Borland

A Guide to Porting Applications

Migrating from BEA® WebLogic® to Borland® Enterprise Server

By Borland Software Corporation

August 2002

This paper has been created in cooperation with Carbon 5, based on a real-world migration project

Contents

Introduction

Portability is one of the biggest claims to fame for Java.™ "Write once, run anywhere" is the main slogan promoted by Sun Microsystems for Java. But is this assertion actually true? For J2SE™ applications, the answer is usually yes, and some would say most definitely, yes. For J2EE™ applications, the answer is also yes, but with some minor caveats. Sun has gone a long way toward portability, especially with J2EE 1.3, but the hard truth is that no specification can cover 100 percent of an application's required functionality.

Therein lies the motivation for this paper: a porting guide for J2EE applications from BEA® WebLogic Server™ 6.1 to Borland® Enterprise Server 5.0.1, AppServer™ Edition.

Reduce porting times

The goal of this porting guide is to dramatically reduce the time it takes to port J2EE applications from the WebLogic Server to Borland Enterprise Server for the engineer with little or no Borland Enterprise Server experience. In order to achieve this goal, this paper presents technical information specific to this subject, digressing into related topics only very briefly and only when they are deemed contextually necessary.

Individuals may use this guide both by reading it completely before attempting a WebLogic Server to Borland Enterprise Server port, or by referring to it during a port. Use of both techniques will help to get the most out of this document.

Enterprise Server™

white paper

For those experienced with WebLogic™

The target audience for this porting guide have the following experience:

- Highly technical
- Good understanding of J2EE components
- Working experience with WebLogic Server
- Little or no experience with Borland Enterprise Server, AppServer Edition

An individual comes away from reading this document with a strong understanding of the practical implications of porting a J2EE application from the WebLogic Server to Borland Enterprise Server. They know about common issues encountered during a port of this type, along with the solutions to these issues. Overall, individuals take away the knowledge needed to significantly reduce the time frame of a WebLogic Server to Borland Enterprise Server porting project.

This document assumes that the individual has an existing WebLogic Server-based J2EE application and has successfully installed Borland Enterprise Server.

The approach

Certain basic steps should be taken when attempting to port J2EE applications , and perhaps any porting project in general.

1. Install the new environment
2. Build/compile the application in the new environment
3. Deploy the application to the new environment
4. Test the deployed application and fix bugs

With this simple recipe in mind, an engineer can slice up the problem domain into more manageable goals, thereby increasing the chance of success.

Bottom-up approach

Within the basic recipe, a more detailed set of instructions is very helpful. This includes a high-level order of operations, tips and tricks, and a step-by-step guide.

First of all, when porting a J2EE application, one should concentrate on a single J2EE component as much as possible in each step of the basic recipe, and do so in a bottom-up manner. This means one should first port the fundamental, underlying layers and work his or her way up the dependency stack to the highest layer. Of course, some porting issues will span more than one J2EE platform technology. When this happens, it is best to make only the minimal necessary changes across the other components, knowing that these issues can be fixed later in the porting process. Of course, installing the new environment requires addressing many components at once.

JNDI allows an application to locate objects and services and does not rely on any other J2EE platform technologies. It is this writer's opinion that JNDI is the most vendor-specific component in most J2EE application servers (this is definitely the case with the WebLogic

Server and Borland Enterprise Server) and should be looked at first during a port. Move up the stack to JDBC,® which relies only on JNDI. Proceed to EJB™ and Java™ Message Service (JMS), and finally to servlet and JSP.™ These are the J2EE components that are specifically covered by this document.

Here is a suggested ordered list for porting J2EE components. We follow this ordering throughout this guide:

1. JNDI
2. JDBC
3. EJB
4. JMS
5. Servlet
6. JSP

Order of operations

On a per-component basis, one should use the following step-by-step iterative guide to make continuous progress toward a running J2EE application on Borland Enterprise Server.

- **Make the obvious changes first.**

The first thing to do is to go through the code and/or configuration of the specific component and make any obvious changes. In the first iteration of these steps, this first step has a wide range of choices in changes between different people. This guide should help determine a good amount of those choices. After the first iteration, the obvious modifications become what were learned during the previous iteration.

- **Try it out**.

The next step is to test the changes.

- **Fix problems.**

Obviously if there are no problems, then the port is complete (after the next step). Otherwise, one must address the specific issues that have been discovered.

- **Document your findings and actions.**

This is a very important step. Documenting what was done, and why it was modified should always follow changes to code or configuration. This helps others who attempt similar porting projects in the future. No matter how clearly a person remembers what he did today, next week he may well wish it were written down.

- **Repeat.**

Unless the port is done, it is time to start over on the next component.

A useful tip

The most useful pieces of documentation available during a porting project are existing, working examples. These can be found with the Borland Enterprise Server installation program, and should be utilized whenever necessary.

J2EE™ and CORBA®

Borland Enterprise Server Remote Procedure Call (RPC) functionality (i.e.: EJB) is built on top of Borland® VisiBroker,® the underlying CORBA technology. The EJB functionality in WebLogic Server is built from scratch. So, engineers familiar with the WebLogic Server most likely do not have a good understanding of CORBA-specific rules and idiosyncrasies. Borland Enterprise Server does a good job of hiding the underlying CORBA architecture from the J2EE programmer, but every once in a while, the CORBA foundation becomes visible through an application exception. When this happens, it is best to avoid the issue as long as possible. There are some specific examples below, but use this a rule of thumb: Don't let CORBA warnings scare you. If something is serious enough, it will generate an error instead of a warning. Many J2EE applications can be successfully deployed even when showing multiple CORBA warnings.

Before you get started

The first step in a Borland Enterprise Server porting project is to make sure the latest version of Borland Enterprise Server is installed. The latest supported version of Borland Enterprise Server, AppServer Edition at the time of this writing is 5.0.1, with 5.1 being planned for the third quarter of 2002. Using the latest version of released software usually allows for the easiest possible process.

Starting and stopping the application server

It is very important to know how to start and stop your application server. These are very platform-dependent actions.

