
   - Domain Directory: Use the ellipsis (…) button to select a directory as your domain directory. The domain directory is the directory where your WebLogic domain is stored. For WebLogic 7.x, this is `user_projects/mydomain`. For WebLogic 8.1, it is `user_projects/domains/mydomain`. Setting the domain directory sets the working directory value on the General page. For more information about WebLogic domains, see your WebLogic documentation.
   - User Name: The user name you use to authenticate yourself to the server. This field appears for WebLogic Platform 8.1 and WebLogic Server 7.x.
   - Password: The password you use to authenticate yourself to the server.
   - Server Name: The name you specify is used as a VM parameter for starting the server. JBuilder suggests a default name of “myserver.”
   - Listen Address: Specifies a listen address for this server. The host value must be either the DNS name or the IP address of the server. If you do not specify a Listen Address, the server uses either the machine’s DNS name or its IP address.
   - Listen Port: Specifies the non-SSL listen port for this server. If you do not specify a Listen Port, the server uses 7001 as the default.
   - Version: Select the combination of WebLogic version and service pack you are configuring from the drop-down list.

     The Custom page of the Enterprise|Configure Servers dialog box for WebLogic 7.x and 8.x now includes Listen Address and Listen Port fields. If you edit fields such as User Name, Server Name, Listen Address, and so on, the corresponding entries in the Server Deployment dialog box and the node deployment dialog box are updated accordingly.
   - Add An Admin Console Item To The Enterprise Menu: Checking this check box adds the WebLogic Admin Console item to JBuilder’s Enterprise menu. You must also fill in the next field, Web Browser Path, to have the menu item added.

- Web Browser Path: Specify the path and file name of the web browser of your choice. For example, you might specify `c:/program files/Internet Explorer/iexplore.exe` if you want to use Microsoft's Internet Explorer.
- Add A Configuration Wizard Item To The Enterprise Menu: Adds a WebLogic Configuration Wizard item to the JBuilder Enterprise menu. When you select this menu item, the BEA WebLogic Configuration wizard begins, which guides you through the steps of configuring the domain for your use.
- Use External Compiler: Check this check box if you want to use an external compiler to generate stub files instead of JBuilder's compiler, bcj. WebLogic Platform 8.1 uses the appc compiler, while earlier versions use the ejbc compiler.
- Path: Specify the fully-qualified name of the compiler you want to use. You can use the ellipsis (…) button to navigate to the compiler location.

6 Choose OK to close the dialog box.

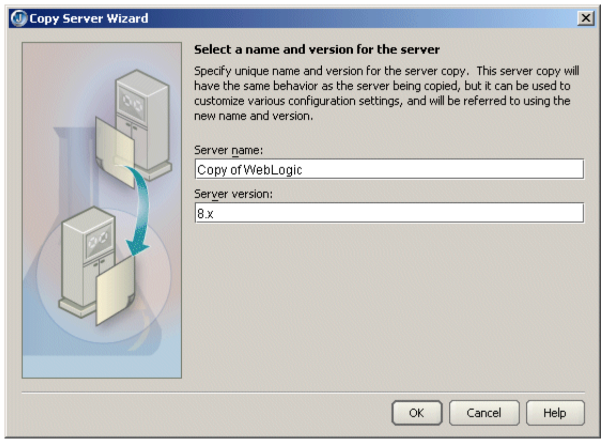
Creating a duplicate configuration to edit

Once you've created a server configuration, you might find you'd like to have a similar one, but with some slight differences. Follow these steps to create a duplicate configuration you can then edit:

1 Choose Enterprise|Configure Servers to display the Configure Servers dialog box.

2 Select the server configuration you would like to duplicate from the left pane.

3 Click the Copy button at the bottom of the left pane to display the Copy Server wizard:



Select a name and version for the server

Specify unique name and version for the server copy. This server copy will have the same behavior as the server being copied, but it can be used to customize various configuration settings, and will be referred to using the new name and version.

Server name: Copy of WebLogic

Server version: 8.x

4 Specify the name you want to use to identify this server configuration in the Name For This Server field.

5 Specify the server version in the Version For This Server field.

6 Click OK.

7 Select your new configuration in the left pane of the Configure Servers dialog box and make the changes to the settings you need using the General and Custom pages.

Copying a server configuration also creates copies of that server's tools. You could, therefore, create a duplicate server configuration and modify just the server's tools as the only difference between the two configurations.

Created libraries

When you configure JBuilder to target a WebLogic server, required libraries are created for you that contain all the application server files you will need for enterprise bean development. These are the libraries created for you, listed by WebLogic server version:

WebLogic Platform 8.1

- WebLogic 8.1 Client: All JARs needed to run a client.
- WebLogic 8.1 Deploy: JARs needed to run the WebLogic 8.1 deploy tool.

WebLogic Server 7.x

- WebLogic 7.x Client: All JARs needed to run a client.
- WebLogic 7.x Deploy: JARs needed to run the WebLogic 7.x deploy tool.

Adding a service pack

If you want to add a service pack to your application server configuration, follow these steps:

1 Choose Enterprise|Configure Servers.

2 Select your server from the list of servers in the pane on the left side of the dialog box.

3 Click the General tab, then the Class tab, if they are not already selected.

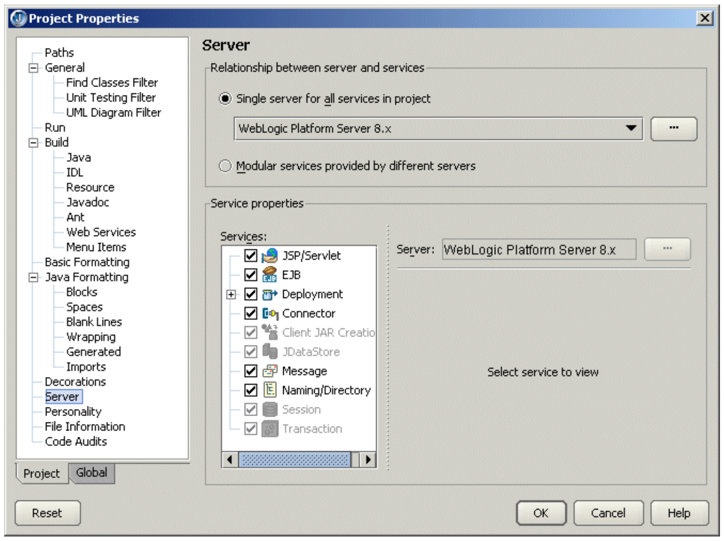
4 Click Add.

5 Navigate to the location of the service pack JAR file.

6 Click OK twice to close all dialog boxes.

Selecting a server

If you are working with more than one version of WebLogic Server and have configured JBuilder for all versions, you must select which version you want to use for your current project. Choose Project|Project Properties and select Server in the tree to display the Server page.


Project Properties

Server

Relationship between server and services

● Single server for all services in project

WebLogic Platform Server 8.x

○ Modular services provided by different servers

Service properties

Services:

- ✓ JSP/Servlet
- ✓ EJB
- ✓ Deployment
- ✓ Connector
- ✓ Client JAR Creatio
- ✓ JDataStore
- ✓ Message
- ✓ Naming/Directory
- ✓ Session
- ✓ Transaction

Server: WebLogic Platform Server 8.x

Select service to view

You can click the Help button on this page for assistance using this page to select a server.

JBuilder can target multiple servers. You can choose a single application server for all stages of EJB and web application development, or you can choose different servers for different aspects of development. For example, you can select one server to use to develop enterprise beans and another to develop web applications.

Decide whether you want to use a single server for all aspects of development or different servers to handle different areas of development.

- To use a single server,
  - **a** Select the Single Server For All Services In Project option and select the server from the drop-down list.
  - **b** If you want to make changes to the configuration settings for the server, click the ellipsis (…) button and edit the settings you want on the General and Custom pages. Click OK.

    You can also use this dialog box to select a different server.
- To use different servers for different services,
  - **a** Select the Modular Services Provided By Different Servers option.
  - **b** Check the check boxes next to all the services for which you want to specify an application server in the Services list.
  - **c** Click one of the checked services to select it in the Services list.

    Note that the web module won't be deployed automatically if it is part of an EAR.
  - **d** In the Service Properties For Project group box, use the Server drop-down list to select the server you want to use for this service.

    If server-specific properties are available for the server you selected for the particular service you are working with, they will appear below the Server drop-down list. If the Deployment service is selected, a Build Target For Deploying To Server drop-down list appears on the Service Properties For Project panel. Select the type of build you want to occur. If the EJB service is selected, you'll see read-only information appear that is intended to help you decide which server is appropriate for your needs.

    If you select the JSP/Servlet service, the Default Runtime Host Name and Default Runtime Port Number fields are available to you. The default values refer to the global server configuration. If you change these values, they are used for all server run configurations created after you made the change.

    If you expand the Deployment service node, you can select from several subnodes that have accompanying properties. The properties vary depending on which is your selected server. All servers have a Build Target For Deploying To Server field, however. Select your desired build option from the drop-down list.

    If you expand the WebLogic Deployment service node and select the WARs node, the service properties include a Map Project Webapps At Runtime option. When this option is selected (the default value) all web applications in the project deploy from the web application directory and not as a WAR when the server starts up.

    If you expand the WebLogic Deployment service node and select the EARs node, the service properties include a Map Project EJB Modules At Runtime option. When this option is selected (the default value) all EJBs in the project deploy from the EJB directory and not as a EAR when the server starts up.

    All the WebLogic Deployment service nodes have Deploy As Archive and Deploy As Exploded Directory options. Checking the Deploy As Archive option enables the archive to be deployed. Checking the Deploy As Exploded Directory option means the archive is exploded into a directory as its contents are deployed instead.

If you select the EJB service, the service properties include a Datasource Mapping combo box with options to deploy (auto-deploy) data sources for all EJB modules in the project. These are the options:

- Map Project Datasources: Maps data sources for all EJB modules in `config.xml` for the domain specified in the server configuration. Datasource entries are removed when the server shuts down.
- Map And Persist Project Datasources: Maps data sources for all EJB modules in `config.xml` for the domain specified in the server configuration. Data Source information is persisted in `config.xml`. Datasource entries are removed when the server shuts down.
- Do Not Map Project Datasources: Does not map any data sources.

e If you want to make changes to the configuration settings for the server selected for the service, click the ellipsis (…) button and edit the settings you want on the General and Custom pages. Click OK.

f Repeat these steps for each service you want access to.

Click OK when you are finished with the dialog box to select your specified server for your current project.

If you neglect to select one ore more servers for a project, the Server page of the Project Properties dialog box appears when you start an EJB wizard from the object gallery. Use this page to select your server. All EJB projects must have one or more selected servers.

Working in JBuilder

JBuilder supports WebLogic servers in all areas of development. This section describes some of the WebLogic-specific features available to you.

Using existing code

If you already have existing WebLogic EJB 2.0 deployment descriptors for enterprise beans you created previously, you can create a new EJB module that contains the descriptors. You have two options:

- Use the EJB Module From Descriptors wizard, which creates a new EJB module that contains the deployment descriptors.
- Use the Project For Existing Code wizard, which creates a new JBuilder project containing one or more new EJB modules that contain the deployment descriptors.

See “Creating an EJB module from existing deployment descriptors” in *Developing Applications with Enterprise JavaBeans* for complete information.

Creating WebLogic entity beans in JBuilder

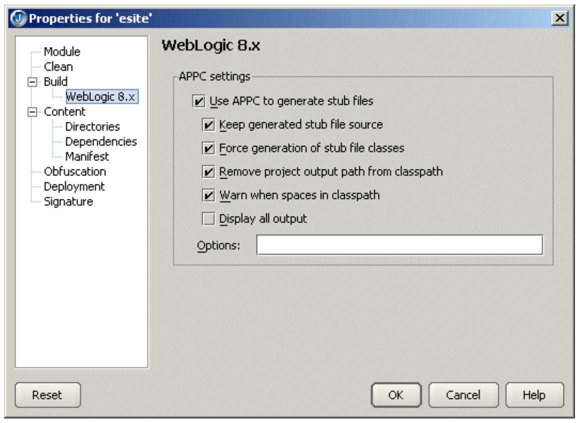
JBuilder's EJB designer has support for WebLogic-specific features. The entity bean inspector includes options such as optimistic concurrency and field groups. There is a WebLogic version of the Table Map Editor. When creating relationships between entity beans, there are WebLogic versions of the relationship inspector and a special WebLogic RDBMS Relation Editor. For information about using these tools, see "Adding WebLogic table references" and "Specifying a WebLogic 7.x – 8.x relationship" in *Developing Applications with Enterprise JavaBeans*.

Using the DD Editor

The DD Editor has several WebLogic-specific pages for information that applies only to WebLogic servers. For information about using the DD Editor to edit J2EE modules, see Chapter 11, "Editing J2EE deployment descriptors." Whenever a WebLogic-specific page is displayed in the DD Editor, you can press *F1* for help on that particular page. You'll also find it convenient to have a copy of your server's documentation handy to look up the type of information your server requires in the deployment descriptors.

Compiling

To set APPC or EJBC options for compiling your WebLogic beans, right-click the EJB module node in the project pane, choose Properties and select the Build|WebLogic page. This page contains the options for your WebLogic compiler. Here, for example, are the APPC options for WebLogic Platform 8.1:



WebLogic 8.x

APPC settings

- [✓] Use APPC to generate stub files
- [✓] Keep generated stub file source
- [✓] Force generation of stub file classes
- [✓] Remove project output path from classpath
- [✓] Warn when spaces in classpath
- [ ] Display all output

Options:

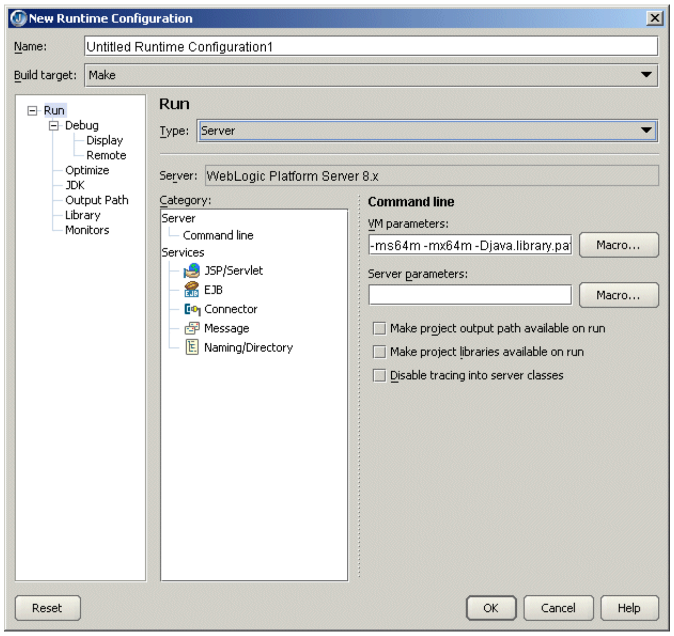
Starting the server

To prepare to start the server from within JBuilder, create a server runtime configuration. Follow these steps:

1 Choose Run|Configurations.

2 In the dialog box that appears, click the New button.

3 Select Server from the Type drop-down list.


New Runtime Configuration

Name: Untitled Runtime Configuration1

Build target: Make

Run
- Debug
  - Display
  - Remote
- Optimize
- JDK
- Output Path
- Library
- Monitors

Run

Type: Server

Server: WebLogic Platform Server 8.x

Category:
- Server
  - Command line
- Services
  - JSP/Servlet
  - EJB
  - Connector
  - Message
  - Naming/Directory

Command line

VM parameters: -ms64m -mx64m -Djava.library.pat

Server parameters:

- [ ] Make project output path available on run
- [ ] Make project libraries available on run
- [ ] Disable tracing into server classes

4 In the Name field, enter a name, such as Server or something that will identify the run configuration.

5 Fill in the VM Parameters and Server Parameters needed to run the server. If you've selected a target application server as described in "Selecting a server" on page 64, default values are already in place. If the default values aren't adequate, you may find the Macro buttons helpful in building the command line you need.

6 Click OK two times.

To start the server, click the down-arrow next to the Run Project icon (▶) and select the server run configuration you just created.

Remote deploying

To deploy a deployable module to a remote server, follow these instructions for your version of WebLogic Server:

WebLogic Application Sever 7.x and WebLogic Platform Server 8.1

Choose one of these options:

- Using the Enterprise|Server Deployment command:

  a Choose Enterprise|Server Deployment to display the WebLogic Deploy Settings dialog box.

  b From the Action drop-down list, select Deploy.

  c Select the archive to deploy from the drop-down Archive To Deploy list or browse to the archive you want to deploy using the ellipsis (…) button.

  d Set the Admin URL to the remote server, such as `<HOST NAME>:<port number>`.

  e Enter the User Name and Password that matches the remote server configuration.

  f Choose OK.

- Deploying from the project pane:
  - a Right-click the module you want to deploy in the project pane and choose Properties.
  - b Select Deployment in the tree to display the Deployment page.
  - c Set the Admin URL to Set the Admin URL to the remote server, such as `<HOST NAME>:<port number>`.
  - d Right-click the module you want to deploy in the project pane and choose Deploy Options|Deploy.

Deploying J2EE modules as an exploded directory

You can deploy WebLogic 7.x and 8.x J2EE modules as an exploded directory. You can set this option at the project-wide level, at the module level, or at deployment time.

To set this option at the project level,

1. Choose Project|Project Properties and select the Server page.
2. Open the Deployment node in the tree on the left to see the types of J2EE module modes.
3. Select the type of module node(s) you want to explode at deployment time.
4. Check the Deploy As Exploded Directory option on the module type page that appears.
5. When you are done selecting the types of modules you want to explode at deployment time, choose OK to close the dialog box.

To set this option at the module level,

1. Right-click the module you want to deploy as an exploded directory and choose Properties.
2. Select the Deployment page.
3. On the WebLogic 8.x page that appears, check the Deploy As Exploded Directory check box.
4. Choose OK to close the dialog box.