Starting from the Start Menu (Windows®)

WebLogic

```
Start / BEA WebLogic E-Business Platform / WebLogic Server 6.0 / Start Default Server
```

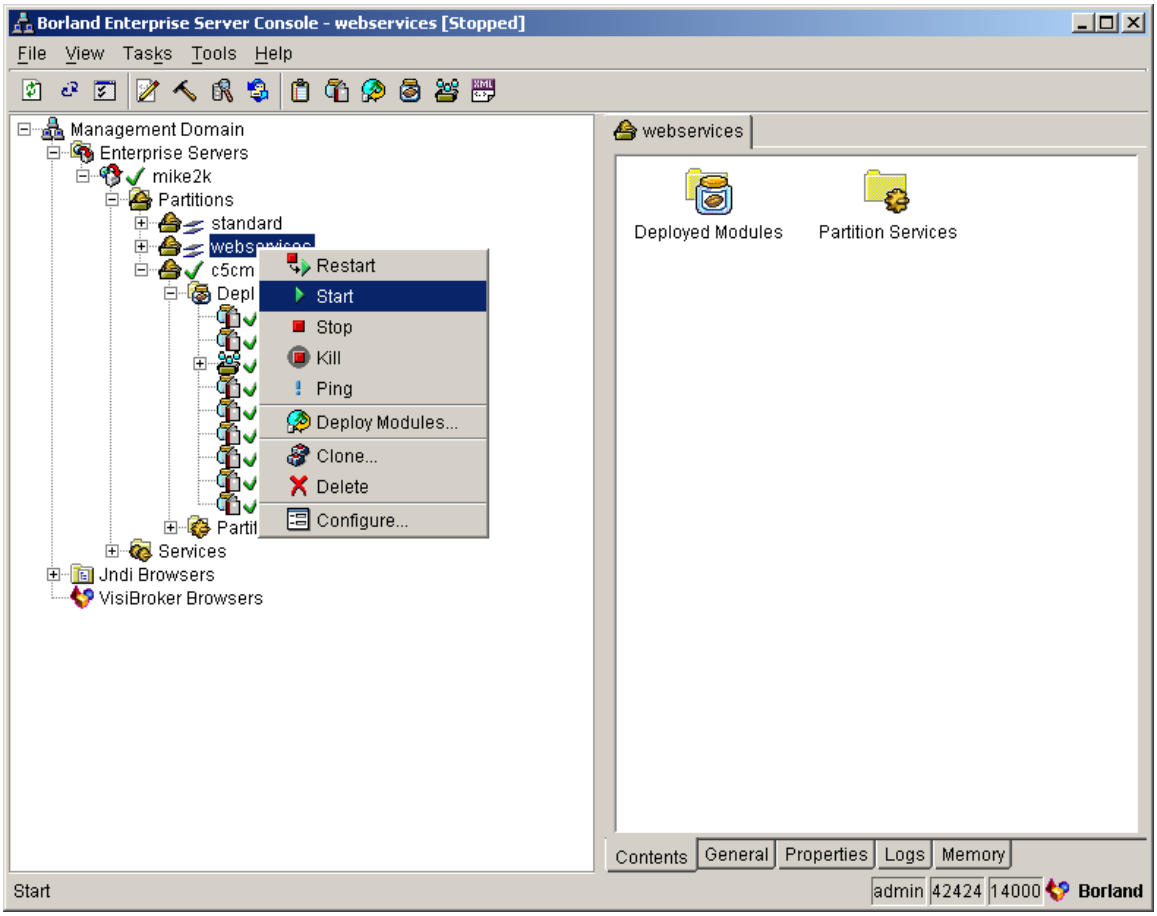
Borland Enterprise Server

```
Start / Borland Enterprise Server / Server
```

Starting from the console

This can be an unusual concept for application server users because usually the console requires a server to be running in order to function. This is certainly the case for WebLogic Server. But in Borland Enterprise Server, a server is split into partitions and services, both of which can be stopped and started from the console. One can right-click the stopped partition or service and select "Start."

The concept of partitions is very important in the Borland Enterprise Server environment, especially when compared to most other application servers, such as WebLogic Server. Partitions allow multiple applications to run on a single Borland Enterprise Server installation, each within their own "sandbox." Each partition can utilize its own chosen set of components and services.



Starting from the console

Starting from the command line

WebLogic

% <wls-home>/config/<domain>/startWebLogic.cmd

Borland Enterprise Server

% <bes-home>/bin/ias

Stopping from the console

WebLogic

In order to stop a server from the WebLogic Server console, one can **right-click the server** in the left-hand tree hierarchy Java applet and select **Shutdown server…**.

Borland Enterprise Server

Similarly to starting from the console, in Borland Enterprise Server a user can **right-click any server, partition, or service** and select **Stop** (or "Kill" if stopping does not work).

Stopping from the command line

WebLogic

% java –classpath <wls-home>/lib/weblogic.jar \

```
        weblogic.Admin stop
```

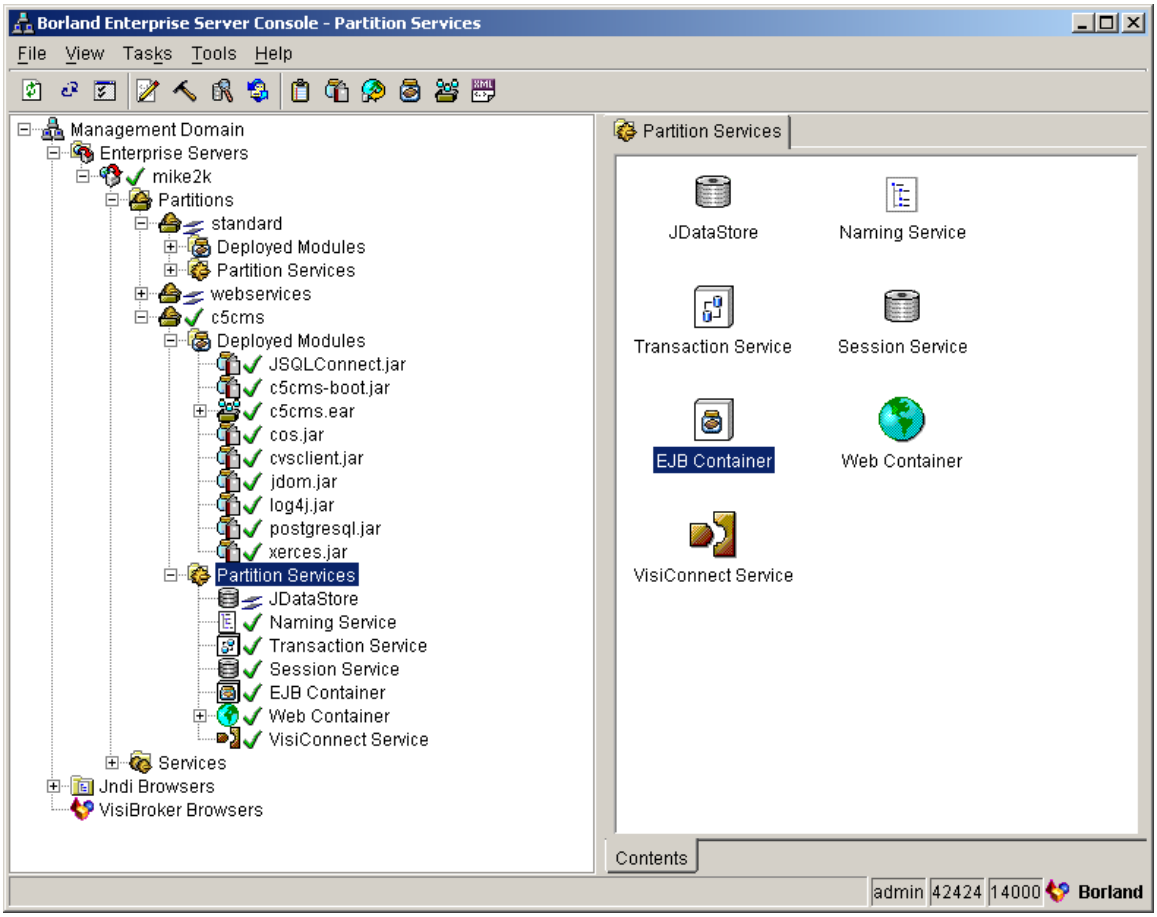
Borland Enterprise Server

% <bes-home>/bin/iastool –stop –server <server-name>

Managing services

Managing services in general is a bit out of scope for this document, as management comes well after an application has been built and deployed (or ported, for that matter). But both WebLogic Server and Borland Enterprise Server have command line administrative management tools that are used during the compile and deploy processes. Specific uses will be detailed in later sections, but in general, these are the tools to keep in mind.

As stated above, partitions within Borland Enterprise Server are quite useful when managing services. A user may enable or disable a different set of services for each partition within a Borland Enterprise Server installation.



Managing partitions and services from the console

WebLogic

```
% java –classpath <wls-home>/lib/weblogic.jar \
        weblogic.Admin
```

Borland Enterprise Server

```
% <bes-home>/bin/iastool
```

Important files and directories

A mapping of file and directory locations from WebLogic Server to Borland Enterprise Server is one of the most useful sets of information for a porting project. Following is a table of important files and directories in WebLogic Server and Borland Enterprise Server.

Application root

WebLogic `<wls-home>/config/<server>`

Borland Enterprise Server `<bes-home>/var/servers/<server>/partitions/<partition>`

Application configuration

WebLogic `<wls-home>/config/<server>/config.xml`

Borland Enterprise Server `<bes-home>/var/servers/<server>/adm/properties/<many-files>`

Event log file

WebLogic `<wls-home>/config/<server>/logs/weblogic.log`

Borland Enterprise Server `<bes-home>/var/servers/<server>/adm/logs/partitions/<partition>/event.log`

Error log file

WebLogic `<wls-home>/config/<server>/logs/weblogic.log`

Borland Enterprise Server `<bes-home>/var/servers/<server>/adm/logs/partitions/<partition>/error.log`

All partitions classloader library directory

WebLogic none

Borland Enterprise Server `<bes-home>/var/servers/<server>/partitions/lib`

Single partition classloader library directory

WebLogic `<wls-home>/config/<server>/lib`

Borland Enterprise Server `<bes-home>/var/servers/<server>/partitions/<partition>/lib`

Where to put WARs

WebLogic `<wls-home>/config/<server>/applications`

Borland Enterprise Server `<bes-home>/var/servers/<server>/partitions/<partition>/wars`

Where to put EARs

WebLogic `<wls-home>/config/<server>/applications`

Borland Enterprise Server `<bes-home>/var/servers/<server>/partitions/<partition>/ears`

Where to put stand-alone EJB™

WebLogic `<wls-home>/config/<server>/applications`

Borland Enterprise Server `<bes-home>/var/servers/<server>/partitions/<partition>/ejb_jars`

J2EE™ components and JNDI

Java Naming and Directory Interface (JNDI) is the core of any J2EE application. It is how services and objects get registered and looked up. It is required by most— if not every—other component of J2EE. It is the first building block in the J2EE stack. Because of this, it is the most platform-dependent component within J2EE—not in its implementation, in fact most JNDI implementations are interchangeable, but in how it is used to register and look up objects in a specific application server platform. The basic idea of JNDI is simple enough, however, to make understanding platform dependencies manageable.

The five most common items stored in JNDI that are user-configured in a J2EE application are:

- EJB remote home interfaces
- EJB local home interfaces
- JDBC data sources
- JMS connection factories
- JMS topics
- JMS queues

More about these specific items will be explained in later sections.

Binding EJB objects (remote and local home interfaces) into JNDI at server startup time is very similar between WebLogic Server and Borland Enterprise Server. Binding JDBC and JMS objects, however, is somewhat different between the two application servers.

Within the WebLogic Server, JDBC and JMS objects are bound to JNDI via the configuration in:
`<wls-home>/config/<server>/config.xml`

Within Borland Enterprise Server, these objects are bound to JNDI via the configuration in the file jndi-definitions.xml. This file can reside in the META-INF directory of an enterprise archive (EAR file), or it can be deployed directly to Borland Enterprise Server using the following command:

```
<bes-home>/bin/iastool -deploy \
 -jars jndi-definitions.xml \
 -server <server>
```

It should be noted that the -jars parameter could specify more than just JAR files. Files of type XML, JAR, WAR, and EAR can be deployed in this manner.

More specifics about binding JDBC and JMS objects using these configuration files can be found in later sections.

What must be ported for JDBC®

JDBC drivers are database-specific, and often times have nothing to do with the application server itself. Both WebLogic Server and Borland Enterprise Server support all JDBC 2.0 compliant drivers. Often porting projects include swapping JDBC drivers, however, so direction is provided in this document.

Many factors weigh in when porting JDBC code. JDBC driver, JDBC version, database vendor and version, and application architecture are all factors. Some quick analysis of the application specifics can help locate areas to concentrate on during the port.

If the application being ported uses EJB Container Managed Persistence (CMP) as the only persistence mechanism (i.e.: there is no SQL code in the Java application), then only JDBC configuration must be ported. This is probably the easiest case to handle. If that is not the case, then a few more questions must be answered.

If the Borland Enterprise Server application is using the same database vendor and version, and the same JDBC driver as the WebLogic Server application, then the same conclusion can be drawn: most likely, only JDBC configuration must be ported, not JDBC code.

If this is not the case, the port becomes a bit more complicated. Changing JDBC drivers, database vendors, or database versions can introduce bugs due to incompatibilities or inconsistencies between drivers and databases. Each database vendor has extended SQL syntax support in a different way, and code that is specific to one database vendor may not run correctly in another database. Even between database versions, these types of problems can occur. The best way to find these problems is to run the new application with the new drivers/database, find bugs, and fix them.

JDBC® 1.2 vs. JDBC® 2.0

An upgrade from JDBC 1.2 to 2.0 is a major change in JDBC drivers being introduced right now by many vendors. It might be useful to understand some of the main differences between the two versions.

The biggest difference in 2.0 is the addition of data sources. These objects represent serializable database connection objects. Java code can look them up in JNDI and get a database connection from them. JDBC 2.0 also offers support for database connection pooling, which was previously supported by some application servers in a proprietary way (WebLogic Server does this).

Another major change in JDBC 2.0 revolves around the ability to update database content through a result set. Previously, only by executing an update command could one modify the database. With 2.0, a new paradigm is added where a JDBC result set can be added in Java code, thereby updating the database.

JNDI specifics

Acquiring a JDBC data source is the entry point into any database access in a J2EE application. Each application server is configured differently in this respect.