To set this option at deployment time,

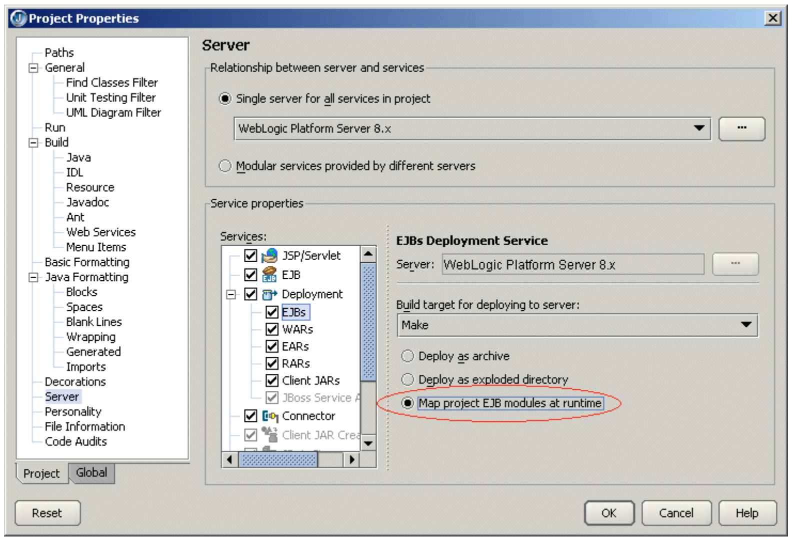
1. Choose Enterprise|Server Deployment to display the WebLogic Deploy Settings dialog box.
2. Check the Deploy As Exploded Directory check box.
3. Choose OK to close the dialog box.

Mapping EJB project modules at runtime

EJB project modules can be mapped at runtime. Mapping EJB modules modifies your domain config file by adding an application tag to it. The advantage to doing this is that the module is not deployed as an archive. You would want to use this feature only for development and not during production. Follow these steps:

1. Choose Project|Project Properties.
2. Select the Server page.
3. Open the Deployment node in the tree on the left to see the types of J2EE module modes.
4. Check the EJB node in the tree on the Server page and EJB module deployment properties appear.

5 Select the Map Project EJB Modules At Runtime option:



Server

Relationship between server and services

(●) Single server for all services in project

WebLogic Platform Server 8.x

( ) Modular services provided by different servers

Service properties

Services:

- [✓] JSP/Servlet
- [✓] EJB
- [✓] Deployment
  - [✓] EJBs
  - [✓] WARs
  - [✓] EARs
  - [✓] RARs
  - [✓] Client JARs
  - [✓] JBoss Service A
- [✓] Connector
- [✓] Client JAR Crea

EJBs Deployment Service

Server: WebLogic Platform Server 8.x

Build target for deploying to server: Make

( ) Deploy as archive

( ) Deploy as exploded directory

(●) Map project EJB modules at runtime

6 Choose OK to close the dialog box.

EJB modules that are included in EARs won't be mapped at runtime. If you try to deploy a module (either EJB or web) that has been mapped, JBuilder displays a warning. If you try to map a module (either EJB or web) that has been already deployed, the deployment will fail and JBuilder displays a warning. You'll also get a warning if you right-click a module, choose Properties, and in the Build options page select Never from the Build <Module> Directory drop-down list and then attempt to deploy as an exploded directory or to map an EJB or web module.

Redeploying individual classes

For WebLogic 8.x only, once you have mapped an EJB project module at runtime as explained in the previous section, you can then redeploy individual classes from the module directory:

1 Right-click the class you want to redeploy (reload) in the project pane.
2 Choose Redeploy.

WebLogic 8.x Platform Server deployment features

The OpenTool for WebLogic 8.x Platform Server has improved deployment features. You can now deploy submodules so you can deploy a module that has been newly added to an EAR or redeploy it.

Use the deploy options on the context menu when you right-click the archive and choose Properties. You can also access the deployment options when you choose Enterprise|Server Deployment and choose from the actions on the drop-down Action list.

There are two additional options on the context menu that appears when you right-click the archive and choose Deploy Options: Deactivate and Activate. Deactivate suspends deployed components, leaving staged data in place for subsequent reactivation. To reactivate a deactivated component, choose Activate. To totally remove the application and any staged data from the server, choose Undeploy.

JBuilder now allows WebLogic Server 7.x and 8.x EAR, WAR, RAR, and EJB modules to be deployed as an exploded directory. To permit this, follow these steps.

1 Choose Enterprise|Server Deployment to display the WebLogic Deploy Settings dialog box.

2 Check the Deploy As Exploded Directory option.

3 Choose OK to close the dialog box.

A few WebLogic Server properties, such as field groups and other CMP WebLogic-specific properties, were previously accessed only through the EJB Designer. You can now access them through the EJB DD Editor as well.

Remote debugging

To prepare to debug remotely, launch the WebLogic Server with debug parameters. To do this, modify `startWebLogic.sh` in your domain directory and add the following parameters to the JAVA_OPTIONS variable:

```
-Xdebug
-Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,address=3999,suspend=n
```

Within JBuilder, follow these steps:

1. In the project from which you want to launch the remote debug session, click Run|Configurations. If you have not yet created a server type run configuration, click New and select type Server. If you have already created a server type run configuration, select it and choose Edit.
2. Select Debug|Remote in the tree.
3. Check the Enable Remote Debugging option.
4. Enter the host name (the name of the machine on which the server is running.)
5. Click OK.
6. Click the down arrow next to the Debug Project icon ( ) on the JBuilder toolbar and select the debug run configuration you just created or edited. You will now be able to set breakpoints in Java code, such as in EJBs, servlets, and so on.

Remote debugging of JavaServer Pages

To debug JSPs remotely, open the project containing the web application you want to debug and follow these steps,

1. Expand the web application node in the project and expand the deployment descriptors node under the web application node.
2. Double-click `weblogic.xml` and add the following descriptor elements:

```
  <jsp-descriptor>
      <jsp-param>
        <param-name>keepgenerated</param-name>
        <param-value>true</param-value>
      </jsp-param>
      <jsp-param>
        <param-name>debug</param-name>
        <param-value>true</param-value>
      </jsp-param>
      <jsp-param>
        <param-name>precompile</param-name>
        <param-value>true</param-value>
      </jsp-param>
      <jsp-param>
        <param-name>compileFlags</param-name>
        <param-value>-g</param-value>
      </jsp-param>
    </jsp-descriptor>
```

3. Rebuild project and deploy the web application (as a WAR).

4 Launch the WebLogic server with debug parameters. To do this, modify `startWebLogic.sh` in your domain directory and add the following parameters to the JAVA_OPTIONS variable:

```
-Xdebug
-Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,address=3999,suspend=n
```

Note that these steps can be done on a local machine provided WebLogic is set up and configured in JBuilder.

On the local machine, follow these steps:

1 Transfer the project to the local machine where you wish to debug the JSP and open the project in JBuilder.

2 Under the source path for the project (normally `<PROJECT_ROOT>/src`), create a directory named `jsp_servlet`. If you need to use a source path other than the default, you can obtain this setting from the General page of the Project Properties dialog box.

3 Copy the JSP(s) you want to debug to the directory you just specified.

4 Open the JSP(s) and set breakpoints where required.

5 Choose Run|Configurations.

6 Click the New button, select the Application from the drop-down list, and click the Debug tab.

7 Check the Enable Remote Debugging option.

8 Enter the host name of the remote server in the Host Name field and set the Build Target to None. Accept all other default settings.

9 Attach to the remote server by clicking on the debug project icon in the toolbar.

10 Load your JSP in a web browser. The debugger stops at breakpoints in the JSP.

Updating projects with the latest server settings

Every time you open a project, JBuilder updates it with the latest server settings for the project's selected server(s). So, for example, if you have modified server settings for one project and you have others that use the same server configuration, the next time you open those other projects, your modified server settings will be in force automatically. If you want to use a similar but unique server configuration for a particular project, create a duplicate configuration using the Copy button, and use the Copy Servers wizard to edit it to your needs.

Be aware that this automatic updating of projects with the latest server settings can occur only with server configurations modified in JBuilder 9. If you attempt to transfer server libraries from a previous JBuilder version, your projects won't be updated with them. You can still tell JBuilder to update the project manually:

1 Right-click the project node of the project you want to update in the project pane and choose Properties.

2 Select the Servers page.

3 Click the ellipsis (…) button next to your selected single server or the separate service servers and make your changes.

4 Click OK to close the dialog box.

Chapter

8

Using JBuilder with IBM WebSphere servers

This chapter explains how set up and use IBM WebSphere servers with JBuilder. JBuilder supports WebSphere Application Server 4.0.7 Single Server, WebSphere Application Server 4.0.7 Advanced Edition, and WebSphere Application Server 5.0.2.4, 5.1.0.4.

Configuring JBuilder for WebSphere servers

To configure JBuilder settings to target WebSphere servers,

1 Choose Enterprise|Configure Servers.

  The Configure Servers dialog box appears.

2 Select the WebSphere server you want to configure by clicking it in the pane on the left.

  The right side of the dialog box lists the default settings for the selected server. The General page has fields that all servers have in common, while the Custom page has fields that are specific to the selected server. In some cases, modifying a Custom setting will update a setting on the General page.

3 Check the Enable Server check box at the top of the dialog box.

  Checking this option enables the fields for the selected application server. You won't be able to edit any fields until it is checked. The Enable Server check box also determines whether this server will appear in the list of servers when you select a server for your project using the Servers page of the Project|Project Properties dialog box.

  JBuilder provides default settings for the selected server. If JBuilder's default settings are not appropriate, edit any of the settings as needed. Some servers require you to enter settings in addition to the defaults. Clicking the OK button in this dialog box when not all required values are set or selecting another server while the current one is enabled displays a message dialog box that informs you about the missing settings.

4 If you want to change any of the settings, click the ellipsis (…) button next to the field and make your changes. If you installed WebSphere into `[drive]:/Program Files/WebSphere`, the default Home Directory is `[drive]:/Program Files/WebSphere/AppServer`. Otherwise, the Home Directory becomes whichever directory you specify.

5 Click the Custom tab to view fields unique to the server. Change or fill in these fields:

For WebSphere Application Server 4.0.7:

- IBM JDK Installation Directory: Where the IBM installation of the JDK is installed. JBuilder uses this field to locate the proper version of the JDK the server requires. This field is set when you specify the Home Directory, so it should be the `java` directory of the Home Directory; for example, the directory would be `[drive]:/Program Files/WebSphere/AppServer/java` if `[drive]:/Program Files/WebSphere/AppServer` is your Home Directory.
- DB2 Installation Directory: The directory in which DB2 is installed on your system. Usually this is in `[drive]:/SQLLIB`.
- Add An Administrator's Console Item to The Enterprise Menu: Checking this check box adds the WebSphere Admin Console item to JBuilder's Enterprise menu so you have quick access to it from the JBuilder IDE.
- Add an Application Assembly Tool Item to The Enterprise Menu: Checking this check box adds the WebSphere Application Assembly item to JBuilder's Enterprise menu so you have quick access to this tool from the JBuilder IDE.

For WebSphere Application 5.0.2.4, 5.1.0.4:

- JDK Installation Directory: Where the IBM installation of the JDK is installed. This field is set when you specify the Home Directory, so it should be the `java` directory of the Home Directory; for example, `[drive]:/Program Files/WebSphere/AppServer/java`.
- Embedded Message Path: The directory where WebSphere's messaging software is located. This field is set by default and is usually in the `Ibm/WebSphere MQ` directory. So, if you installed WebSphere into `[drive]:/Program Files/WebSphere`, the Embedded Messaging Path would be `[drive]:/Program Files/Ibm/WebSphere MQ`. You can, however, specify any path directly that contains the messaging software.
- Cell Name: The cell name is set by default. It is read in from your local server configuration. You can edit this field, but you will seldom need to do so. For more information about cells, see your WebSphere documentation.
- Node Name: The node name is set be default. It is read in from your local server configuration. You can edit this field, but you will seldom need to do so. For more information about nodes, see your WebSphere documentation.
- Server Name: The server name which you are configuring. It is read in from your local server configuration. JBuilder suggests a default name, which you can change to meet your needs.
- Add An Administrative Console Item To The Enterprise Menu: Checking this check box adds the WebSphere Admin Console item to JBuilder's Enterprise menu so you have quick access to it from the JBuilder IDE. You must also fill in the next field, Web Browser Path, to have the menu item added.
- Web Browser Path: Specify the path and file name of the web browser of your choice. For example, you might specify `c:/Program Files/Internet Explorer/iexplore.exe` if you want to use Microsoft's Internet Explorer.
- Add An Application Assembly Item To The Enterprise Menu: Checking this check box adds the WebSphere Application Assembly item to JBuilder's Enterprise menu so you have quick access to this tool from the JBuilder IDE.

- Add An Administrative Scripting Item To The Enterprise Menu: Checking this check box adds the WebSphere Administrative Scripting item to JBuilder's Enterprise menu so you have quick access to this tool from the JBuilder IDE.
- This step is optional. Click the CMP Mapping button to display the CMP Mapping page. Use this page to view and possibly change the container-managed persistence (CMP) mappings JBuilder uses. By examining the tables on this page, you can see the mappings JBuilder is using. Usually you will not need to change any of these mappings, especially if you are using the more common database drivers. You can, however, edit these mappings if you are using other drivers, want to do your own type mappings, and so on. For more information about editing mappings and aliases, click the Help button.

6 Choose OK to close the dialog box and configure JBuilder.

When you configure JBuilder to target a WebSphere server, required libraries are created for you. To see a list of the WebSphere libraries, see "The created libraries" on page 23.

Selecting a server

If you are working with more than one version of WebSphere Application Server and have configured JBuilder for all versions, you must select which version you want to use for your current project. Choose Project|Project Properties and choose Server. See "Selecting a server" on page 24 for more information.

Generating WebSphere deployment descriptors for WebSphere 4.0

When you are using the Enterprise JavaBean 1.x wizard or the EJB 1.x Entity Modeler to create entity beans that target WebSphere 4.0 Advanced Edition, the wizards do not generate the `Map.mapxmi` and `Schema.dbxmi` deployment descriptors. When you build your bean, EjbDeploy, which is called during the build process, will generate them for you. You can then modify the mapping and run Make again to keep the mapping in the JAR.

If you've used the Entity Modeler to create your entity beans and the field names in your bean don't map directly to columns in the database, you can choose to have JBuilder create WebSphere CMP deployment descriptors that override the default EjbDeploy behavior:

1 Right-click the EJB module node in the project pane when WebSphere 4.0 Advanced Edition is your selected server.

2 Choose Properties on the context menu.

3 Select the Build|WebSphere page.

4 Check the Generate CMP Descriptors check box.

5 Make your changes:

The list of Data Mapping Files contains the default mapping files that are used to map Java primitive types to database types. You can modify, add, and remove mapping files. The Database Type Substitution In Generated Descriptors list displays the substitutions JBuilder makes if the Java primitive types can't be directly mapped to a type in the database. You can modify this list. For example, you might want another Java type to be substituted for a particular original type in a specified database. Or you can add additional substitutions to the list.

6 Click OK.

When you compile your bean, the two deployment descriptors (`Map.mapxmi` and `Schema.dbxmi` are generated for entity beans with container-managed persistence for the WebSphere 4.0 Advanced Edition only. If you are using one edition of WebSphere 4.0 and change your target server to the other version, you must recompile your project to ensure JBuilder generates the correct deployment descriptors for you.

If the database schema name used is different from the username, manually modify the `Schema.dbxmi` file as follows:

1 Make the project to generate the CMP descriptors.

2 Double-click the EJB module node in the project pane to open the EJB module viewer.

3 Click the source view for `Schema.dbxmi`.

4 Add the following line above the last line of the source:`<RDBSchema:RDBSchema xmi:id="RDBSchema_1" name="<your_schema_name>" database="<database_id>" tables="<table_id>"/>`

5 Rebuild the project to incorporate the change.

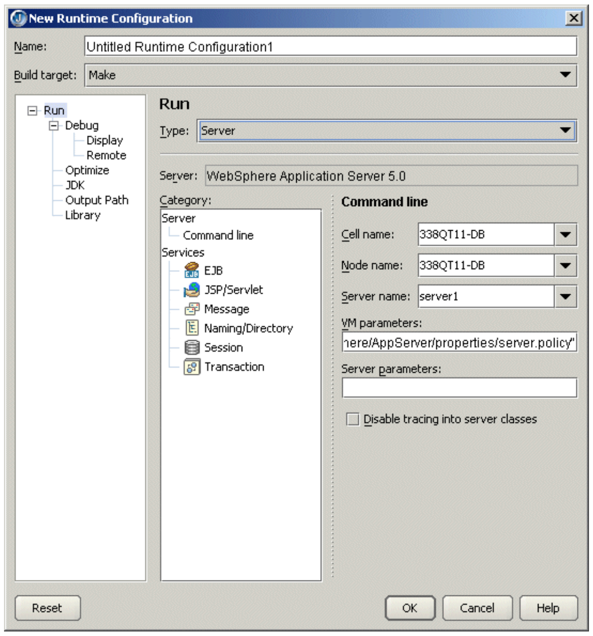
Starting the server

To prepare to start the server from within JBuilder, create a server runtime configuration. Follow these steps:

1 Choose Run|Configurations.

2 In the dialog box that appears, click the New button.

3 Select Server from the Type drop-down list.



Name: Untitled Runtime Configuration1
Build target: Make
Run
Type: Server
Server: WebSphere Application Server 5.0
Cell name: 338QT11-DB
Node name: 338QT11-DB
Server name: server1
VM parameters: here/AppServer/properties/server.policy"
Server parameters:

4 In the Name field, enter a name, such as Server or something that will identify the run configuration.

5 Fill in the VM Parameters and Server Parameters needed to run the server. If you've selected a target application server as described in "Selecting a server" on page 75, default values are already in place.

6 Click OK.

To start the server, click the down-arrow next to the Run Project icon (▶) and select the server run configuration you just created.