In the WebLogic Server, JDBC configuration is located in the config.xml file. Connection pools are established, and data source are defined which point to those pools. The configuration looks like this:

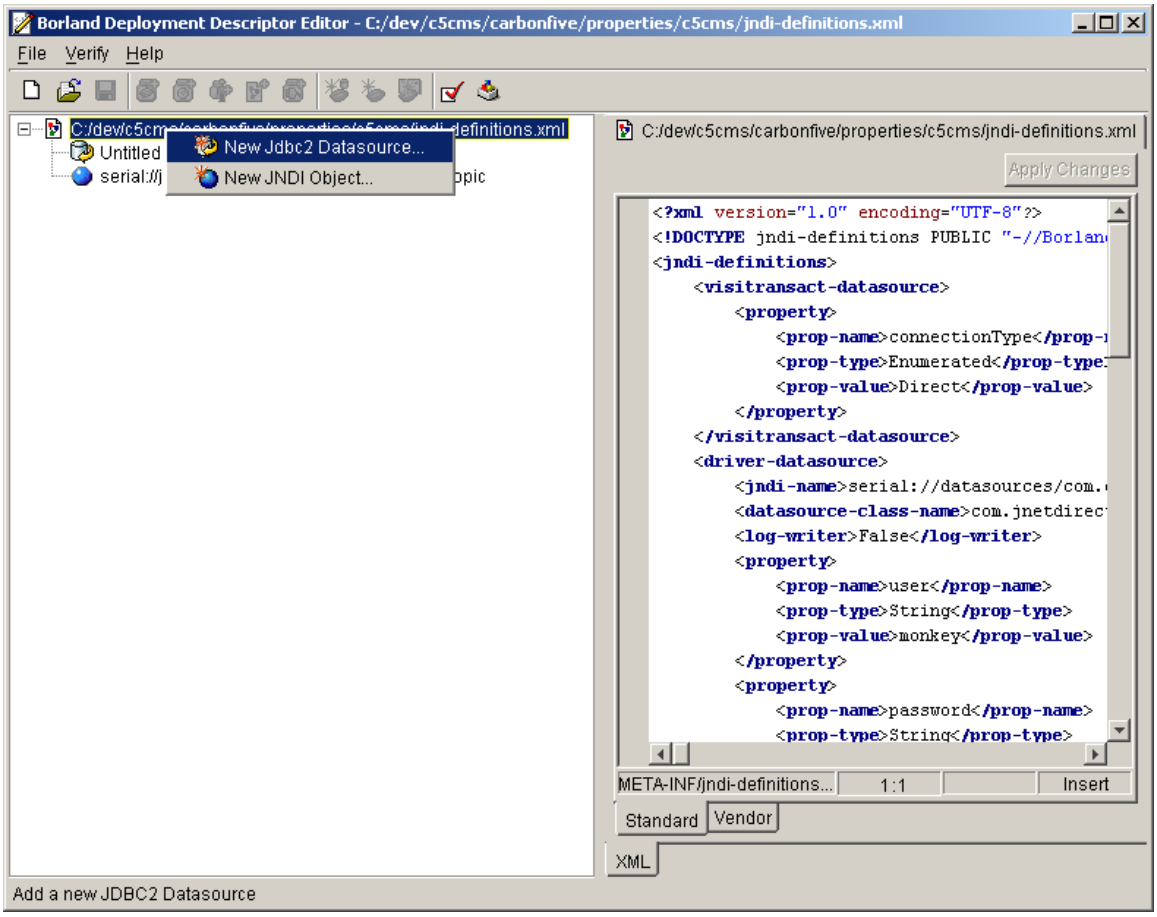
```
<JDBCConnectionPool
 CapacityIncrement="5"
 DriverName="weblogic.jdbc.mssqlserver4.Driver"
 InitialCapacity="5"
 MaxCapacity="25"
 Name="MssqlPool"
 Properties="user=jim;password=beam"
 RefreshMinutes="5"
 Targets="MyWebApp"
 TestTableName="test_table"
 URL="jdbc:weblogic:mssqlserver4:MyDB@mssql1:1433"/>

<JDBCDataSource
 JNDIName="env/jdbc/MsSqlDataSource"
 Name="MSSQLDataSource"
 PoolName="MssqlPool"
 Targets="MyWebApp"/>
```

In Borland Enterprise Server, the same configuration is located in the jndi-definitions.xml file. The configuration looks like this:

```
<jndi-definitions>
 <driver-datasource>
  <jndi-name>serial://datasources/MsSqlDataSource
  </jndi-name>
  <datasource-class-name>
   com.jnetdirect.jsql.JSQLDataSource
```

```
  </datasource-class-name>
  <log-writer>False</log-writer>
  <property>
   <prop-name>user</prop-name>
   <prop-type>String</prop-type>
   <prop-value>jim</prop-value>
  </property>
  <property>
   <prop-name>password</prop-name>
   <prop-type>String</prop-type>
   <prop-value>beam</prop-value>
  </property>
  <property>
   <prop-name>databaseName</prop-name>
   <prop-type>String</prop-type>
   <prop-value>MyDB</prop-value>
  </property>
  <property>
   <prop-name>serverName</prop-name>
   <prop-type>String</prop-type>
   <prop-value>mssql1</prop-value>
  </property>
  <property>
   <prop-name>portNumber</prop-name>
   <prop-type>Integer</prop-type>
   <prop-value>1433</prop-value>
  </property>
 </driver-datasource>
</jndi-definitions>
```

You will notice that the JNDI name of the Borland Enterprise Server data source starts with “serial://”. The Borland Enterprise Server Naming Service provides a means to store the data source objects into a Serial Context (URL starts with "serial://") instead of the default CosNaming namespace that can store only CORBA object references. Objects like JDBC data sources and J2EE resource connection factory objects need to be looked up from JNDI. Any JNDI provider that can store Java serializable and referenceable objects can be substituted if the user wants to (e.g. an LDAP directory external to Borland Enterprise Server). Plain Java objects that implement both *java.io.Serializable* and *javax.naming.Referenceable* can be stored under this special JNDI URL context "serial://".



Adding a data source with the Deployment Descriptor Editor

Tips and tricks

The most common tip in the JDBC arena, if an application needs to make direct JDBC calls, is to have a class that returns a database connection. With this architecture, a JNDI lookup for a data source (or pooled data source) is centralized and can be modified easily. This way, if no custom SQL code is used in JDBC calls, only this centralized lookup must be modified in JDBC code. The standard J2EE pattern that addresses centralizing JNDI lookups is called Service Locator.

EJB™

EJB™ 1.1 vs. EJB™ 2.0

WebLogic supports EJB 1.1, and Borland Enterprise Server supports EJB 2.0. Herein lies the main porting difficulty between these two platforms. A brief explanation of the major differences is in order.

Remember that this porting guide describes steps to take when moving from WebLogic Server 6.1 to Borland Enterprise Server 5.0.1. Note that an optional add-on exists for the WebLogic Server which implements a non-compliant version of EJB 2.0, and WebLogic 7.0 does support EJB 2.0, but as of the writing of this document, WebLogic 7.0 is very new and is outside the scope of this porting document.

Message Driven Beans

Because Message Driven Beans (MDB) are not supported in EJB 1.1, there is no porting path to document. While it is true that WebLogic Server did have some special MDB support in its 6.1 product, it is out of the scope of this document.

Local beans

Local beans did not exist in EJB 1.1, therefore there is no porting path to document.

CMP 2.0

The new implementation of Container Managed Persistence (CMP) is a vast upgrade in EJB 2.0. It is generally agreed that CMP in EJB 1.1 is not ideal for certain applications, especially those with complex database relationships, and the majority of applications utilized Bean Managed Persistence (BMP) instead. A brief discussion of BMP vs. CMP follows. For the purposes of this document, it is assumed that all EJB code is EJB 1.1 BMP ported to EJB 2.0 BMP.

BMP versus CMP

The two models for managing persistence within EJB are BMP and CMP. The distinction lies in who takes responsibility for generating SQL code to view and modify data in the database. In BMP, the programmer takes this responsibility, where in CMP, the container itself generates all SQL code.

EJB 1.1 offered a CMP specification that did not describe database table relationships well. Versions of Borland Enterprise Server before 5 did have some extensions that allowed EJB 1.1 CMP to be used in many applications, but it was not until EJB 2.0 that implementations of CMP were considered useful across a broad array of applications. A detailed description of CMP 2.0 is outside the scope of this document, as it is not a common task for porting applications from WebLogic Server to Borland Enterprise Server, but should be seriously considered when developing new applications with Borland Enterprise Server.

Deployment descriptors

Both WebLogic Server and Borland Enterprise Server have two deployment descriptors: one generic to EJB, and one specific to the application server. The generic descriptor, called ejb-jar.xml, is nearly identical between implementations. The difference is the document type definition. In WebLogic Server, the document type will refer to the EJB 1.1 DTD, while the Borland Enterprise Server document type will refer to the EJB 2.0 DTD. For simple cases using BMP, the file does not need to change beyond that.

WebLogic Server ejb-jar.xml document type

```
<!DOCTYPE ejb-jar PUBLIC
 '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
 'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
```

Borland Enterprise Server ejb-jar.xml document type

```
<!DOCTYPE ejb-jar PUBLIC
 '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN'
 'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>
```

The real difference between applications exists in the vendor-specific EJB deployment descriptors. For WebLogic Server, this is called weblogic-ejb-jar.xml, and for Borland Enterprise Server, it is called ejb-borland.xml. Each has a set of proprietary features. For a porting project, create the simplest possible deployment descriptors, and then add custom configuration after the application is working. Examples of both simple vendor-specific deployment descriptors follow.

WebLogic Server weblogic-ejb-jar.xml

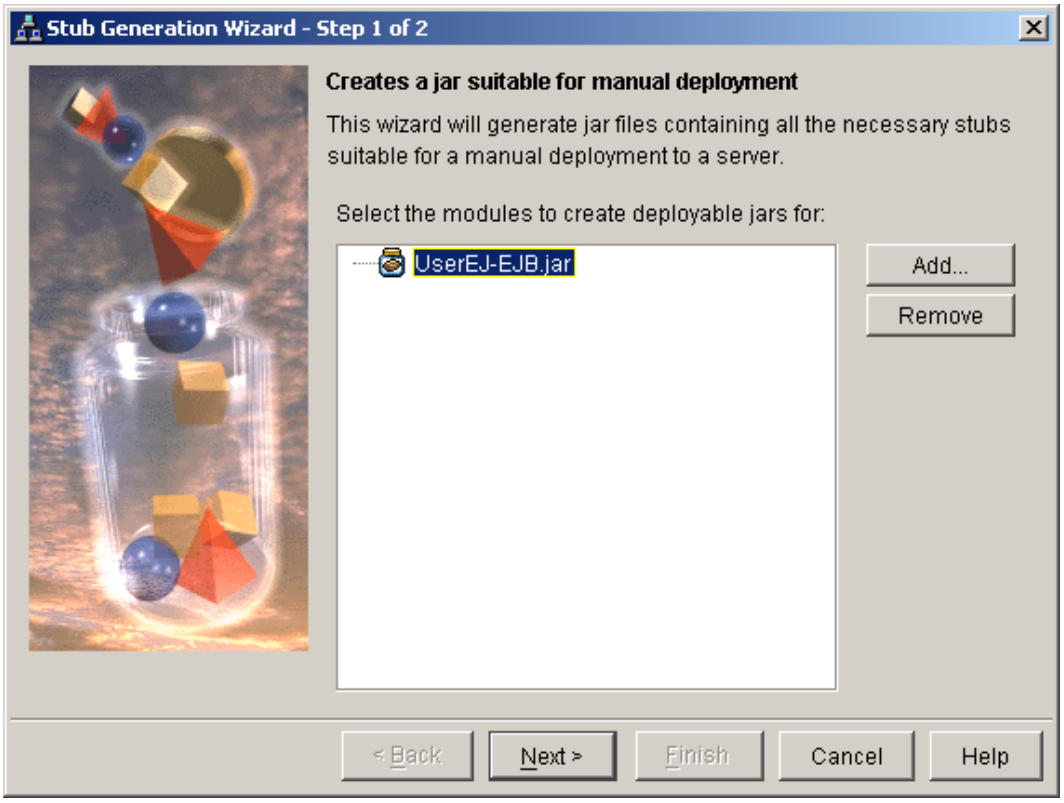
```
<?xml version="1.0"?>

<!DOCTYPE weblogic-ejb-jar PUBLIC
 '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'
 'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>

<weblogic-ejb-jar>
 <weblogic-enterprise-bean>
  <ejb-name>GroupEJB</ejb-name>
  <jndi-name>ejb/GroupEJHome</jndi-name>
 </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

Borland Enterprise Server ejb-borland.xml

```
<?xml version="1.0"?>

<!DOCTYPE ejb-jar PUBLIC
 '-//Borland Software Corporation//DTD Enterprise JavaBeans 2.0//EN'
 'http://www.borland.com/devsupport/appserver/dtds/ejb-jar_2_0-borland.dtd'>

<ejb-jar>
 <enterprise-beans>
  <entity>
   <ejb-name>GroupEJB</ejb-name>
   <bean-home-name>ejb/GroupEJHome</bean-home-name>
  </entity>
 </enterprise-beans>
</ejb-jar>
```

Compiling EJB™

Once the deployment descriptors are ported, you can move on and compile your EJB. Both WebLogic Server and Borland Enterprise Server have command line tools as well as GUI-based tools that create deployable versions of a bean. Similarly to administrative management tools, WebLogic Server uses a Java-based program for compiling, and Borland Enterprise Server uses a command line utility, but basically they do the same thing.



Compiling EJB in the console wizard

The following code assumes you have compiled your EJB class files and have packaged a jar with the necessary classes.

WebLogic Server

```
java -classpath <wls-classpath> weblogic.ejbc \
 -keepgenerated \
 -compiler javac GroupEJB_raw.jar \
 GroupEJB.jar
```

Borland Enterprise Server