By default in WebSphere 5.x, the stdout and stderr streams are redirected to log files at application server startup. To view the output in JBuilder when you start the server in JBuilder:

1. Start the server.
2. Start the Administrative Console from the JBuilder Enterprise menu.
3. Choose Servers|Application Servers|<server_name>|Process Definition|ProcessLogs.
4. On the Configuration page, set the Stdout File Name and the Stderr File Name to Console.

Deploying

The following sections describe deploying to both the WebSphere Application Server 5.x and the WebSphere Application Server 4.0.

Using the DD Editor

The DD Editor has several WebSphere-specific pages for information that applies only to WebSphere servers. For information about using the DD Editor to edit J2EE modules, see Chapter 11, "Editing J2EE deployment descriptors." Whenever a WebSphere-specific page is displayed in the DD Editor, you can press *F1* for help on that particular page. You'll also find it convenient to have a copy of your server's documentation handy to look up the type of information your server requires in the deployment descriptors.

To prepare to deploy a deployable module, follow the steps outlined for your version of WebSphere:

WebSphere Application Server 5.x

This section explains how to deploy remotely and locally to the WebSphere Application Server 5.x.

Remote deploying

WebSphere Application Server 5.x supports remote deployment using the Deployment Manager version of the application server. The Deployment Manager must be configured to manage the remote server instance. Please refer to your server documentation for more information.

Once you have configured your Deployment Manager to manage the remote server instance, you can deploy to the remote server instance in JBuilder if you follow these steps:

1. Right-click the archive node you want to deploy in JBuilder's project pane and choose Properties.
2. Select Deployment in the tree to display the Deployment page and make sure the General tab is selected.
3. Set the Cell Name to the Deployment Manager's cell name.
4. Set the Node Name to the remote server node name.
5. Set Server Name to the remote server name.
6. Click the Connection tab.
7. Set the Connection Type field to SOAP or RMI.

8 Set the Host field to the host name where the Deployment Manager is running.
9 Set the Port field to the port number of the Deployment Manager.
10 Set the User Name and Password fields if they are required; for example, if security is enabled.

Deploying locally

To deploy locally to a server which is not running,

1 Right-click the node you want to deploy in JBuilder's project pane and choose Properties.
2 Select Deployment in the tree and click the Connection tab.
3 Set the Connection Type to NONE.

If the connection type is set to NONE, deployment actions performed while the server is running won't start or stop the deployed module.

WebSphere Server 4.0

If you are using WebSphere 4.0 Advanced Edition, you must create an EAR group that contains your beans. You then deploy the EAR group. For more information about creating EAR groups, see "Creating an EAR file" in *Developing Applications with Enterprise JavaBeans*.

Enterprise|Server Deployment displays a WebSphere Deployment dialog box. Configure your deployment settings with this dialog box. JBuilder passes the settings you specify on to the WebSphere deployment tool. The deployment tool for WebSphere 4.0 differs depending on the version of the server you are using. For the Single Server, WebSphere's deployment tool is SEAppInstaller. For the Advanced Edition, the deployment tool is XmlConfig. Therefore, the appearance of the Deploy Settings dialog box will vary between these two WebSphere Server 4.0 editions.

If your target application server is WebSphere 4.0 Advanced Edition and you want an XML file to be generated as input to the WebSphere XMLConfig utility, check the Generate XML check box. If this option isn't checked, the file won't be created. If you make your own modifications to the generated XML file (named `deploy_<selectednode>.xml` and appearing under the EJB module node or EAR group), uncheck this option to be sure you don't lose your changes. If you use the Deployment Options on the context menu (right-click the EJB module and choose Deployment Options <jar name>.jar to see the deployment commands), the generated XML file is `deploy.xml`. It appears under the project node.

Remote deploying

Follow these steps to use the WebSphere Deploy Settings dialog box to deploy to a remote server:

1 Choose Enterprise|Server Deployment.
2 Select the Deploy action.
3 Specify these options in the Options field:

```
-nameServiceHost<host name> -nameServicePort<port number>
```

4 Set the Primary Node and Server Name to match your server configuration.
5 Click OK.

You can also deploy a module to a remote server from JBuilder's project pane:

1 Right-click the module you wish to deploy.

2 Select Properties.

3 Select Deployment in the tree to display the Deployment page.

4 Specify these options in the Options field:

```
-nameServiceHost<host name> -nameServicePort<port number>
```

5 Set the Primary Node and Server Name to match your server configuration.

6 Right-click the module in the project pane and choose Deploy Options|Deploy.

Deploying locally

These are the deploy options you'll find on the Deploy Options context menu:

- Deploy — Deploys a JAR to the currently running container of the project application server. If the Build Target option is Make for the current runtime configuration (Choose Run|Configurations, select the current runtime configuration, and click Edit then click Server and select Deployment in the tree of services to see the Build Target), this option will "make" the JAR's contents before deploying it to the container.
- Redeploy — Deploys a JAR again to the currently running container. If the Build Target option is Make for the current runtime configuration (Choose Run| Configurations, select the current runtime configuration, and click Edit then click Server and select Deployment in the tree of services to see the Build Target), this option will "make" the JAR's contents before redeploying it to the container.
- Undeploy — Undeploys an already deployed JAR in the running container.
- List Deployments — Lists all JARs deployed in the running container.
- Stop Container — Stops the container. This option appears for WebSphere 4.0 Advanced Edition only. When a deployed EJB changes, the container must be stopped and then restarted for the changes to register.
- Start Container — Starts the container. This option appears for WebSphere 4.0 Advanced Edition only.

Enabling remote debugging

The steps to enable remote debugging of a bean running on a WebSphere server vary depending on which version of WebSphere you are using.

Note You won't be able to debug JSPs in a remote debug session.

WebSphere Application Server 5.x

Follow these steps to enable remote debugging for WebSphere Application Server 5.x:

1 Launch the server with the `-script` option:

```
WEBSPHERE_HOME/bin/startserver -script
```

This command should write a script called `start_<server name>` in the `WEBSPHERE_HOME/bin` directory.

2 Edit the launch script and add the following remote debug parameters to the java command line:

```
-Xdebug
 -Djava.compiler=NONE
 -Xrunjdwp:transport=dt_socket,server=y,address=3999,suspend=n
```

3 Launch the server using the script.

4 In JBuilder, choose Project|Project Properties. If you have not yet created a server run configuration, choose New and create a new configuration of type Server. If you already have a server run configuration, choose Edit.

5 Select Debug|Remote in the tree and check the Enable Remote Debugging and Attach options and click OK.

Click the down arrow next to the Debug Project icon ( ) on the JBuilder toolbar and select the debug run configuration you just created or edited. You will now be able to set breakpoints in Java code, such as in EJBs, servlets, and so on.

WebSphere Single Server 4.0

If you are using WebSphere Single Server 4.0, follow these steps to enable remote debugging:

1 Launch the server with the `-script` option:

```
WEBSPHERE_HOME/bin/startserver -script
```

This command should write a script called launch in the `WEBSPHERE_HOME/bin` directory.

2 Edit the launch script and add the following remote debug parameters to the java command line:

```
-Xdebug
-Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,address=3999,suspend=n
```

3 Launch the server using the script.

4 In JBuilder, choose Project|Project Properties. If you have not yet created a server run configuration, choose New and create a new configuration of type Server. If you already have a server run configuration, choose Edit.

5 Select Debug|Remote in the tree.

6 Check the Enable Remote Debugging and Attach options and click OK.

Click the down arrow next to the Debug Project icon ( ) on the JBuilder toolbar and select the debug run configuration you just created or edited. You will now be able to set breakpoints in Java code, such as EJBs, servlets, and so on.

WebSphere Server Advanced Edition 4.0

WebSphere Server Advanced Edition 4.0 launches another Java virtual machine for debugging, so you must follow instructions so you can debug your enterprise beans:

1 Start the WebSphere Server 4.0.

2 Start the WebSphere Server 4.0 console application (which you'll find at `<ws4_ae home>\bin\adminclient.bat` if you're running on Windows.)

3 When the console application appears, expand the tree on the left until you see the Application servers node. Select this node.

4 In the panel on the right side of the console, click the JVM Settings tab. Click the Advanced Settings button. (You might need to scroll the panel to see this button.)

5 In the dialog box that appears, enter the following VM options:

```
-Xdebug
-Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,address=3999,suspend=n
```

6 If you start the server outside of JBuilder, choose Run|Configurations and click the New button in the dialog box that appears. In the Configuration Name field, specify a name for the new run configuration server for debugging on your application server.

7 Select Debug|Remote and make these changes on the Debug page:

  a Check the Enable Remote Debugging check box.

  b Select the Attach option.

  c In the Port Number field, enter the port number you noted during the server startup process.

  d Click OK twice to close all dialog boxes.

8 Click the down arrow next to the Debug Project icon ( ) on the JBuilder toolbar and select the debug run configuration you just created or edited.

  You will be attaching to the application server process. Now you can deploy your web applications and you'll be able to stop at breakpoints you set in the bean, JSPs, or servlets.

To start the server in debug mode within JBuilder, follow these steps:

- Choose Run|Configurations.
- Select the Command node and enter the remote debug port number in the Debug Transport Address field.

Now clicking the debug toolbar button launches the server in debug mode and attaches to it, all in one session.

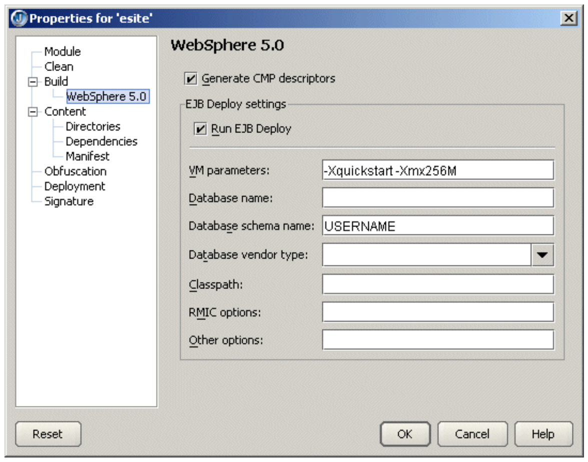
Container-managed persistence (CMP) 1.1 and 2.0 in WebSphere 5.x

You can set the schema name and database vendor type as properties on EJB module nodes:

1 Right-click the EJB module and choose Properties.

2 Select Build|WebSphere 5.x in the tree.

3 Enter your schema and database vendor information in this dialog box:



WebSphere 5.0

☑ Generate CMP descriptors

EJB Deploy settings

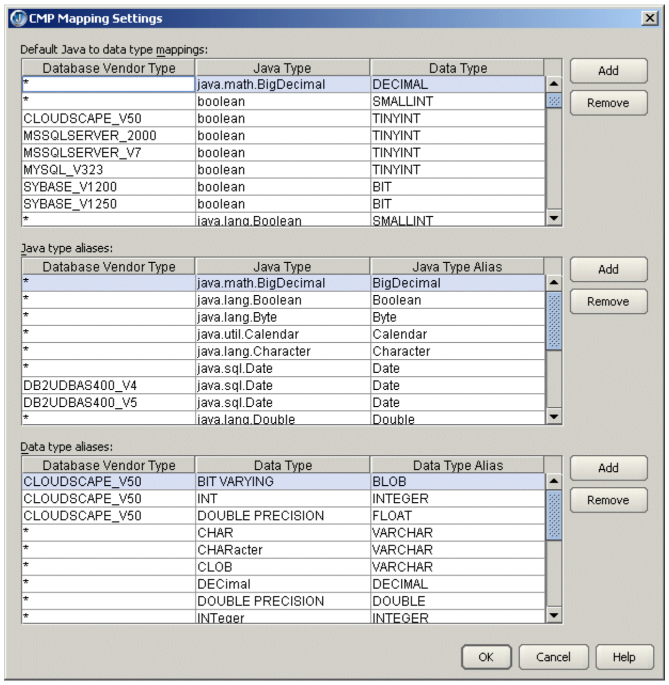
☑ Run EJB Deploy

VM parameters: -Xquickstart -Xmx256M
Database name:
Database schema name: USERNAME
Database vendor type:
Classpath:
RMIC options:
Other options:

4 Click OK.

To change the default CMP mapping behavior such as the Java to database type mappings,

1 Choose Enterprise|Configure Servers.
2 Select WebSphere Application Server 5.x in the list of application servers.
3 Click the Custom button.
4 Click the CMP Mapping button.
5 Make your changes in the CMP Mapping Settings dialog box that appears:



Default Java to data type mappings:

Database Vendor Type	Java Type	Data Type
*	java.math.BigDecimal	DECIMAL
*	boolean	SMALLINT
CLOUDSCAPE_V50	boolean	TINYINT
MSSQLSERVER_2000	boolean	TINYINT
MSSQLSERVER_V7	boolean	TINYINT
MYSQL_V323	boolean	TINYINT
SYBASE_V1200	boolean	BIT
SYBASE_V1250	boolean	BIT
*	java.lang.Boolean	SMALLINT

Java type aliases:

Database Vendor Type	Java Type	Java Type Alias
*	java.math.BigDecimal	BigDecimal
*	java.lang.Boolean	Boolean
*	java.lang.Byte	Byte
*	java.util.Calendar	Calendar
*	java.lang.Character	Character
*	java.sql.Date	Date
DB2UDBAS400_V4	java.sql.Date	Date
DB2UDBAS400_V5	java.sql.Date	Date
*	java.lang.Double	Double

Data type aliases:

Database Vendor Type	Data Type	Data Type Alias
CLOUDSCAPE_V50	BIT VARYING	BLOB
CLOUDSCAPE_V50	INT	INTEGER
CLOUDSCAPE_V50	DOUBLE PRECISION	FLOAT
*	CHAR	VARCHAR
*	CHARacter	VARCHAR
*	CLOB	VARCHAR
*	DECimal	DECIMAL
*	DOUBLE PRECISION	DOUBLE
*	INTeger	INTEGER

6 Click OK.

These are your options:

- Default Java To Data Type Mappings: This list is used if JBuilder is unable to resolve the database schema type for a CMP field. This could happen if the JDBC driver fails to report a type for a database column or if the corresponding column (as defined in the CMP field mapping) is missing.
- Java Type Aliases: This list is used to translate between actual Java types in the bean and types defined in IBM database type mapping files.
- Data Type Aliases: This list is used to translate between database vendor types (as reported by the JDBC driver) and types defined in IBM database type mapping files.

Chapter

9

Using JBuilder with JBoss servers

This chapter explains how set up and use the JBoss application server with JBuilder. JBuilder officially supports JBoss 3.2.5 and JBoss 3.0.8.

Configuring JBuilder for the JBoss application server

To configure JBuilder settings to target JBoss servers,

1 Choose Enterprise|Configure Servers.

The Configure Servers dialog box appears.

2 Select the JBoss 3.x+ server by clicking it in the pane on the left.

The right side of the dialog box lists the default settings for the selected server. The General page has fields that all servers have in common, while the Custom page has fields that are specific to the selected server. In some cases, modifying a Custom setting will update a setting on the General page.

3 Check the Enable Server check box at the top of the dialog box.

Checking this option enables the fields for the selected application server. You won't be able to edit any fields until it is checked. The Enable Server check box also determines whether this server will appear in the list of servers when you select a server for your project using the Servers page of the Project|Project Properties dialog box.

JBuilder provides default settings for the selected server. If JBuilder's default settings are not appropriate, edit any of the settings as needed. Some servers require you to enter settings in addition to the defaults. Clicking the OK button in this dialog box when not all required values are set or selecting another server while the current one is enabled displays a message dialog box that informs you about the missing settings.

4 If you want to change any of the settings, click the ellipsis (…) button next to the field and make your changes. If you installed JBoss into `[drive]:/jboss325`, for example, the default Home Directory would be `[drive]:/jboss325/jboss-3.2.5`.

5 Click the Custom tab to view fields unique to the server. Change or fill in these fields:

- Server Version: Your choices are server version 3.2.5 and 3.0.8. These are the versions of JBoss JBuilder officially supports.
- Configuration Name: Specify the configuration you want to begin using. You can select a configuration from the drop-down list, or type in your own configuration name. The configuration you specify must exist in the `server` directory, which is a subdirectory of the Home directory.
- Deployment Path: Specify the path used to deploy your modules. By default, the deployment path will be in the `deploy` subdirectory of the directory you specified as the Configuration Name.
- Include System Modules When Listing Deployments: JBuilder can list all your deployed modules. If you wish to see the system modules such as the web container or the EJB container and other services that are a core part of the JBoss server, check this option.
- Include Disabled Modules When Listing Deployments. JBuilder can list all your deployed modules. If you wish to see modules which are currently disabled, check this option. A disabled module is simply one that has been renamed to <module name>.disabled when you chose to disable it within JBuilder using the Disable deployment option.

6 Choose OK to close the dialog box and configure JBuilder.

When you configure JBuilder to target a JBoss server, required libraries are created for you. To see a list of the JBoss libraries, see "The created libraries" on page 23.

Selecting a server

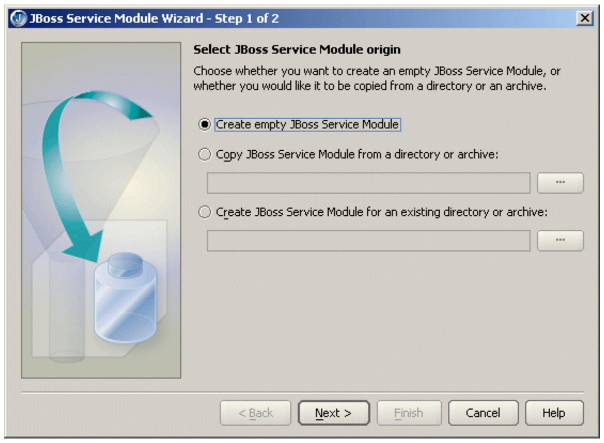
If you are working with more than one version of JBoss Application Server and have configured JBuilder for all versions, you must select which version you want to use for your current project. Choose Project|Project Properties and select Server in the tree to display the Server page. See "Selecting a server" on page 24 for more information.

Creating a JBoss service module

You can use JBoss to create a service module for extending the JBoss server. A service module contains the definition of a service. Once you compile the service module, a service archive (SAR) is created, ready to be deployed to JBoss.

To begin creating a JBoss service module within JBuilder, follow these steps:

1 Choose File|New|Enterprise and double-click the JBoss Service Module wizard. The wizard appears:


JBoss Service Module Wizard - Step 1 of 2

Select JBoss Service Module origin

Choose whether you want to create an empty JBoss Service Module, or whether you would like it to be copied from a directory or an archive.

(●) Create empty JBoss Service Module

( ) Copy JBoss Service Module from a directory or archive:

( ) Create JBoss Service Module for an existing directory or archive:

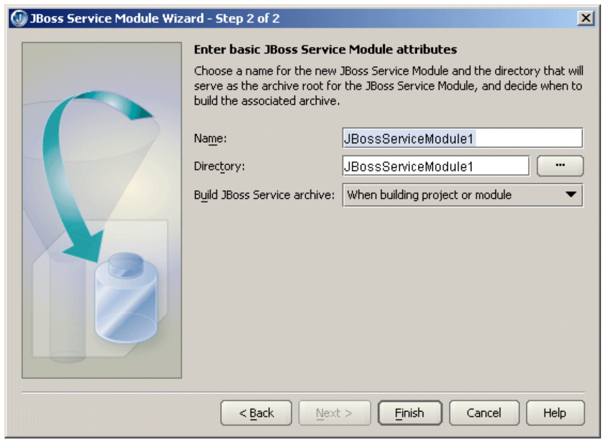
2 Decide whether you are creating an empty JBoss service module that you will fill in later, if you are copying an existing JBoss service module from a directory or archive, if you are creating a JBoss module node that represents a service module in a directory outside your project, or if you are creating a JBoss service node that represents an archive (.sar).

Because your selection determines the behavior of the wizard, the next sections describe how to use the wizard depending on the task you want to accomplish.

If you want to create a new empty JBoss service module,

1 Select the Create Empty JBoss Service Module option on the first page of the JBoss Service Module wizard.

2 Click Next to see Step 2:



JBoss Service Module Wizard - Step 2 of 2

Enter basic JBoss Service Module attributes

Choose a name for the new JBoss Service Module and the directory that will serve as the archive root for the JBoss Service Module, and decide when to build the associated archive.

Name: JBossServiceModule1

Directory: JBossServiceModule1

Build JBoss Service archive: When building project or module

3 Specify a name to identify the new service module in the Name field. The default will be the name of the selected directory.

4 Specify a directory name for the Directory field. This is the directory that will contain the module. By default, the wizard uses the same name for the directory as the name you specified in the Name field. You can use the ellipsis (…) button to navigate to a directory you want to use instead, if you choose.

5 From the Build JBoss Service Archive drop-down list, select when you want the JBoss service archive to be built.

6 Choose Finish.

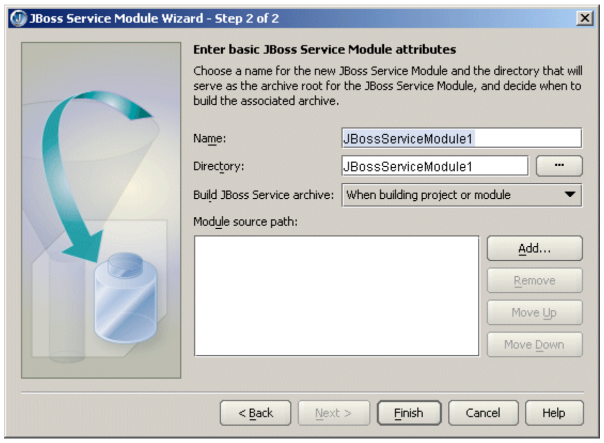
Copying an existing JBoss service module

You can use an existing JBoss service module by copying it to your current project with the JBoss Service Module wizard:

1 Select the Copy JBoss Service Module From A Directory Or Archive option on the first page of the JBoss Service Module wizard.

2 Use the ellipsis (…) button to browse to the location of the directory or archive that contains the module or archive you want to copy to the your project.

3 Click Next.



Enter basic JBoss Service Module attributes

Choose a name for the new JBoss Service Module and the directory that will serve as the archive root for the JBoss Service Module, and decide when to build the associated archive.

Name: JBossServiceModule1

Directory: JBossServiceModule1

Build JBoss Service archive: When building project or module

Module source path:

4 Specify a name to identify the new module in the Name field.

5 Specify a directory name for the Directory field. By default, the wizard uses the same name for the directory as the name you specified in the Name field. You can use the ellipsis (…) button to navigate to a directory you want to use instead, if you choose.

6 You might want to add Java source paths that are necessary to build a module. It isn't absolutely necessary to add the source paths in this wizard, however. You can also add additional source path directories to your project directly. If you choose to add the source path directories using the JBoss Service Module wizard, the directories you specify will be added to your project source path. If you choose to add to the module's source path, click the Add button and navigate to the Java source file directories you want to add. Continue until all source file directories are added. You can order the source file directories using the Move Up and Move Down buttons on the page, and you can select a previously added source file directory and delete with the Remove button.

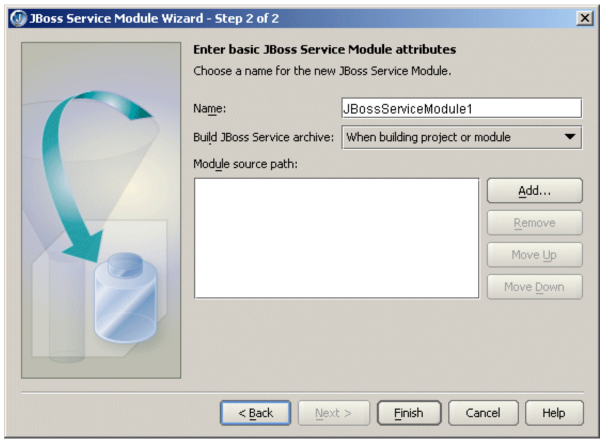
7 Choose Finish.

You can use an existing module in a directory outside your project; you don't have to copy it to your current project:

1 Select the Create A JBoss Service Module For An Existing Directory Or Archive on the first page of the JBoss Service Module DD Editor.

2 Use the ellipsis (…) button to browse to the location of the directory that contains the module.

3 Click Next.



Enter basic JBoss Service Module attributes

Choose a name for the new JBoss Service Module.

Name: JBossServiceModule1

Build JBoss Service archive: When building project or module

Module source path:

4 Specify a name to identify the new JBoss service module in the Name field.

5 From the Build JBoss Service Archive drop-down list, select when you want the service archive to be built.

6 Use the Add button to add the source path for any Java files associated with the module.

   You might want to add Java source paths that are necessary to build a module. It isn't absolutely necessary to add the source paths in this wizard, however. You can also add additional source path directories to your project directly. If you choose to add the source path directories using the JBoss Service Module wizard, the directories you specify will be added to your project source path. If you choose to add to the module's source path, click the Add button and navigate to the Java source file directories you want to add. Continue until all source file directories are added. You can order the source file directories using the Move Up and Move Down buttons on the page, and you can select a previously added source file directory and delete with the Remove button.

7 Choose Finish.

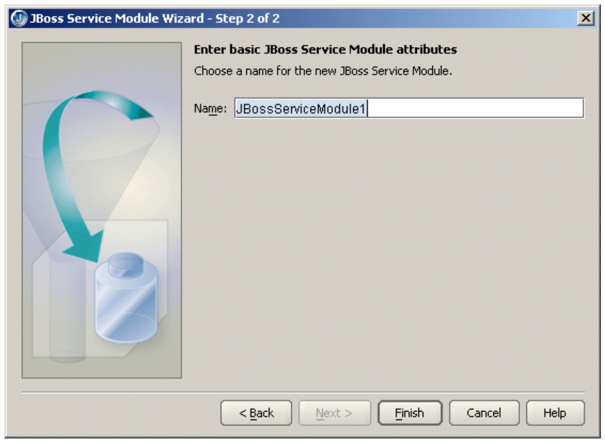
Creating a JBoss service node for an existing service module archive (SAR) in a directory outside your project

You can create a JBoss service module archive node that is outside your current project. Although you will be able to view the contents of the archive using the JBoss Service Module DD Editor, you won't be able to edit it. The module node will be read only. You also won't be able to build it, but you will be able to deploy it.

1 Select the Create A JBoss Service Module For An Existing Directory Or Archive on the first page of the JBoss Service Module DD Editor.

2 Use the ellipsis (…) button to browse to the location of the directory that contains the module.

3 Click Next.


JBoss Service Module Wizard - Step 2 of 2

Enter basic JBoss Service Module attributes

Choose a name for the new JBoss Service Module.

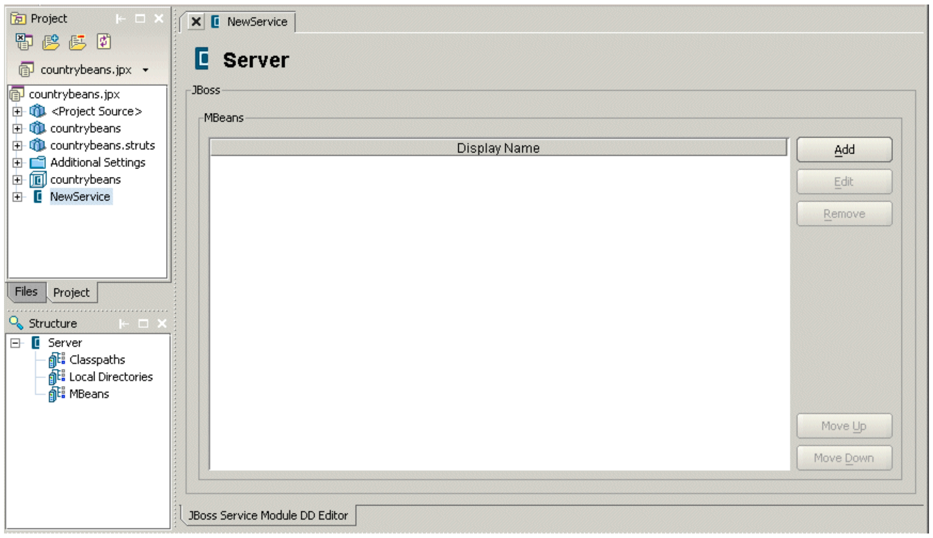
Name: JBossServiceModule1

4 Specify a name to identify the new module in the Name field.

5 Choose Finish.

Editing the JBoss service module deployment descriptor

Once you've created a JBoss service module, you edit the module's deployment descriptor using JBuilder's JBoss Service Module DD Editor. To display the editor, double-click the new service module that appears in the project pane:



Now you can customize the service module deployment descriptor (`jboss-service.xml`) with the values you need for your new service. The JBoss Service Module DD Editor works just like the other J2EE DD Editors in JBuilder. To learn how to add pages and edit them in the DD Editor, see Chapter 11, "Editing J2EE deployment descriptors."

In the structure pane, you can see that there are JBoss-specific nodes for Classpaths, Local Directories, and MBeans. Add a Classpath, Local Directory, or MBean by right-clicking the category of the item you are adding and choosing Add on the context menu that appears. Then double-click that new entry in the structure pane to display a page for it in the JBoss Service Module DD Editor. For assistance filling in the fields on these pages, display the page and press *F1*. You'll find JBoss documentation useful in filling in the fields, too.

At any time you can view the `jboss-service.xml` descriptor in its source form. Expand the service module node in the project pane to see the Deployment Descriptors node and then expand that node too. The `jboss-service.xml` node becomes visible. Double-click it to see the it in the content pane. You can edit the file by hand, if you wish.

MBeans

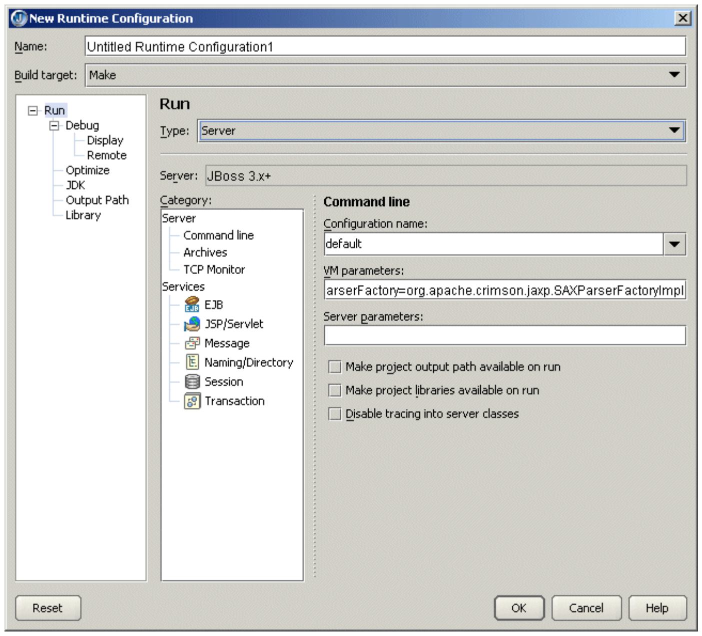
All JBoss services are in MBeans. So your service module will have one or more MBeans. Once you've added an MBean entry to the module, expand the entry in the structure pane to see Attributes, Depends Items, and Depends Lists nodes. Add entries to these categories by right-clicking them and choosing Add. Press *F1* for help filling in the fields.

Starting the server

To prepare to start the server from within JBuilder, create a server runtime configuration. Follow these steps:

1. Choose Run|Configurations.
2. In the dialog box that appears, click the New button.

3 Select Server from the Type drop-down list.



4 In the Name field, enter a name, such as Server or something that will identify the run configuration.

5 Fill in the VM Parameters and Server Parameters needed to run the server. If you've selected a target application server as described in "Selecting a server" on page 84, default values are already in place.

6 Click OK.

To start the server, click the down-arrow next to the Run Project icon (▶) and select the server run configuration you just created.

Deploying locally

To deploy a deployable module to a local server, choose one of these options:

- Using the Enterprise|Server Deployment command:
  - **a** Choose Enterprise|Server Deployment to display the JBoss 3.x+ Deploy Settings dialog box.
  - **b** From the Action drop-down list, select Deploy.
  - **c** Select the archive to deploy from the drop-down Archive To Deploy list or browse to the archive you want to deploy using the ellipsis (…) button.
  - **d** Specify the correct deployment path for JBoss Server. Accept the default value or browse to the correct directory using the ellipsis (…) button.
  - **e** Choose OK.
- Deploying from the project pane:
  - **a** Right-click the module you want to deploy in the project pane and choose Properties.
  - **b** Select Deployment in the tree to display the Deployment page.
  - **c** Specify the correct deployment path for JBoss Server. Accept the default value or browse to the correct directory using the ellipsis (…) button.
  - **d** Choose OK.
  - **e** Right-click the module you want to deploy in the project pane and choose Deploy Options|Deploy.

Both the Server Deployment dialog box and the Deployment page of the Project|Project Properties dialog box have these two options:

- Include System Modules When Listing Deployments
- Include Disabled Modules When Listing Deployments

These options allow you to fine tune what you see when you list deployments using either the List Deployments action in the Server Deployment dialog box or when you right-click a module and choose Deploy Options|List Deployments. Listed deployments appear in a tree in the message pane. A system module is one that extends the server and provides a service your modules can use. A disabled module is one you mark as Disabled by right-clicking the module and choosing Deploy Options|Disable. When a module is disabled, it is simply renamed to <module>.disabled. To enable a module again, right-click the module and choose Deploy Options|Enable.

Deploying exploded web modules

To deploy exploded modules, follow these steps:

1. Create a web module in a directory with the name <modulename>.<fileext>, such as `Module1.war`, for example.
2. Ensure that the directory that contains the web module is at least one level below the project directory. So, for example, if your project is created in a directory named `Project1`, you might create your module in a directory named `Project1/deploy/<modulename>.<fileext>`.
3. Edit the jboss-service.xml file in your JBoss configuration directory (`<CONFIG_NAME>/conf`). Add the parent directory of your module directory (such as Project1/deploy) to the URL attribute element under the mbean named `jboss.deployment:type=DeploymentScanner,flavor=URL`. For example, `deploy/,file://C:/Project1/deploy/`. Be that you add to the existing URL attribute (deploy/).
4. Start the server.

Remote debugging

Within JBuilder, follow these steps:

1. In the project from which you want to launch the remote debug session, click Run|Configurations. If you have not yet created a server type run configuration, click New and select type Server. If you have already created a server type run configuration, select it and choose Edit.
2. Select Debug|Remote in the tree.
3. Check the Enable Remote Debugging option.
4. Enter the host name (the name of the machine on which the server is running.)
5. Specify the correct port number for the remote server.
6. Click OK.
7. Click the down arrow next to the Debug Project icon ( ) on the JBuilder toolbar and select the debug run configuration you just created or edited. You will now be able to set breakpoints in Java code, such as in EJBs, servlets, and so on.

JSP debugging

Tomcat or Jetty do not support JSP source debugging, but you can remedy this problem with the following steps:

1. Download Tomcat 4.0 from `jakarta.apache.org/tomcat`.
2. Browse to the `lib` directory of the Tomcat server install.
3. Search for the `jasper-compiler.jar` and the `jasper-runtime.jar` in your JBoss installation and replace them with the `jasper-compiler.jar` and the `jasper-runtime.jar` found in the Tomcat 4.0 server's `lib` directory.

For more information about JSP debugging, see "Debugging JSPs" in the "JavaServer Pages (JSP)" chapter of the *Developing Web Applications.*

Chapter

10

Using JBuilder with Tomcat

Web Development is a feature of JBuilder Developer and JBuilder Enterprise

Both Java servlets and JavaServer Pages (JSP) run inside web servers. Tomcat, the JavaServer Pages/Java Servlets reference implementation, is included with JBuilder. Although it might differ from your production web server, Tomcat allows you to develop and test your servlets and JSPs within the JBuilder development environment. This chapter explains how to work with Tomcat running inside JBuilder.