```
<bes-home>/bin/iastool -gendeployable \
 -classpath <bes-classpath> \
 -src GroupEJB_raw.jar \
 -target GroupEJB.jar
```

In WebLogic Server, that's all there is to it. Borland Enterprise Server offers a verification tool, however, that one should use. It examines your EJB and displays any errors or warnings it finds. This is how to use it:

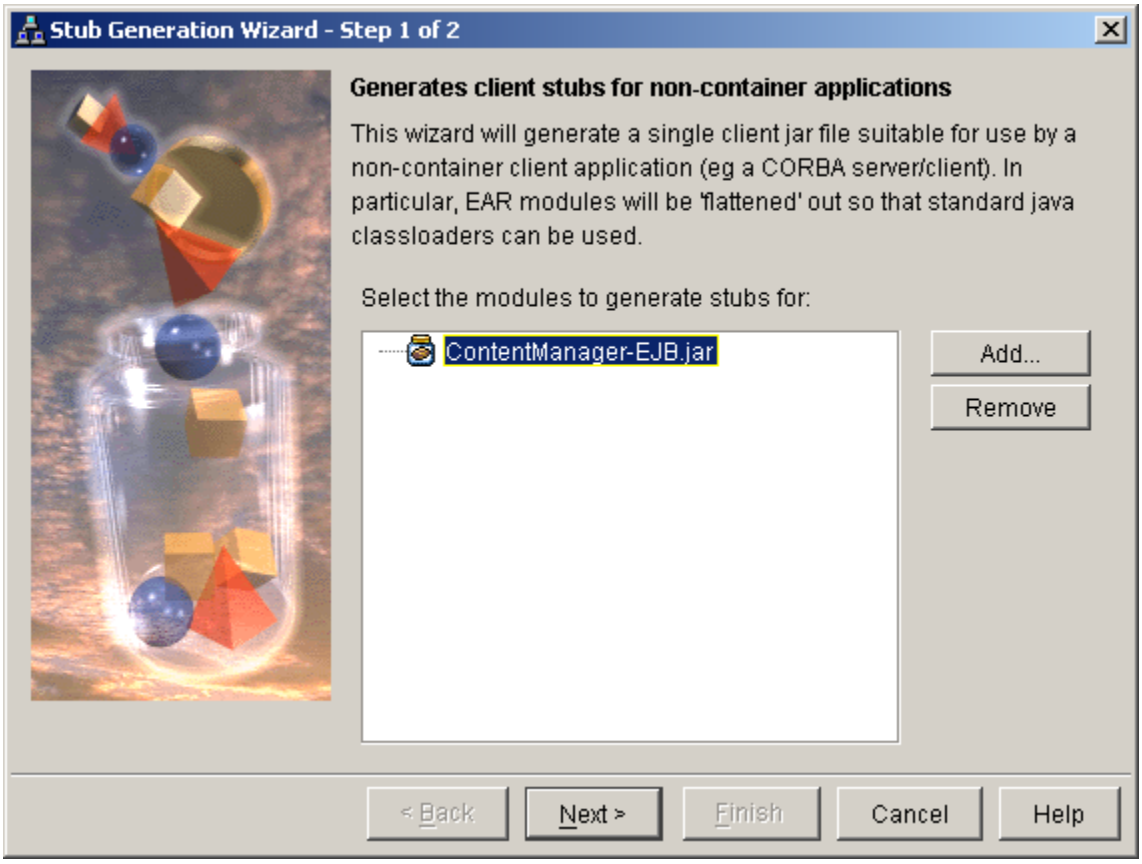
```
<bes-home>/bin/iastool -verify \
 -classpath <bes-classpath> \
 -src GroupEJB.jar \
 -role DEVELOPER
```

Note that the role parameter is set to DEVELOPER. The verify tool accepts three roles: DEVELOPER, ASSEMBLER, and DEPLOYER. When verifying an individual EJB, specify the DEVELOPER role so that external JNDI links and URI references are not checked. At deployment time, you will be able to verify the entire application using the DEPLOYER role.

Client EJB™ stubs

Normally an external EJB client, such as a servlet or stand-alone application, has the EJB JAR in its classpath in order to have access to the necessary RMI stubs and skeletons. Borland Enterprise Server, however, offers a tool to create a client EJB jar with the minimal files necessary for client access. Its use is very similar to that of generating a deployable EJB.

```
<bes-home>/bin/iastool -genclient \
 -classpath <bes-classpath> \
 -src GroupEJB_raw.jar \
 -target GroupEJB-Client.jar
```



Client stubs generation wizard

Deploying EJB™

There are two formats in which to deploy an EJB: stand-alone or within an EAR. Most commonly, J2EE application components are bundled together in an EAR file, which is deployed to an application server.

In WebLogic Server, there are two options for deployment. Both involve copying the EAR or stand-alone JAR file directly to the applications directory. If the hot-deploy feature is turned on (which is not recommended for production), the server will automatically register the change and re-deploy the EAR itself. If hot-deploy is not enabled, the new EAR will be deployed the next time WebLogic Server is restarted.

In Borland Enterprise Server, there again exist two options. Similar to WebLogic Server, one can copy the EAR file directly to the correct directory and restart the server or just the partition. Alternatively, an EAR can be programmatically deployed using the same command line tool as above.

```
<bes-home>/bin/iastool -deploy \
 -classpath <bes-classpath> \
 -jars MyApp.ear \
 -server <server> \
 -partition <partition>
```

Tips and tricks

- During verification, warnings may appear about CORBA backwards mappings not supporting classes with the same name as a package they reside in (for example, com.company.group.Group). This is common and many porting projects can be successful without fixing this warning.
- Make sure the correct DOCTYPE attribute is at the top of each XML deployment descriptor. If the verifier does not recognize some XML elements in your document, a good suggestion is to check the DOCTYPE against some Borland Enterprise Server examples.
- The correct way to cast home interfaces when locating them from JNDI, according to the specification, is to use PortableObject.narrow(). If objects are cast without using this method, application servers that utilize RMI-IIOP® as the underlying communication transport are likely to fail.

  It is important to make sure the latest version of Borland Enterprise Server (currently 5.0.1) is installed. This version corrects some issues from 5.

JMS

Implementations, connection factories, and JNDI specifics

WebLogic Server uses its own implementation of JMS, while Borland Enterprise Server bundles a configured version of SonicMQ.® Both offer similar functionality, merely differing in configuration specifics.

Both WebLogic Server and Borland Enterprise Server utilize JMS connection factories that are located in JNDI, as is part of the JMS spec. Borland Enterprise Server by default creates and binds all necessary JMS connection factories in JNDI (see next section). In WebLogic Server, the programmer is required to establish the necessary JMS connection factories.

By default in Borland Enterprise Server, the four necessary connection factories are located in JNDI here:

```
Queue connection factory          serial://jms/qcf
Topic connection factory          serial://jms/tcf
XA queue connection factory       serial://jms/xaqcf
XA topic connection factory       serial://jms/xatcf
```

In WebLogic Server, JMS configuration is located in config.xml. In Borland Enterprise Server, JMS configuration is located in jndi-definitions.xml. WebLogic Server supports the idea of a JMS server, which encompasses one or more JMS topics and queues. Borland Enterprise Server does not utilize this idea. The following shows an example of each vendor's configuration.

WebLogic Server config.xml

```
<JMSConnectionFactory
 AllowCloseInOnMessage="false"
 DefaultDeliveryMode="Non-Persistent"
 DefaultPriority="4"
 DefaultTimeToLive="0"
 JNDIName="env/jms/ConnectionFactory"
 MessagesMaximum="10"
 Name="JMSConnectionFactory"
 OverrunPolicy="KeepOld"
 Targets="MyApp"/>

<JMSServer
 Name="JMSServer"
 Targets="MyApp">
 <JMSTopic
  JNDIName="/env/jms/MyTopic"
  Name="SomeTopic"
  StoreEnabled="false"/>
</JMSServer>
```

Borland Enterprise Server jndi-definitions.xml

```
<jndi-object>
 <jndi-name>serial://jms/MyTopic</jndi-name>
 <class-name>progress.message.jclient.Topic</class-name>
 <property>
  <prop-name>topicName</prop-name>
  <prop-type>String</prop-type>
  <prop-value>SomeTopic</prop-value>
 </property>
</jndi-object>
```

As is clear from these examples, Borland Enterprise Server again utilizes the serial:// JNDI prefix in order to indicate a local JNDI tree. This convention should be followed with JMS (as well as JDBC) JNDI bindings.

Servlet/JSP™

Implementations

Similarly to JMS, WebLogic Server utilizes its own implementation of servlet and JSP, while Borland Enterprise Server incorporates a third-party implementation, Apache™ Tomcat. The version of Tomcat that Borland Enterprise Server bundles supports a later version of the servlet and JSP specifications than WebLogic Server. Sun® Servlet 2.2 and JSP 1.1 are available in WebLogic Server, while Servlet 2.3 and JSP 1.2 are available in Borland Enterprise Server. Information about these upgrades is detailed below.

Servlet 2.2 versus Servlet 2.3

The differences between Servlet 2.2 and 2.3 are relatively minor. Most likely, the only Servlet-specific change you will need to make involves the document type header in the web.xml file. Make sure it is set correctly for Servlet 2.3:

WebLogic Server web.xml (Servlet 2.2)

```
<!DOCTYPE web-app PUBLIC
 '-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN'
 'http://java.sun.com/j2ee/dtds/web-app_2_2.dtd'>
```

Borland Enterprise Server web.xml (Servlet 2.3)

```
<!DOCTYPE web-app PUBLIC
 '-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN'
 'http://java.sun.com/dtd/web-app_2_3.dtd'>
```

Other than that, Servlet 2.3 also offers a more flexible event mechanism, including new event listeners, and a filter chain mechanism. But because the functionality is new, not modified, there are no porting efforts necessary for this change.

JSP™ 1.1 versus JSP™ 1.2

The JSP specification has changed a bit more than the servlet specification. JSP files can now be authored entirely in an XML-compliant language. Much of the configuration of tag libraries has been modified and augmented as well. These modifications are the focus of this document's porting suggestions for JSP.

Deployment descriptors

WebLogic Server and Borland Enterprise Server each have two Web application deployment descriptors. One is common (web.xml), and the other is vendor-specific. For WebLogic Server, this file is called weblogic.xml, and for Borland Enterprise Server, it is called web-borland.xml.

The following are examples of the vendor-specific deployment descriptors for each platform:

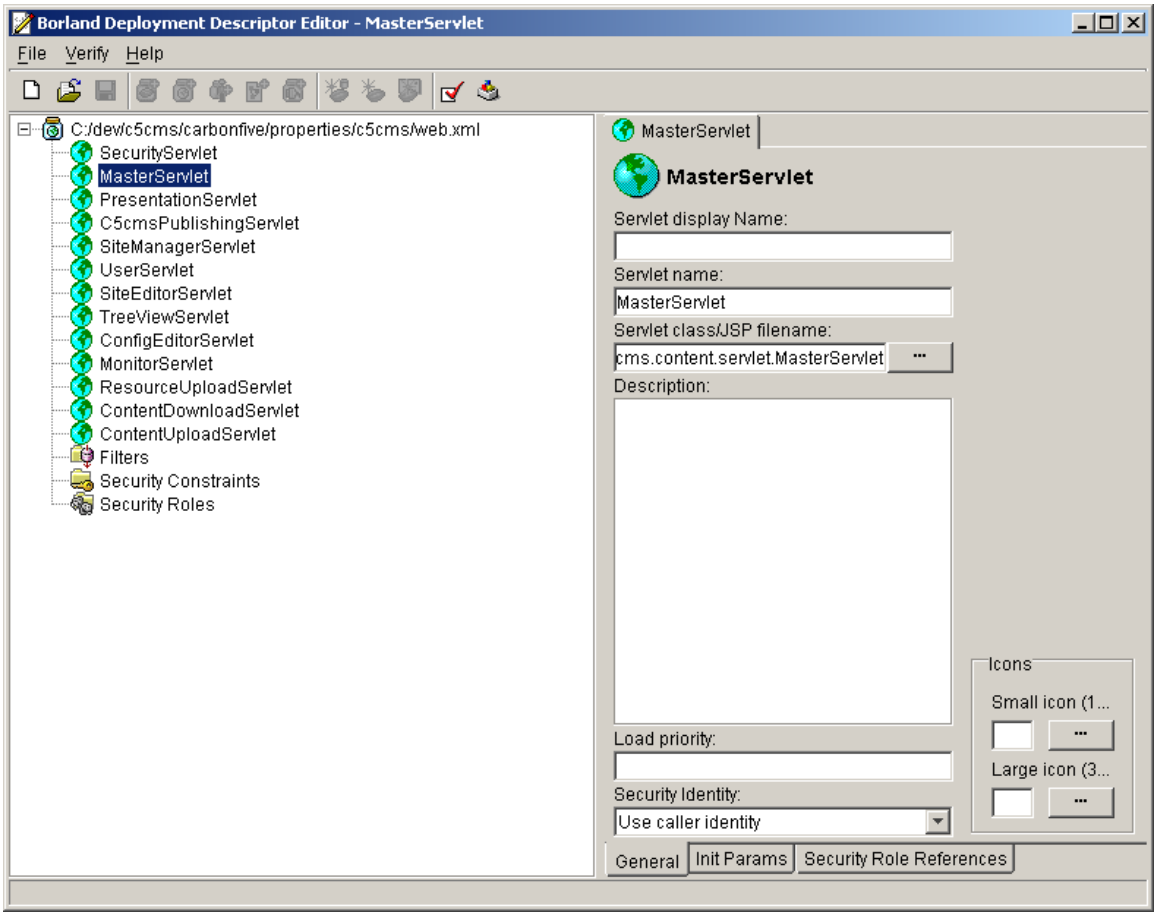
WebLogic Server weblogic.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE weblogic-web-app PUBLIC
 '-//BEA Systems, Inc.//DTD Web Application 6.0//EN'
 'http://www.bea.com/servers/wls600/dtd/weblogic-web-jar.dtd'>

<weblogic-web-app>

 <ejb-reference-description>
  <ejb-ref-name>ejb/GroupEJB</ejb-ref-name>
  <jndi-name>ejb/GroupEJHome</jndi-name>
 </ejb-reference-description>

 <ejb-reference-description>
  <ejb-ref-name>ejb/UserEJB</ejb-ref-name>
  <jndi-name>ejb/UserEJHome</jndi-name>
 </ejb-reference-description>

</weblogic-web-app>
```