When you install JBuilder, two versions of Tomcat, 4.1.30 and 5.0.27, are automatically installed in your JBuilder directory. By default, they are automatically configured. You can examine the configuration with the Configure Servers dialog box (Enterprise|Configure Servers).

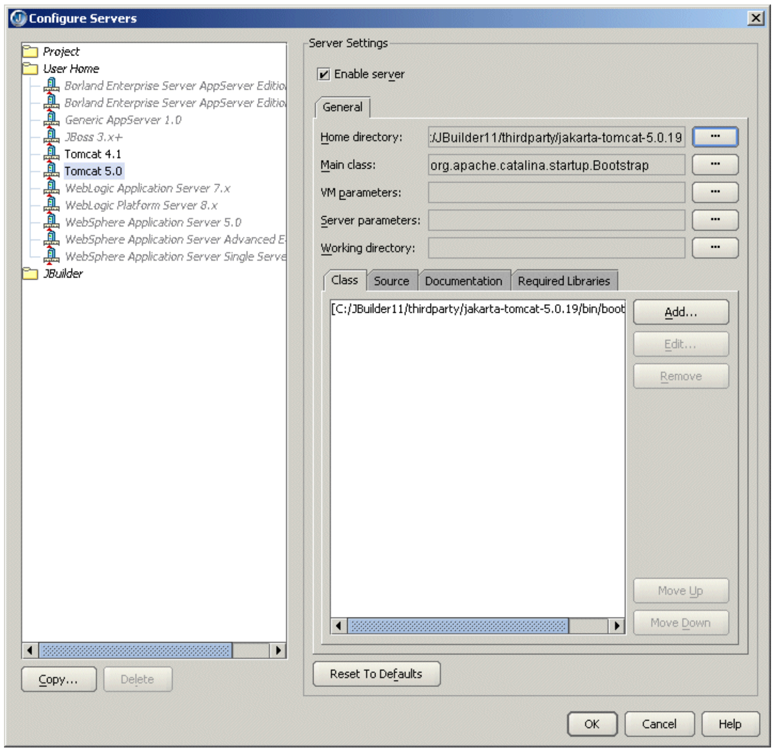
Tomcat 4.is the JDK 1.4 Lightweight Edition. This edition does not contain an XML parser (JDK 1.4 has one built in) and a few other publicly available components. You can download the full edition from `http://jakarta.apache.org/` and use the Configure Servers dialog box to point to it.

In JBuilder Enterprise, Tomcat 5.0 is the default server. To change the default for your project, see "Setting up servers within JBuilder" on page 20. Tomcat 5.0 supports JSR-45, allowing you to debug non-Java source, both locally and remotely. For more information, see "Debugging non-Java source" in *Building Applications with JBuilder.*

Tip If you want to use an earlier version of Tomcat, you can download the binary distribution locally and use the Configure Servers dialog box (Enterprise|Configure Servers) to configure it.

To view Tomcat configurations,

1 Choose Enterprise|Configure Servers. The Configure Servers dialog box is displayed.



Project
User Home
- Borland Enterprise Server AppServer Editio
- Borland Enterprise Server AppServer Editio
- Generic AppServer 1.0
- JBoss 3.x+
- Tomcat 4.1
- Tomcat 5.0
- WebLogic Application Server 7.x
- WebLogic Platform Server 8.x
- WebSphere Application Server 5.0
- WebSphere Application Server Advanced E
- WebSphere Application Server Single Serve

JBuilder

☑ Enable server

Home directory: :/JBuilder11/thirdparty/jakarta-tomcat-5.0.19
Main class: org.apache.catalina.startup.Bootstrap
VM parameters:
Server parameters:
Working directory:

[C:/JBuilder11/thirdparty/jakarta-tomcat-5.0.19/bin/boot

Note In the tree on the left side of the dialog box, entries in gray have not yet been configured. Entries in red are invalid.

2 Choose one of the Tomcat installations from the User Home folder. The Enable Server option at the top of the right side of the dialog box is automatically checked. You do not need to change any settings.

3 Click OK to close the dialog box.

If you have installed Tomcat to another directory and want to run Tomcat from that directory instead of the default, you can change the settings to point to that directory. The following table explains the settings.

Table 10.1 Configure Servers dialog box settings for Tomcat

Option	Description
Home directory	Tomcat's home directory. If you're reconfiguring one of the Tomcat versions installed with JBuilder, this will be in the `<jbuilder>/thirdparty` directory.
Main class	The main class for starting the Tomcat web server.
VM parameters	The parameters to pass to the Java VM running the web server.
Server parameters	The parameters to pass to the web server.
Working directory	The working directory.
Class	The location of Tomcat class files.
Source	The location of Tomcat source files.
Documentation	The location of Tomcat documentation files.
Required Libraries	The libraries that Tomcat requires.

If you'd like more information about Tomcat or would like to run it standalone, refer to the documentation WAR file in the following folder:

- Tomcat 4.1 — `<jbuilder>/thirdparty/jakarta-tomcat-4.1.30/webapps`
- Tomcat 5.0 — `<jbuilder>/thirdparty/jakarta-tomcat-5.0.27/webapps`

Tomcat 4.1 does not support JSP debugging. You can remedy this problem with the following steps:

1. Download the Tomcat 4.0 binary from `http://jakarta.apache.org/`.
2. Browse to the Tomcat 4.0 installation directory. Open the `lib` folder.
3. Copy `jasper-compiler.jar` and `jasper-runtime.jar` to the `<jbuilder>/thirdparty/jakarta-tomcat-4.1.30/common/lib` directory. You will overwrite the two existing files.

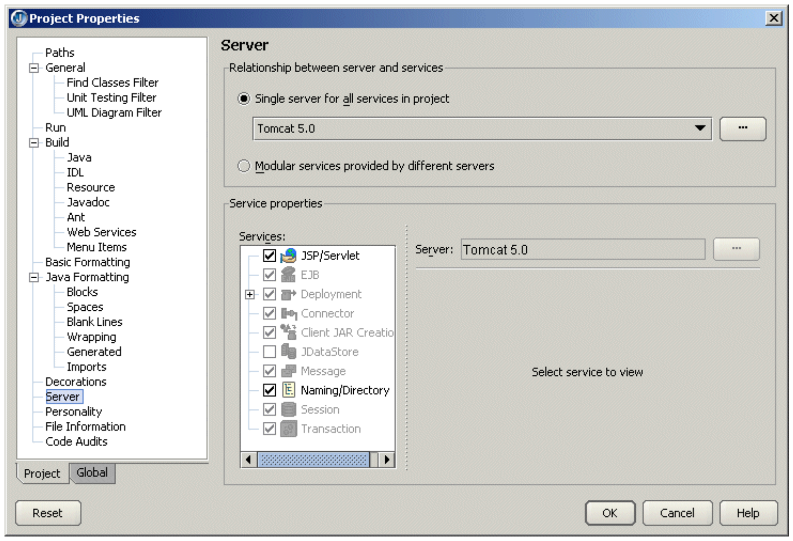
Note The Borland Enterprise Server AppServer uses Tomcat internally for the production web server. If you're using the Borland Enterprise Server, you do not need to configure Tomcat separately.

Selecting Tomcat as your project's web server

Before you begin web development, you should select Tomcat as the web server for your project.

To select the Tomcat web server for your project,

1. Choose Project|Project Properties.
2. Click the Server page. The Server page is displayed.



Server

Relationship between server and services

● Single server for all services in project: Tomcat 5.0

○ Modular services provided by different servers

Service properties

Services:
- ☑ JSP/Servlet
- ☑ EJB
- ☑ Deployment
- ☑ Connector
- ☑ Client JAR Creatio
- ☐ JDataStore
- ☑ Message
- ☑ Naming/Directory
- ☑ Session
- ☑ Transaction

Server: Tomcat 5.0

3. Select the Single Server For All Services In Project option and select the version of Tomcat you want from the drop-down list.
4. If you want to avoid having libraries added to your project that you won't use, uncheck the check box in front of the service(s) you don't need in the Services list. If you disable services, the corresponding JBuilder features will be disabled.

Important Tomcat 4.1 and 5.0 support the JSP/Servlet and Naming/Directory services.

5. If you want to make changes to the configuration settings for the selected version of Tomcat, click the ellipsis (…) button to the right of the server name and edit the settings you want on the General page. Click OK when you're finished.

Running your servlet or JSP with Tomcat

Before you run your web application, you should configure web view options to control how Tomcat is launched in the IDE. For more information, see "Configuring the IDE for web run/debug" in the "Working with web applications in the JBuilder IDE" chapter of *Developing Web Applications.*

Before you web run your servlet or JSP in JBuilder, you also need to create a runtime configuration, set runtime properties for your project, and compile your servlet or JSP. For complete information about these topics, see "Working with web applications in the JBuilder IDE" in *Developing Web Applications*. Once you've completed these tasks, you are ready to run your servlet or JSP.

To web run your servlet or JSP in JBuilder, right-click the servlet or JSP file in the project pane and choose Web Run. The Web Run command runs your configuration in Tomcat without debugging it. If your servlet runs from an HTML or SHTML file, right-click that file and choose Web Run.

If the Web Run or Web Debug command is grayed out on the context menu for a servlet or JSP, check the runtime configuration. To run a web application, you need to create a runtime configuration with the server type selected as the current runner. That server must support the JSP/Servlet service. To create a runtime configuration, see "Creating a runtime configuration" in the "Working with web applications in the JBuilder IDE" chapter of *Developing Web Applications.*

Note Applets cannot be web run or web debugged. This is because applets don't have a URL or a web context to run in. Additionally, applets run in a client browser as opposed to a server. Typically, you run an applet in Sun's AppletViewer or in JBuilder's AppletTestbed. For more information, see "JBuilder's AppletTestbed and Sun's appletviewer" in the "Working with applets" chapter of *Developing Web Applications*.

Starting Tomcat

When you choose Web Run, JBuilder starts Tomcat, using:

- The runtime configuration
- The options set on Tools|Preferences|Web Run|Debug|Optimize

Messages are logged to the web server tab displayed in the message pane. HTTP commands and parameter values are also echoed to this pane. The name of the tab will reflect the name of the web server, for example "Tomcat" for the Tomcat web server.

Stopping Tomcat

To stop Tomcat, click the Reset Program button 🟥 on the web server tab. To start the web server again and re-run your web application, click the Restart Program button ▶ on the tab. You'll usually follow these steps when you make changes to source code, re-compile, and re-run. You don't need to close the web server pane each time you start the web server.

Note The first time you press the Reset Program button, it simply sends a command to the server to shutdown. In particular, if you are trying to debug your web application's lifecycle event handlers, this would call the `Servlet.destroy()` and `ServletContextListener.contextDestroyed()` methods. If you're not debugging, the server will usually shutdown by itself in a few seconds. If you press the Reset Button a second time, the server process is terminated immediately.

Changing Tomcat's port number

Occasionally, you may run into a problem where the default port number assigned to your web server (typically 8080) is in use by another application. You can reassign the port number, or simply instruct JBuilder to search for an unused port. (In a typical installation, this is set as the default.)

To change Tomcat's port number,

1. Choose Run|Configurations and select the runtime configuration for your web application. Choose Edit.
2. Choose the JSP/Servlet Service in the tree on the lower left side of the Edit Runtime Configuration dialog box.
3. Enter the port number the web server should listen to in the Port Number field.
4. Choose the Search For Unused Port option to tell JBuilder to choose another port if the specified one is in use. (The port is only searched for the first time a web run is requested.) It is useful to select this option when you are running more than one servlet or JSP, as otherwise you might get a message that the port is busy. It is also useful to check this option in the event that a user problem brings the web server down. With this option selected, you will be protected if the web server is not shut down properly. This option works in conjunction with the Web View options on the Web page of the Preferences dialog box when the specified port is in use by a non-web process.

Creating a custom server.xml file with Tomcat

Typically, when you run your web application using Tomcat in the JBuilder IDE, JBuilder creates a `serverXXXX.xml` file (where XXXX is the port number Tomcat is listening to) and deletes it when you shut down Tomcat. You can edit the file and force JBuilder to keep it.

To create a custom `server.xml` file,

1. Run your web application in JBuilder as you normally would.
2. Go to your project's `Tomcat\conf` directory while Tomcat is running. This folder is created by JBuilder when your web application is running.
3. Make a copy of the `conf` directory in a `temp` directory.
4. Open `serverXXXX.xml` in the `conf` directory in a text editor.
5. Remove the second line of the file:

   ```
   <!--This comment marks this file as generated, so it may be deleted and
   regenerated at any time. To preserve manual changes to this file, delete
   this comment.-->
   ```

   With this line removed, JBuilder will not automatically delete the file when you stop the server.
6. Make other changes to the server configuration file as needed. (Save the file and leave it open in the editor.)
7. Go back to JBuilder and shut down Tomcat.
8. Save the edited file to the copy of the `conf` directory.
9. Copy the `conf` directory back to your project's Tomcat directory.
10. Restart the web server and your web application.

Now, when you re-start the web application and the web server, your edited `serverXXXX.xml` file will be used instead of an auto-generated one.

Debugging JSPs with Tomcat 4.1

Tomcat 4.1 does not support JSP debugging. You can remedy this problem with the following steps:

1. Download the Tomcat 4.0 binary from `http://jakarta.apache.org/`.
2. Browse to the Tomcat 4.0 installation directory. Open the `lib` folder.
3. Copy `jasper-compiler.jar` and `jasper-runtime.jar` to the `<jbuilder>/thirdparty/jakarta-tomcat-4.1.30/common/lib` directory. You will overwrite the two existing files.

Chapter

11

Editing J2EE deployment descriptors

JBuilder has a Deployment Descriptor Editor, also known as the DD Editor, you can use to edit the deployment descriptors of various types of J2EE modules. This chapter introduces you to the DD Editor and gives you an overview of how to use it.

The DD Editor can edit these types of J2EE modules:

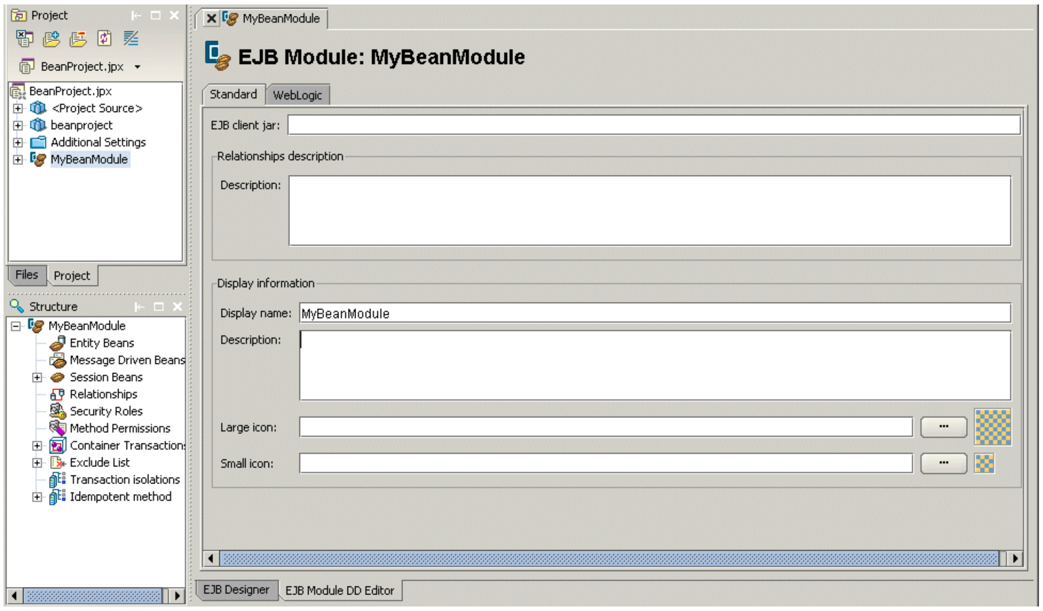
- Web — A Web module contains all the compiled source files, resources, and deployment descriptors needed to deploy a web application. You can create web modules using JBuilder's web wizards and tools. For complete information about web modules, see *Developing Web Applications.*
- EJB — An EJB module contains all the compiled source files, resources, and deployment descriptors needed to deploy one or more enterprise beans. You can create enterprise beans using JBuilder's EJB wizards and tools. For complete information about creating EJB modules and enterprise beans, see “Creating session beans with the EJB designer”, “Creating beans with the Enterprise JavaBean 1.x wizard”, “Creating entity beans with the EJB designer”, and “Creating EJB 1.x entity beans from an existing database table.” All these chapters are in the *Developing Applications with Enterprise JavaBeans* book.
- Application client — An application client module contains all the resources and deployment descriptors for a client application that references one or more enterprise beans. Create the module using the File|New|Enterprise|Application Client Module wizard. For more specific information about creating application client modules and editing the deployment descriptors, see “Creating an application client module” in the “Developing enterprise bean clients” chapter of *Developing Applications with Enterprise JavaBeans*.
- Connector — A connector module contains the resource adapter and deployment descriptors that connect the selected application server with an existing enterprise information system. Create a connector module using the File|New|Enterprise| Connector Module wizard. For more specific information about creating connector

modules and editing the deployment descriptors, see Chapter 12, "Integrating with Enterprise Information Systems."

- Application — An application module can contain any of the other types of J2EE modules and wraps them into a single application and includes its own deployment descriptor entries. Create an application module using the File|New|Enterprise| Application Module wizard. For more specific information about creating application modules and editing the deployment descriptors, see "Creating an application module" in the "Deploying enterprise beans" chapter of *Developing Applications with Enterprise JavaBeans*.
- JBoss service — A JBoss service module contains the definition of a service. Once you compile the service module, a service archive (SAR) is created, ready to be deployed to JBoss. For information about creating a JBoss service module, see "Creating a JBoss service module" on page 84.