Borland Enterprise Server web-borland.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC
 '-//Borland Software Corporation//DTD Web Application 2.3//EN'
 'http://www.borland.com/devsupport/appserver/dtds/web-app_2_3-borland.dtd'>

<web-app>

 <context-root>cms</context-root>

 <ejb-ref>
  <ejb-ref-name>ejb/GroupEJB</ejb-ref-name>
  <jndi-name>ejb/GroupEJHome</jndi-name>
 </ejb-ref>

 <ejb-ref>
  <ejb-ref-name>ejb/UserEJB</ejb-ref-name>
  <jndi-name>ejb/UserEJHome</jndi-name>
 </ejb-ref>

</web-app>
```

Their functions are very similar: to define local reference names for EJB home interfaces bound to JNDI. But their formats are slightly different.



Web.xml deployment descriptor editor

JSP™ tag libraries

JSP tag library definitions contain the most necessary modification during a porting project. If the application's TLD files are not JSP 1.2 compliant, the Borland Enterprise Server verifier tool will issue warnings when it verifies the WAR file and will issue errors when these custom tags are accessed. The necessary modifications to TLD files are as follows:

- Make sure to have the correct JSP 1.2 DOCTYPE declaration (see examples).
- The <info> tag is now <description>.
- All tags with multiple words in them now have dashes between the words (see examples).

Following are examples of a TLD file for WebLogic Server and Borland Enterprise Server:

WebLogic Server JSP 1.1

```
<taglib>
 <tlibversion>1.0</tlibversion>
 <jspversion>1.1</jspversion>
 <shortname>Data Entry</shortname>
 <uri>/dataentry.tld</uri>
 <info>
  Data entry TLD file
 </info>

 <tag>
  <name>de-is-visible</name>
  <tagclass>com.company.tags.DeIsVisibleTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>
      Displays if the user can see data entry fields.
  </info>
 </tag>
</taglib>
```

Borland Enterprise Server JSP 1.2

```
<!DOCTYPE taglib PUBLIC
 '-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN'
 'http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd'>

<taglib>
 <tlib-version>1.0</tlib-version>
 <jsp-version>1.2</jsp-version>
 <short-name>Data Entry</short-name>
 <uri>/dataentry.tld</uri>
 <description>
  Data entry TLD file
 </description>

 <tag>
  <name>de-is-visible</name>
  <tag-class>com.company.tags.DeIsVisibleTag</tag-class>
  <body-content>JSP</body-content>
  <description>
      Displays if the user can see data entry fields.
  </description>
 </tag>
</taglib>
```

Tips and tricks

- DOCTYPE headers in all XML deployment descriptors must be correct, otherwise the Borland Enterprise Server container won't recognize the elements and won't start up correctly.
- If non-XML-compliant HTML exists in any JSP documents, following DOCTYPE header is required:

```
<!DOCTYPE HTML PUBLIC
 '-//W3C//DTD HTML 4.01 Transitional//EN'
 'http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd'>
```

- Order is important in web.xml. Tomcat will complain if your elements are out of order.

Miscellaneous

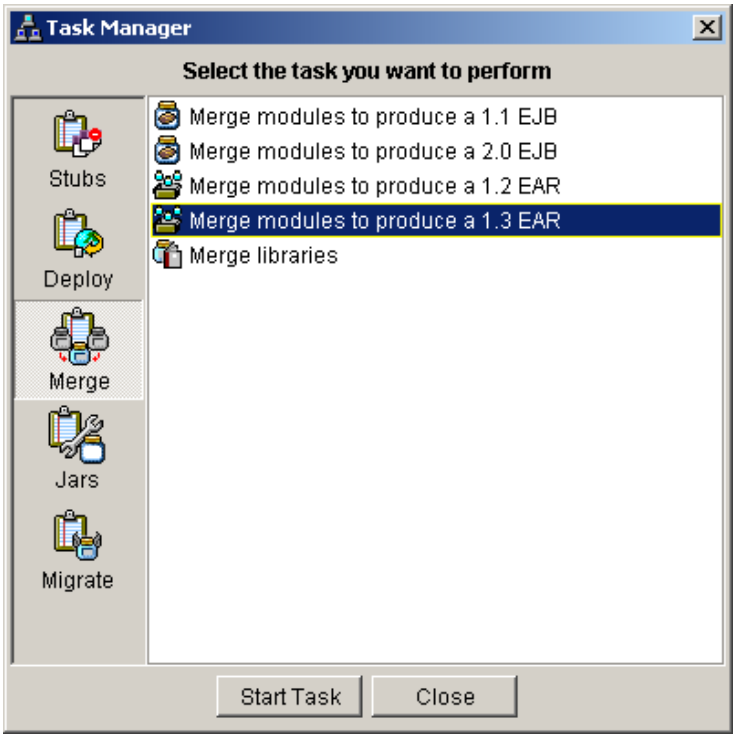
J2EE™ application bundling

In the ideal case, a J2EE application is bundled together as a single EAR file, which includes all components (EJB, WAR, etc). This makes moving the application from server to server very easy–just the copying of a single file. With WebLogic Server and Borland Enterprise Server, creating and deploying EAR files is handled somewhat differently.

Borland Enterprise Server offers a tool to merge many different files together into a single EAR file. Below is an example of how to use this tool to merge a WAR into an existing EAR.

```
<bes-home>/iastool -merge \
 -classpath <bes-classpath> \
 -jars partial-app.ear,webapp.war \
 -target app.ear \
 -type ear1.3
```

Alternatively, one can package the entire EAR by hand, but if a separate WAR and EAR file exist during a porting project, merging them with the tool is a good option. Using the merge tool will automatically create the application.xml file that must exist within an EAR file.



Task Manager–merge tool

Tips and tricks

- When deploying a separate WAR and EAR into a Borland Enterprise Server partition, servlets that load on startup and need access to JNDI-bound objects from the EAR will attempt to access those objects before they have been bound. This will result in an error. The solution is to create a single EAR file.
- The Borland Enterprise Server merge utility sometimes produces an incorrect application.xml file. The solution is to create the EAR file and application.xml manually, instead of using the merge utility.
- Borland Enterprise Server is tightly integrated with Borland® JBuilder,™ which handles many aspects of packaging and deploying. IDE users will get a lot of value out of using these two tools together.

Conclusion

While this guide is not intended to solve every potential issue that arises during a WebLogic Server to Borland Enterprise Server 5.0.x, AppServer Edition porting project, hopefully it will greatly reduce the time spent on such an endeavor. Certainly any specific porting project should produce documentation around the idiosyncrasies encountered, and combined with this document, should make another similar project almost trivial. Good luck.

For more information, contact an Enterprise Sales Representative in your region. Call 1-800-632-2864 in the U.S. International customers can find the nearest regional office at http://www.borland.com/company/borland_worldwide.html

Borland®

100 Enterprise Way
Scotts Valley, CA 95066-3249
www.borland.com | 831-431-1000