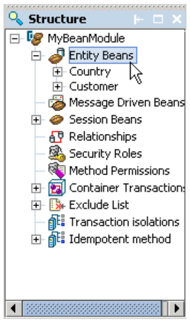
Displaying the DD Editor

After you have created a J2EE module, display the DD Editor by double clicking the new module in the project pane. If the module is an EJB module, you must then click the EJB DD Editor tab in the content pane of the browser. You'll also see nodes appear in the structure pane. As an example, here is how the browser appears when a EJB module has been double-clicked, the EJB DD Editor page in the content pane selected, and the BEA WebLogic Platform Server 8.x is the selected server for the project:



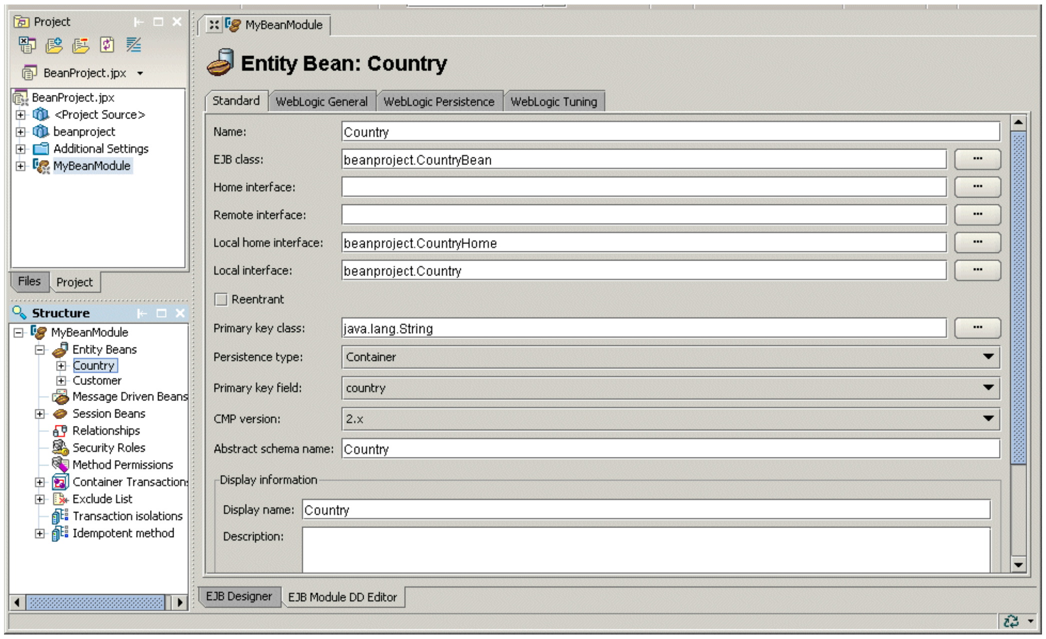
Display name: MyBeanModule

Some of the nodes in the structure pane are expandable as they contain data already. In this particular EJB module, two entity beans were created using the EJB designer and added to the `MyEJBModule` module. As a result, the Entity Beans node is expandable. When this node is expanded, you can see the entity beans the module contains:



Standard and server-specific pages

Usually you will see a Standard page for each node in the structure pane that can be expanded. This Standard page contains fields that apply to all servers. Often you will also see one or more tabs that apply to a specific server. Click the server-specific tab to see the page. The pages available depend on which server you selected for the project. In this example, there are three additional pages available for a WebLogic entity bean as WebLogic is the selected server:



Entity Bean: Country

Name:	Country
EJB class:	beanproject.CountryBean
Home interface:	
Remote interface:	
Local home interface:	beanproject.CountryHome
Local interface:	beanproject.Country
Reentrant	☐
Primary key class:	java.lang.String
Persistence type:	Container
Primary key field:	country
CMP version:	2.x
Abstract schema name:	Country

Display information

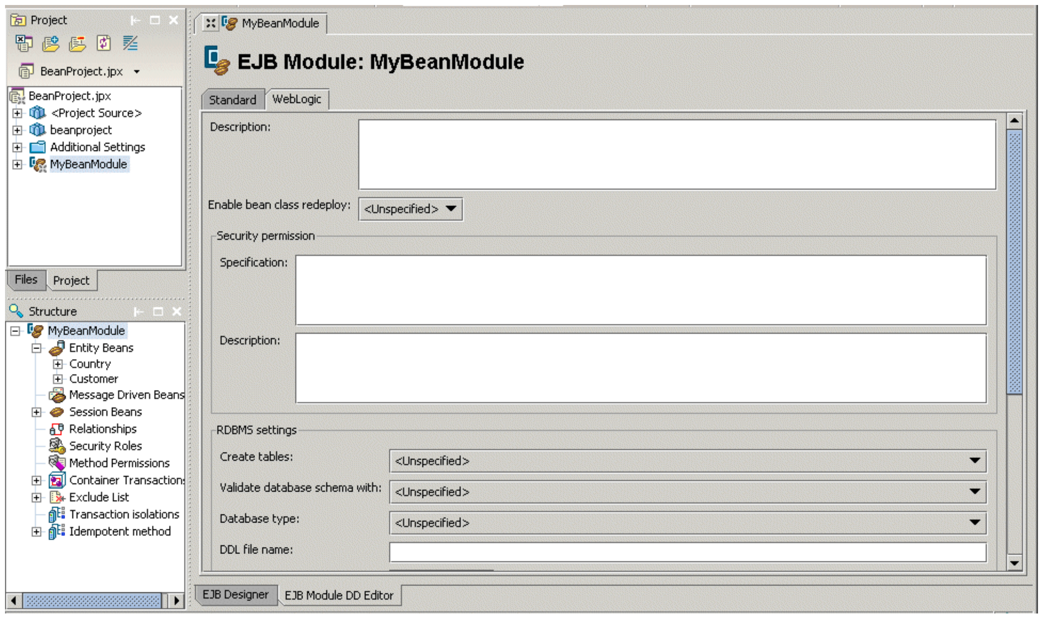
Display name:	Country
Description:	

Tip To see information about a page in the DD Editor, press *F1*.

Fill in each Standard page with the information requested. You can find more specific information about filling in these fields in other parts of the documentation that discuss the various types of J2EE modules:

- EJB modules
- Web modules
- Application client modules
- Connector modules
- Application modules

Click any server-specific tabs and fill in each of these pages also. Whether there are server-specific pages available depends on the type of node you are editing and on your selected server. Here you can see a WebLogic-specific page selected for an EJB module:



Enable bean class redeploy: <Unspecified>

Create tables: <Unspecified>

Validate database schema with: <Unspecified>

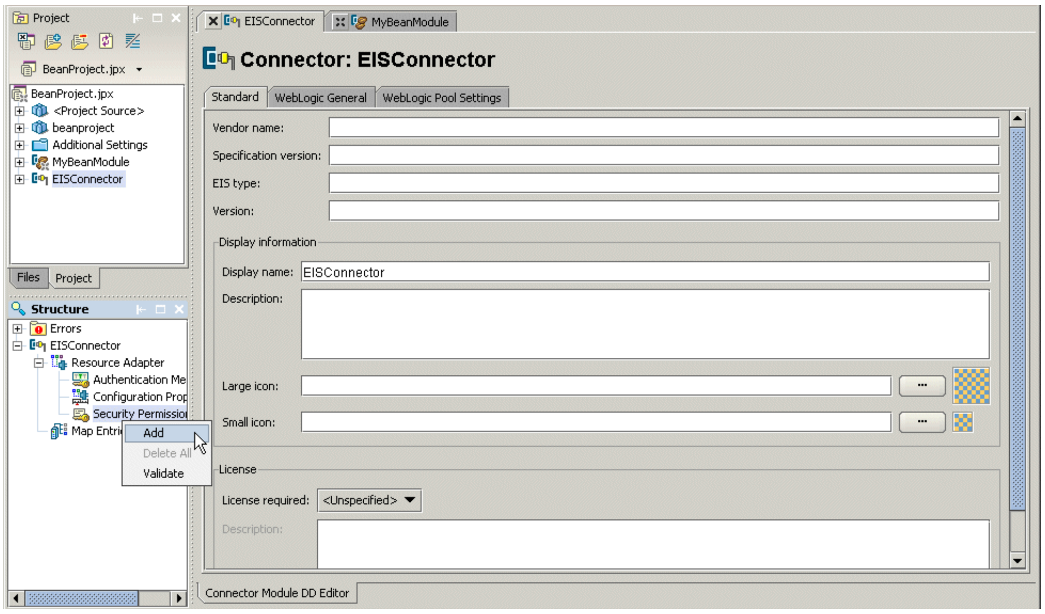
Database type: <Unspecified>

You may have noticed already the nodes in the structure pane that are marked with the server-specific icon ( ). These too are specific to a particular server. In this particular case, that server is BEA WebLogic Server.

Adding to and deleting from a deployment descriptor

To add descriptor information to a module,

- Right-click the type of node for which you want to add information to the descriptor in the structure pane and choose Add.



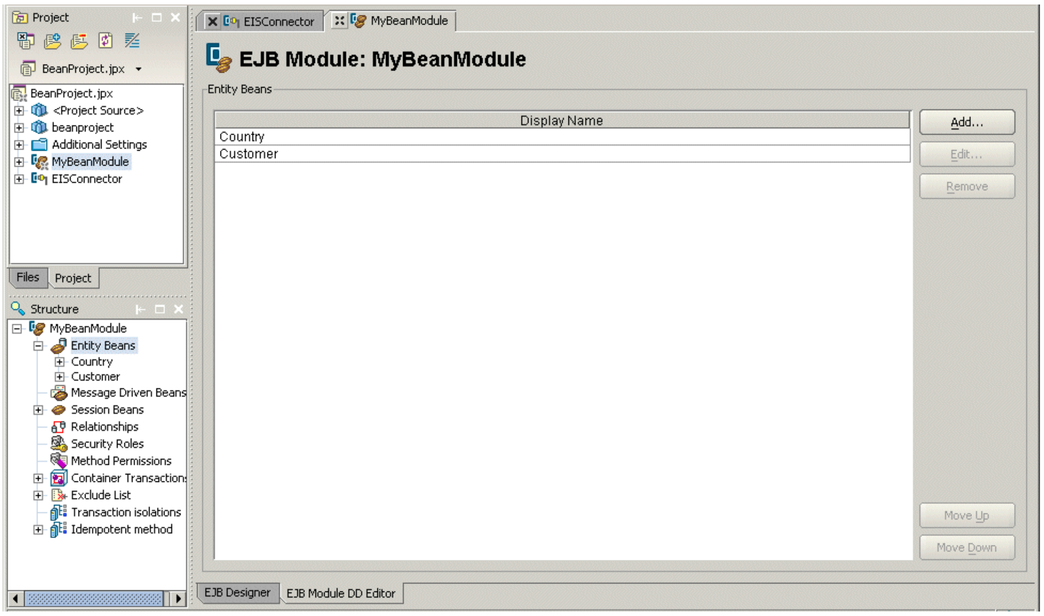
Display name: EISConnector

License required: <Unspecified>

A new page appears in the content pane that corresponds to the type of node you selected. The sample image shows a security permission being added to a connector module.

- Fill in the information requested on this page and new node for that page appears under the node you selected in the structure pane.

Double-clicking a top-level category node, such as Entity Beans in an EJB module, displays a page with a table containing all existing entries of that type. For example, if an EJB module contains two entity beans, double-clicking the Entity Beans node displays a table containing an entry for each entity bean. You can also use this page to add and delete entries or reorder the list with the Move Up and Move Down buttons.



To add an entry, click the Add button, then select the new entry in the table and double-click it. The appropriate page appears. Fill in the required information.

There are two ways to delete an entry from a category:

- Right-click the node you want to delete in the structure pane and choose Delete.
- Double-click the top-level category node. On the page that appears, select the entry from the table you want to delete and click the page's Remove button.

Validating a deployment descriptor entry

You can help ensure the deployment descriptor information that is common to all servers is entered correctly by using the DD Editor's validator. The validator checks that the deployment descriptor conforms to the DTD specification and also makes other checks such as insuring proper naming conventions are used.

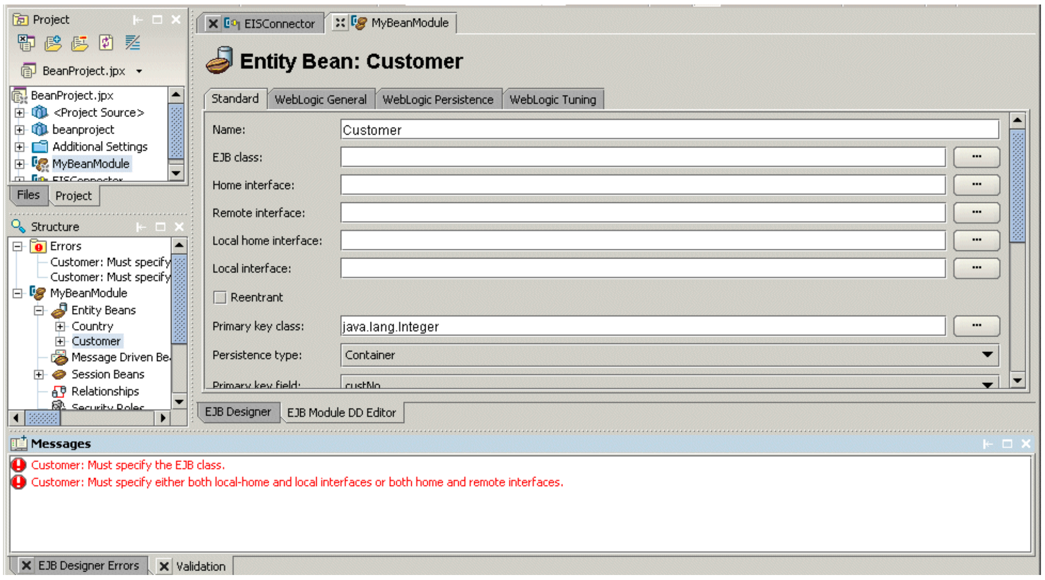
To validate a deployment descriptor,

1 Right-click the node in the structure pane you want to validate.

2 Choose Validate on the context menu that appears.

Note Server-specific data is not checked. Only data found on Standard pages is validated.

If errors or warnings exist, they will appear on a Validation page in the message pane:



Name: Customer
EJB class:
Home interface:
Remote interface:
Local home interface:
Local interface:
Reentrant
Primary key class: java.lang.Integer
Persistence type: Container

Customer: Must specify the EJB class.
Customer: Must specify either both local-home and local interfaces or both home and remote interfaces.

Click an error message and the page on which the error occurs appears in the DD Editor.

There is also an Errors folder that appears in the structure pane when the deployment descriptor has errors. The Errors folder appears only for the Standard DD editor pages. Open the Errors folder to see the error messages.

The validator is invoked as part of the build process. Output from the validator is merged with the build output. You can turn this default behavior off:

1. Right-click the module and choose Properties.
2. Select the Build node.
3. Uncheck the Validate Deployment Descriptors During Build option.
4. Choose OK.

Chapter

12

Integrating with Enterprise Information Systems

The J2EE Connector architecture addresses the problem of integrating Java enterprise applications with existing Enterprise Information Systems (EIS). An EIS can be a system such as ERP, CRM, and supply chain management applications and database systems. Previously, if an enterprise application had to access information in an EIS, some one had to build a custom connection between the application server and the EIS.

Now the J2EE Connector architecture defines a uniform way to integrate J2EE application servers with existing information systems. Each EIS vendor creates a resource adapter using this architecture. A resource adapter is much like a JDBC driver, as both provide a standard API through which an application can access a resource that is outside the J2EE server. This resource adapter plugs into a J2EE compliant application server. As long as the resource adapter follows the Connector specification, it provides a scalable, secure, and transactional connection to an application server.

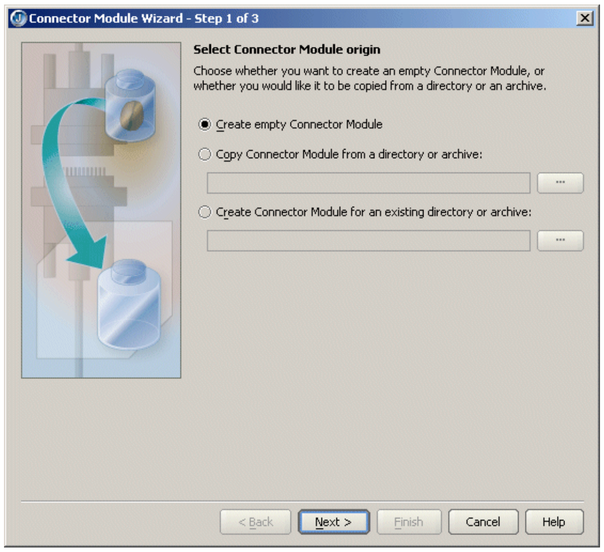
To use a resource adapter for an application server, you can use JBuilder's Connector Module wizard, which creates the deployment descriptors for the resource adapter. Then you use the Connector Module DD Editor to provide the information the resource adapter needs to make the connection to the EIS.

Creating a connector module node in the project pane

To create a connector module, you use JBuilder's Connector Module wizard. Besides creating a new empty connector module, the Connector Module wizard also gives you the option to copy an existing connector module to your project instead of creating a new one. You can also use the wizard to create a module node in the project pane that represents an existing module directory outside your project. Or you can use it to create a module node that represents an archive (.rar) outside your project; you can view the archive using the Connector Module DD Editor, but you won't be able to edit it.

To begin creating a connector module node that contains a resource adapter, follow these steps:

1 Choose File|New|Enterprise and double-click the Connector Module (RAR) wizard.



Connector Module Wizard - Step 1 of 3

Select Connector Module origin

Choose whether you want to create an empty Connector Module, or whether you would like it to be copied from a directory or an archive.

(●) Create empty Connector Module

( ) Copy Connector Module from a directory or archive:

( ) Create Connector Module for an existing directory or archive:

2 Decide whether you are creating an empty connector module that you will fill in later, if you are copying an existing connector module from a directory or archive, if you are creating a module node that represents a connector module in a directory outside your project, or if you are creating a module node that represents an archive (.rar).

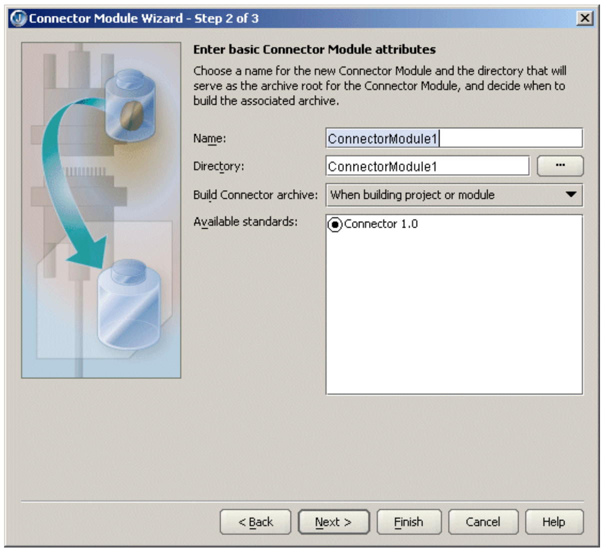
Because your selection determines the behavior of the wizard, the next sections describe how to use the wizard depending on the task you want to accomplish.

Creating an empty connector module

If you want to create a new empty connector module,

1 Select the Create Empty Connector Module option on the first page of the Connector Module wizard.

2 Click Next to see Step 2:



Connector Module Wizard - Step 2 of 3

Enter basic Connector Module attributes

Choose a name for the new Connector Module and the directory that will serve as the archive root for the Connector Module, and decide when to build the associated archive.

Name: ConnectorModule1

Directory: ConnectorModule1

Build Connector archive: When building project or module

Available standards: (●) Connector 1.0

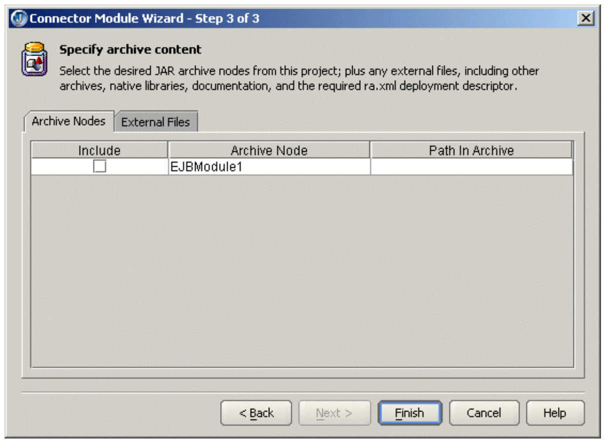
3 Specify a name to identify the new module in the Name field. The default will be the name of the selected directory.

4 Specify a directory name for the Directory field. This is the directory that will contain the module. By default, the wizard uses the same name for the directory as the name you specified in the Name field. You can use the ellipsis (…) button to navigate to a directory you want to use instead, if you choose.

5 From the Build Connector Archive drop-down list, select when you want the connector archive to be built.

6 Select which Connector specification you are using from the options in the Available Standards box.

7 Click Next to go to Step 3:


Connector Module Wizard - Step 3 of 3

Specify archive content

Select the desired JAR archive nodes from this project; plus any external files, including other archives, native libraries, documentation, and the required ra.xml deployment descriptor.

Include	Archive Node	Path In Archive
☐	EJBModule1	

8 Check the Include box of the archives found in your current project that you want to become part of the Connector module. The wizard lists all archives it finds in your current project.

9 If you want files that are outside modules in your project included, click the External Files tab, click the Add button to display the Select One Or More Files dialog box, and use it to select the files you want to add. Continue adding files until all the files you want added to the connector module are added. Click OK.

10 Choose Finish.

Copying an existing connector module

You can use an existing connector module by copying it to your current project with the Connector Module wizard:

1 Select the Copy Connector Module From A Directory Or Archive option on the first page of the Connection Module wizard.

2 Use the ellipsis (…) button to browse to the location of the directory or archive that contains the module or archive you want to copy to the your project.

3 Click Next.

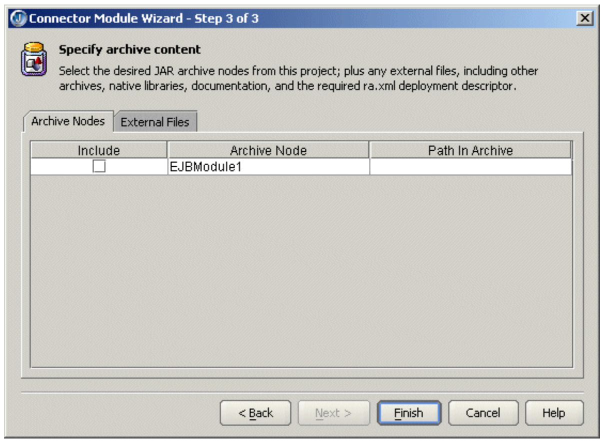
4 Specify a name to identify the new module in the Name field.

5 Specify a directory name for the Directory field. By default, the wizard uses the same name for the directory as the name you specified in the Name field. You can use the ellipsis (…) button to navigate to a directory you want to use instead, if you choose.

6 From the Build Connector Archive drop-down list, select when you want the connector archive to be built.

7 Select which Connector specification you are using from the options in the Available Standards box.

8 Click Next to go to Step 3:


Connector Module Wizard - Step 3 of 3
Specify archive content
Select the desired JAR archive nodes from this project; plus any external files, including other archives, native libraries, documentation, and the required ra.xml deployment descriptor.
Archive Nodes
External Files
< Back
Next >
Finish
Cancel
Help

Include	Archive Node	Path In Archive
☐	EJBModule1	

9 Check the Include box of the archives found in your current project that you want to become part of the Connector module. The wizard lists all archives it finds in your current project.

10 If you want files that are outside modules in your project included, click the External Files tab, click the Add button to display the Select One Or More Files dialog box, and use it to select the files you want to add. Continue adding files until all the files you want added to the connector module are added.

11 Choose Finish.

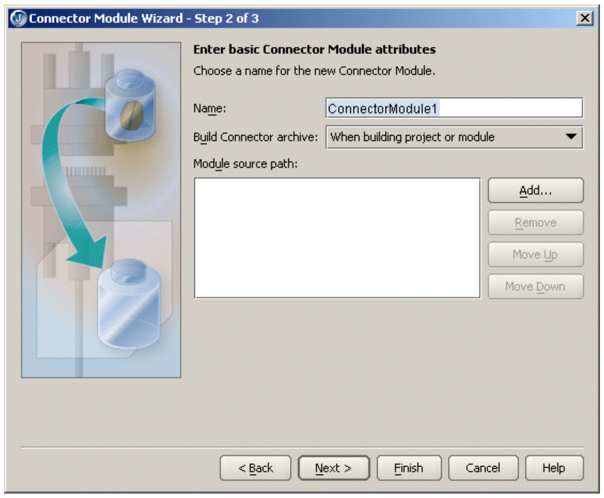
Creating a module node for an existing module in a directory outside your project

You can use an existing module in a directory outside your project; you don't have to copy it to your current project:

1 Select the Create A Connector Module For An Existing Directory Or Archive on the first page of the Connector Module DD Editor.

2 Use the ellipsis (…) button to browse to the location of the directory that contains the module.

3 Click Next.


Connector Module Wizard - Step 2 of 3
Enter basic Connector Module attributes
Choose a name for the new Connector Module.
Add...
Remove
Move Up
Move Down
< Back
Next >
Finish
Cancel
Help

Name: ConnectorModule1
Build Connector archive: When building project or module
Module source path:

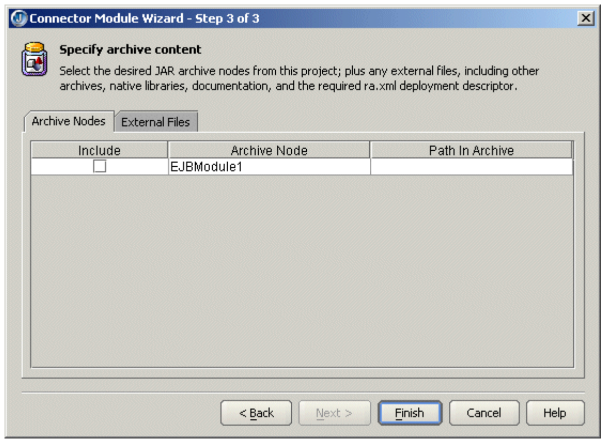
4 Specify a name to identify the new module in the Name field.

5 From the Build Connector Archive drop-down list, select when you want the connector archive to be built.

6 Use the Add button to add the source path for any Java files associated with the module.

You might want to add Java source paths that are necessary to build a module. It isn't absolutely necessary to add the source paths in this wizard, however. You can also add additional source path directories to your project directly. If you choose to add the source path directories using the Connector Module wizard, the directories you specify will be added to your project source path.

7 Click Next to go to Step 3:


Connector Module Wizard - Step 3 of 3

Specify archive content

Select the desired JAR archive nodes from this project; plus any external files, including other archives, native libraries, documentation, and the required ra.xml deployment descriptor.

Include	Archive Node	Path In Archive
☐	EJBModule1	

8 Check the Include box of the archives found in your current project that you want to become part of the Connector module. The wizard lists all archives it finds in your current project.

9 If you want files that are outside modules in your project included, click the External Files tab, click the Add button to display the Select One Or More Files dialog box, and use it to select the files you want to add. Continue adding files until all the files you want added to the connector module are added.

10 Choose Finish.

Creating a connector module node for an existing archive (RAR) outside your project

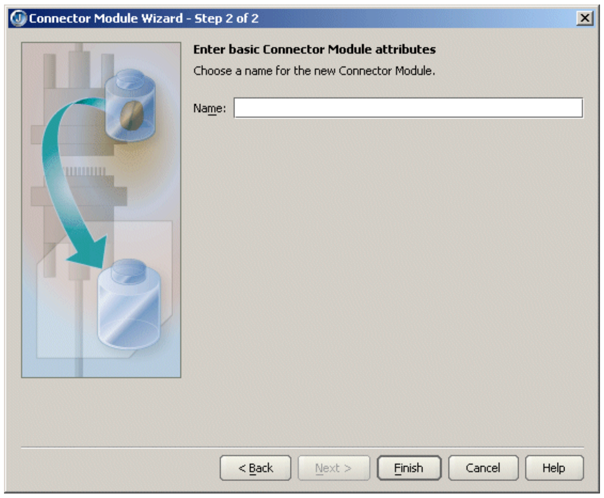
You can create a connector module node for a resource adapter that is outside your current project. Although you will be able to view the contents of the archive using the Connector Module DD Editor, you won't be able to edit it. The module node will be read only. You also won't be able to build it, but you will be able to deploy it.

To create a connector module node for a resource adapter outside your project,

1 Select the Create A Connector Module For An Existing Directory Or Archive.

2 Use the ellipsis (…) button to browse to the location of the resource adapter.

3 Click Next to go to the next page:


Connector Module Wizard - Step 2 of 2
Enter basic Connector Module attributes
Choose a name for the new Connector Module.
Name:
< Back
Next >
Finish
Cancel
Help

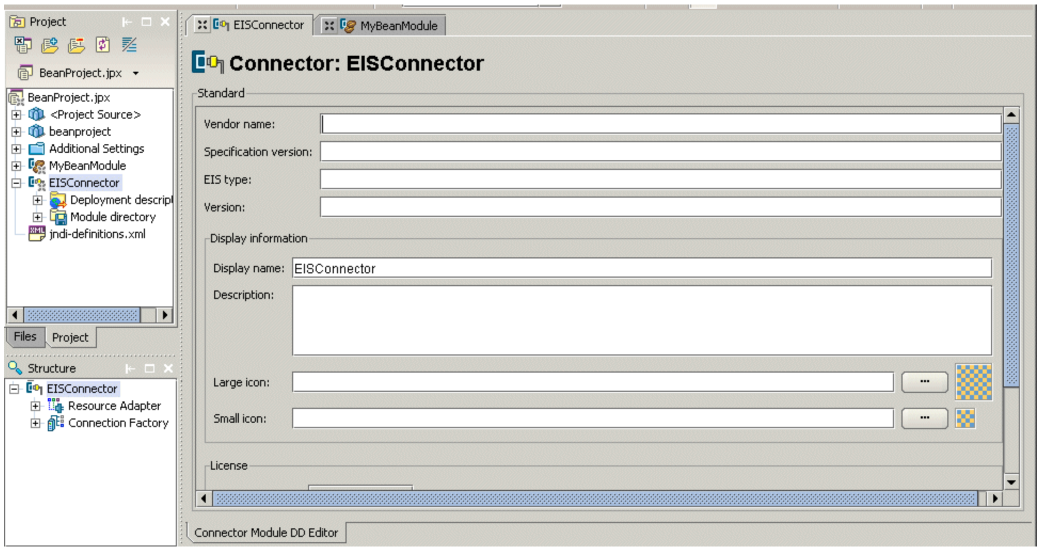
4 Specify the name you want to use as the name of the connector for the Name field. The default name will be the name of the archive with the extension as a separate word at the end, such as `Connector1 RAR`.

5 Choose Finish.

Viewing the connector module deployment descriptors

No matter which method you used to create a new connector module node, you'll see it appear in the project pane when you finish using the Connector Module wizard.

To view or edit the connector module deployment descriptor(s), double-click the connector module node in the project pane. (You won't be able to edit an archive located outside your current project.) The Connector Module DD Editor appears:



Display name: EISConnector

Fill in the fields on the Connector page in the Connector DD Editor. These are the fields:

- **Vendor Name:** Enter the name of the EIS vendor.
- **Specification Version:** Specify the Connector specification version being used.
- **EIS Type:** The type of EIS, which helps identify which EIS instances the connector can be used with.

- **Resource Adapter Version:** Specify the version of the resource adapter being used.
- **Language**: Select the language you are using from the drop-down list.
- **Display Name:** Specify a short name that can be displayed in a GUI.
- **Description:** Use this field to include any information that the component file producer wants to provide to the deployer. This information is optional.
- **Large Icon:** Specify a JPEG or GIF file containing a small image (16x16 pixels) to be used in a GUI. This information is optional.
- **Small Icon:** Specify a JPEG or GIF file containing a small image (32x32 pixels) to be used in a GUI. This information is optional.
- **License Required:** Use the drop-down list to specify whether a license is required or not.
- **Description:** If a license is required, this field specifies the licensing requirements for the resource adapter module. For example, you might include the duration of license, number of connection restrictions, and so on. This information is optional.

Editing the resource adapter descriptors

By looking in the structure pane, you'll see that your connector module contains a resource adapter node. Other nodes that you may see are server-specific nodes, so which nodes appear depends on your server.

Note You won't be able to edit an archive that is located outside your current project.

You must supply the information requested on the Resource Adapter page. Fill in the following fields:

- **Managed Connection Factory Class:** Specify the class that implements the `ManagedConnectionFactory` interface, which either matches an existing connection to the EIS with the incoming request or creates a new physical connection to the EIS. When the application server needs a connection to the EIS, it asks this class to retrieve an existing connection or create a new one.
- **Connection Factory Interface:** Specify the `ConnectionFactory` interface, which allows an application component to connect to an EIS instance.
- **Connection Factory Implementation Class:** Specify the class that implements the `ConnectionFactory` interface. This class will contain a `getConnection()` method that requests the application server to allocate a connection using the `ConnectionManager.allocateConnection()` method.
- **Connection Interface:** Specify the `Connection` interface, which connects an application with the EIS.
- **Connection Implementation Class:** Specify the class that implements the Connection interface. This class must include a `close()` method so the application can terminate a connection with the EIS.
- **Transaction Support:** Use the drop-down list to specify whether the connection will have No Transaction Support, Local Transaction Support, or XATransaction Support. Local transactions are managed internally by the EIS. XA transactions are managed by a transaction manager external to the EIS.
- **Supports Reauthentication:** Check this field if you want the resource adapter to support reauthentication.

For more information about resource adapters and the J2EE Connector Architecture, see these articles on the web:

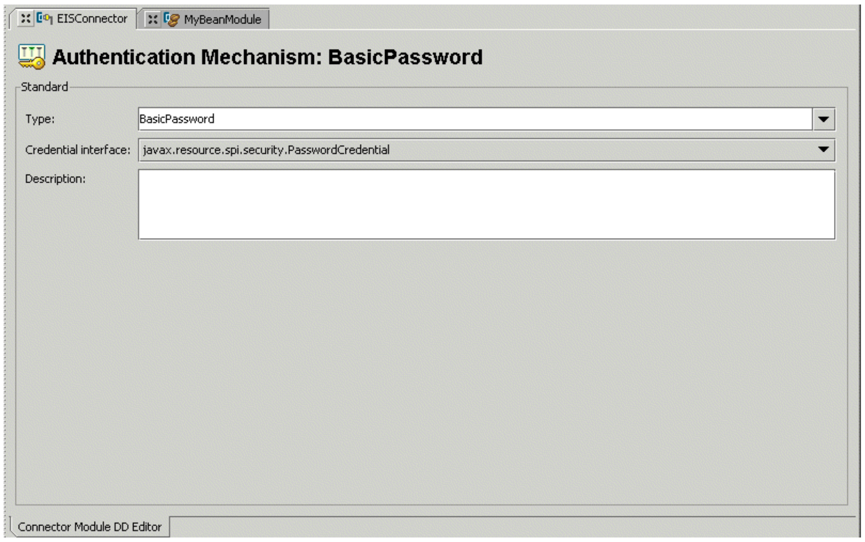
- "The Java 2, Enterprise Edition (J2EE) Connector Architecture's Resource Adapter" at `http://developer.java.sun.com/developer/technicalArticles/J2EE/connectorclient/resourceadapter.html`
- "J2EE Connector Architecture" at `http://java.sun.com/j2ee/connector/`.
- "J2EE Connector Architecture" at `http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Connector.html`, which is part of Sun's J2EE Tutorial.

Expand the resource adapter node in the structure pane to see the subnodes. Which subnodes appear depends on which version of the Connector standard you are using. To add a new node of any of these types, right-click the node and choose Add.

Authentication Mechanism page

Connector modules have an Authentication Mechanism node in the structure pane. Expand the Resource Adapter node to see it.

If you add a Authentication Mechanism node, it looks like this:



Authentication Mechanism: BasicPassword

Standard

Type: BasicPassword

Credential interface: javax.resource.spi.security.PasswordCredential

Description:

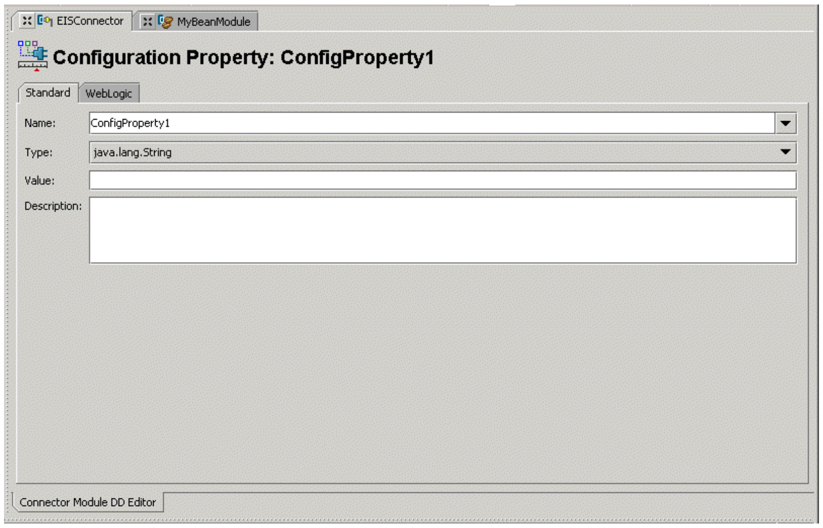
These are the fields on this page:

- **Type:** Use the drop-down list to specify the type as BasicPassword or Kerbv5, meaning a Kerboros version 5 authentication mechanism.
- **Credential Interface:** Use the drop-down list to specify the interface. If you chose the BasicPassword type, select the `PasswordCredential` interface, which encapsulates a user name and password. If you chose the Kerbv5 type, select the `GenericCredential` interface, which defines a way to access the security credentials of a Principal.
- **Description:** Use the optional Description field to specify any resource adapter specific requirement for the support of security contract and authentication mechanism.

Configuration Property page

Connector modules have a Configuration Property node in the structure pane. Expand the Resource Adapter node to see it.

If you add a Configuration Property page, it looks like this:


EISConnector
MyBeanModule
Configuration Property: ConfigProperty1
Standard
WebLogic
Connector Module DD Editor

Name: ConfigProperty1
Type: java.lang.String
Value:
Description:

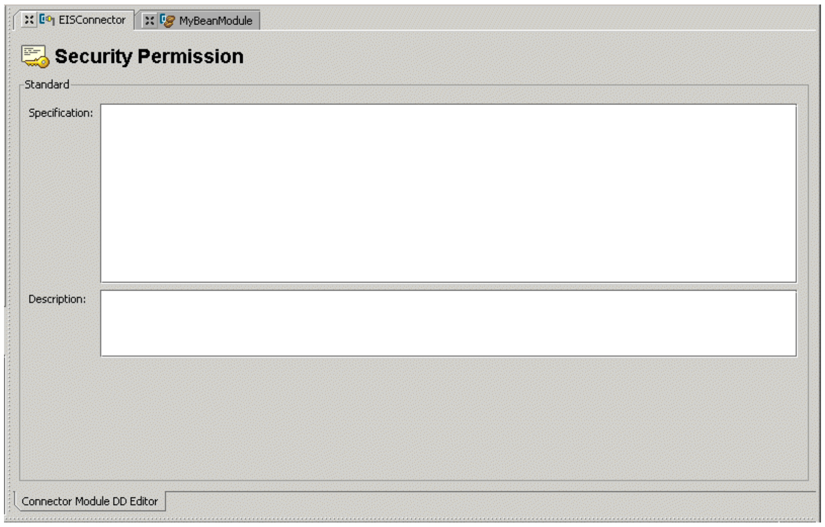
These are the fields on this page:

- **Name:** Specify the name of the configuration property.
- **Type:** Specify the type of the configuration property by using the drop-down list.
- **Value:** Specify the value of the configuration property.
- **Description:** Describe the configuration property.

Security Permission page

Connector modules have a Security Permission node in the structure pane. Expand the Resource Adapter node to see it.

If you add a Security Permission page, it looks like this:



Specification:
Description:

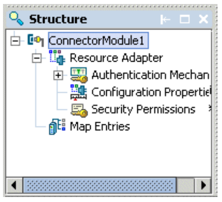
These are the fields on this page:

- **Specification:** Specifies a security permission that is required by the resource adapter code.
- **Description:** An optional description.

Server-specific pages

As you work with pages in the Connector Module DD Editor, you may find server-specific pages. To get help for these pages, press *F1*. Your server documentation should also be a good source of information about the type of data your server requires.

Some servers have a server-specific node in a connector module that is not a part of the resource adapter. For example, here you see a Map Entries node in the structure pane, which appears if your selected server is WebLogic 8.x:



Again, if you need help with the pages linked to a server-specific node, press *F1* and also consult your server's documentation.

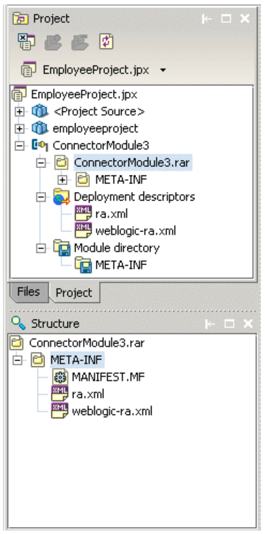
Compiling the connector module

When you successfully compile a connector module, a RAR (resource archive) is created.

To compile a connector module, right-click the module in the project pane and choose Make.

Note You will not be able to make or build a read-only module.

To see the resulting RAR, expand the connector module node in the project pane:



You can also expand the RAR node to see the RAR's deployment descriptors. Double-click a deployment descriptor to see it in its XML format. If there are errors in the resulting XML file, the Errors folder in the structure pane will tell you what the problems are.

Chapter

13

Building J2EE modules

This chapter presents specific information about packaging J2EE modules in JBuilder. It explains about setting module build properties, including the use of custom filters and file types. It also provides a few common build scenarios and tips for improving build performance.

J2EE modules

You can create J2EE modules in JBuilder using the wizards available in the object gallery (File|New). JBuilder recognizes the following types of J2EE modules:

- Web — A Web module contains all the compiled source files, resources, and deployment descriptors needed to deploy a web application. You can create web modules using JBuilder's web wizards and tools. Once you compile the web module, a WAR file is created. For complete information about web modules, see *Developing Web Applications.*
- EJB — An EJB module contains all the compiled source files, resources, and deployment descriptors needed to deploy one or more enterprise beans. You can create enterprise beans using JBuilder's EJB wizards and tools. Once you compile the EJB module, a JAR file is created. For complete information about creating EJB modules and enterprise beans, see "Creating session beans with the EJB designer", "Creating beans with the Enterprise JavaBean 1.x wizard", "Creating entity beans with the EJB designer", and "Creating EJB 1.x entity beans from an existing database table." All these chapters are in the *Developing Applications with Enterprise JavaBeans* book.
- Application client — An application client module contains all the resources and deployment descriptors for a client application that references one or more enterprise beans. Create the module using the File|New|Enterprise|Application Client Module wizard. Compiling an application client module creates a JAR file. For more specific information about creating application client modules and editing the deployment descriptors, see "Creating an application client module" in the "Developing enterprise bean clients" chapter of *Developing Applications with Enterprise JavaBeans*.
- Connector — A connector module contains the resource adapter and deployment descriptors that connect the selected application server with an existing enterprise information system. Create a connector module using the File|New|Enterprise|

Connector Module wizard. Compiling a connector module creates a RAR (Resource Archive) file. For more specific information about creating connector modules and editing the deployment descriptors, see Chapter 12, "Integrating with Enterprise Information Systems."

- Application — An application module can contain any of the other types of J2EE modules and wraps them into a single application and includes its own deployment descriptor entries. Create an application module using the File|New|Enterprise| Application Module wizard. Compiling an application module creates an EAR (Enterprise Archive) file. For more specific information about creating application modules and editing the deployment descriptors, see "Creating an application module" in the "Deploying enterprise beans" chapter of *Developing Applications with Enterprise JavaBeans*.
- JBoss service — A JBoss service module contains the definition of a service. Once you compile the service module, a service archive (SAR) is created, ready to be deployed to JBoss. For information about creating a JBoss service module, see "Creating a JBoss service module" on page 84

Setting module build properties

The content of an archive is controlled by a combination of node level properties (filters for file types and classes and dependencies). The content of the archive will match the content of the module directory. To begin editing the properties of a module, right-click the module in the project pane and choose Properties on the context menu.

File type filters

In the Properties dialog box for a module, the Content page contains a list of include file type filters that control which file types are copied over from the project's output path to the module directory before the archive is created. The file types listed here match the global file types recognized by JBuilder. (See Tools|Preferences|Browser|File Types.) Each module has a list of pre-selected file type filters depending on the type of module. For example, the EAR module has only the Archive and the XML file type selected by default.

The file types therefore work in combination with the include and exclude filters. For example, if you add a recursive exclude filter for a certain file type, it will override the file type filter selection.

Class and resource filters

You can customize archive content using custom filters. Add filters on the Content page of the Properties dialog box for a module. The following default rules apply for the different module types:

- EJB JAR content is by default controlled by the classes defined in the module descriptors and dependent classes. You must add any other classes using custom filters. The Java Class file type is included by default. To turn off this behavior, uncheck the option Only Include Module Specific Java Classes and add custom filters.
- Web JAR content is by default controlled by the classes defined in the module descriptors and dependent classes. You must add any other classes using custom filters. The Java Class file type is included by default. To turn off this behavior, uncheck the option Only Include Module Specific Java Classes and add custom filters.
- An application client JAR by default includes all classes in the project's output path and the generated classes, such as IDL stubs. The Java class file type is included by default.

- A connector module RAR by default contains only descriptors. Classes in the project's output are not included.
- An application module EAR by default contains descriptors and archive files defined in the descriptor. The Java class type is excluded by default.

Note To add custom classes to a module, you must remove the Java class file type filter (if it is included by default for the module) before you add a custom Java class include filter. The default Java class file type filter includes all classes in the project, which overrides any custom include class filters.

Clean filters

You add clean filters using the Clean page of the Properties dialog box for a module. Clean filters control which files are deleted in the module directory before the module is rebuilt whenever you choose either the Rebuild Project or the Rebuild Module context menu command. A number of clean filters (exclude filters) are added to prevent the deletion of mandatory files, such as descriptors, CVS content, and so on. Be sure that you add the appropriate exclude clean filters to prevent the deletion of any custom content added manually to the module directory (such as custom descriptors that JBuilder does not recognize).

Dependencies

Use the Content|Dependencies page of the Properties dialog box for a module to add libraries to the module. This page lists all libraries in a project including dependent libraries. These are the default values for J2EE modules:

- EJB modules: All libraries are excluded by default. Including a library will expand the library inside the EJB JAR.
- Web modules: All libraries except the server libraries (Client and Servlet) are included in the WAR. Including a library will include the JAR in the WAR's WEB-INF/lib directory.
- Connector modules: All libraries are excluded by default. Including a library will include the JAR inside the RAR (at the root level).
- Application client modules: All libraries are excluded by default. Including a library will expand the library inside the JAR.
- Application modules: All libraries are excluded by default. Including a library will include the JAR inside the EAR (at the root level). Use the Application|Other page of the Properties dialog box for the application module to include files in a specific directory structure inside an EAR.

Adding custom file types

You can add custom file types to the file types that JBuilder recognizes:

1. Choose Tools|Preferences.
2. Select the Browser|File Types page.
3. Select Generic Resource File in the Recognized File Types list box.
4. Click the Add to display the Add Custom Extension dialog box.
5. Enter the new file extension in the dialog box and choose OK.

   The extension you specified appears in the Associated Extensions list.

Note Create all resource files in the project's source path to ensure that a resource file is copied over to the project's output path and to the module directory.

To select file types to be copied over to the project's output path and the module directory,

1 Choose Project|Project Properties.
2 Select the Build|Resource page and select the custom file type.
3 Select the Copy option.
4 Choose OK.

Note You will need to refresh and make the project after this step before you rebuild the module. You will have to add an include filter for this file type for all other modules (**/*.<file type>) on the Content page of the Properties dialog box for the module.

Sample build scenarios

Here are a few common build scenarios to help you understand how to work with build properties:

EJB JAR without any bean classes

This sample EJB JAR includes only stubs/skeletons and descriptors. The server used is WebLogic Server. Follow these steps:

1 Right-click the EJB module in the project pane and choose Properties.
2 Select the Build|WebLogic page and uncheck the Remove Project Output Path From Classpath option.
3 Select the Content page.
4 Remove the Java Class file type filter.
5 Choose OK.

EJB JAR with custom files

This sample adds `.dat` files to the EJB JAR. Please ensure that you create a `.dat` file in the project's source path for this test.

1 Choose Tools|Preferences.
2 Select the Browser|File Types page.
3 Select Generic Resource File in the Recognized File Types list box.
4 Click the Add button to display the Add Custom Extension dialog box.
5 Enter `dat` and choose OK.
6 Choose Project|Project Properties and select the Build|Resource page.
7 Select `dat` from the list of extensions and click the Copy button.
8 Choose OK.
9 Right-click the EJB module and select Properties.
10 Select the Content page and click the Add Filters button to display the Add Filters dialog box.
11 Select from one of these methods of specifying a filter:
   - Select the Include option and specify `**/*.dat` in the Expression field.
   - Click the File Types tab, select Generic Resource File, and choose OK.
12 Choose OK.

EJB JAR without EJB designer XML descriptors

Follow these steps:

1. Right-click the EJB module and select Properties.
2. Select the Content page and click the Add Filters page.
3. Select the Exclude option.
4. Enter this exclude filter in the Expression field: `META-INF/ejb-modeler*.xml`
5. Choose OK.

WAR with custom descriptor(s) in the WEB-INF directory

Perform this one step: Copy the custom descriptor(s) over to the WEB-INF directory of the web module.

Note The descriptor must have a standard extension listed in the XML file type (Tools|Preferences|Browser|File Types). If the file type is not recognized by JBuilder, add the required extension to the global file types. You must do this whenever adding custom descriptors to any J2EE module in JBuilder.

WAR without any classes

This sample will create a WAR with descriptors only. Follow these steps:

1. Right-click the web module and choose Properties.
2. Select the Content page and remove the Java Class file type filter.
3. Choose OK.

WAR with custom class filters

Follow these steps:

1. Right-click the web module and choose Properties.
2. Select the Content page and remove the Java Class file type filter.
3. Add include filters for the required classes (for example, <package name>/*.*).
4. Choose OK.

WAR with custom file types

This sample adds the `.inc` extension to the JSP file type:

1. Choose Tools|Preferences.
2. Select the Browser|File Types page.
3. Select JSP File in the Recognized File Types list box.
4. Click the Add button to display the Add Custom Extension dialog box.
5. Enter `inc` and choose OK.
6. Right-click the web module's Module Directory in the project pane and select New|File.
7. Create a file named `test1` with the extension `inc` in the module directory at the root level.

EAR with custom files

This sample adds .dat files along with the archives and descriptors to an application module (EAR):

1 Choose Tools|Preferences.
2 Select the Browser|File Types page.
3 Select Generic Resource File in the Recognized File Types list box.
4 Click the Add button to display the Add Custom Extension dialog box.
5 Enter `dat` and choose OK.
6 Choose Project|Project Properties.
7 Select the Build|Resource page.
8 Select dat in the list box and select the Copy option.
9 Right-click the application module and choose Properties.
10 Select the Content page and click the Add Filters button.
11 Select the Include option and add the follow filter in the Expression field: `**/*.dat`
12 Choose OK.

RAR with classes and dependencies

Follow these steps:

1 Right-click the connector module and choose Properties.
2 Select the Content page.
3 Select the Java Class file type and click the Add Filters button to add a new filter.
4 Select the Include option and add a new filter in the Expression field.
5 Choose OK.
6 Select the Content|Dependencies page and select the required libraries. Select the Include All option.
7 Choose OK.

Application client module with classes and dependencies

Follow these steps:

1 Right-click the application client module and choose Properties.
2 Select the Content page.
3 Select the Java Class file type and click the Add Filters button to add a new filter.
4 Select the Include option and add a new filter in the Expression field.
5 Choose OK.
6 Select the Content|Dependencies page and select the required libraries. Select the Include All option.
7 Choose OK.

These steps will expand the libraries into the application client JAR.

Improving module build performance

To improve module build performance, try these suggestions:

- Remove file type filters that are not being used in the module. To do this, right-click the module and choose Properties. Select the Module page and remove the file type filters that are not in use for the module.
- Turn off archive generation if the archive is not being deployed. To do this, right-click the module and choose Properties. Select the Build page and select Never in the Build <module type> Archive drop-down list.
- Turn off the Build <Module> Directory option for modules other than web modules and also for web modules if you are using a server other than Tomcat or WebLogic. To do this, right-click the module and choose Properties. Select the Build page and in the Build <module type> Directory drop-down list, select Never.

Warning Anything that is not excluded using an exclude filter on the Clean page will be deleted during the clean process. Be sure to add an exclude filter on the Clean page for any custom file types that you are using in your module directory.

Web modules

To improve web module build performance, turn off dependencies that are no being used in the module:

1. Right-click the web module and choose Properties.
2. Select the Content|Dependencies page.
3. For each library that is not needed, select the Exclude All option for the Dependency Rule value.
4. Choose OK.

Turning off the option Only Include Module Specific Java Classes and adding custom filters to control content also improves module build performance.

Index