BEA WebLogic Workshop™ Help

Version 8.1 SP4
December 2004

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Table of Contents

Upgrading Workshop Applications

The purpose of this guide is to help you upgrade web service applications that you created in WebLogic Workshop 7.0 to WebLogic Workshop for WebLogic Server 8.1. The WebLogic Workshop team has added many powerful new features to WebLogic Workshop to make it easier than ever for you to build enterprise applications in Java. In the process we have tried to maintain backward compatibility where possible; where that has not been possible, this guide will help you to make the changes needed to move your applications forward.

The following section, New Features in WebLogic Workshop, describes the enhancements to WebLogic Workshop 8.1 that may require you to make changes to your application in order to upgrade it. If you are looking for a step−by−step description of the migration process, you can skip directly to Migrating Your Existing Application.

Note: This guide is not specifically intended to help you upgrade from WebLogic Workshop 8.1, Beta edition, to the released version of WebLogic Workshop.

New Features in WebLogic Workshop

A number of new features in WebLogic Workshop 8.1 may require you to make changes to your application in order to get it working. Fortunately, most of the changes are not major.

Application and Project Model

In WebLogic Workshop 7.0, you worked with a single project within the Workshop IDE, and the project corresponded to a directory beneath your WebLogic installation.

WebLogic Workshop 8.1 introduces a new application model. Now when you develop in Workshop, you first create a new application. The Workshop application is actually an exploded J2EE application, and the whole application is deployed to WebLogic Server as an EAR file. The Workshop application is managed by a .work file, which contains settings for the application and the projects it contains. When you create a new application, Workshop creates a corresponding directory for that application, and the .work file resides within that directory.

Within your application you can have multiple projects. You can create any of a number of different types of projects, including web projects for building web services and web interface applications; Java control projects for building custom, distributable controls, Java projects for building libraries; and schema projects for generating XMLBeans classes from XML schema files. Other WebLogic Platform components, like WebLogic Portal and WebLogic Integration, provide their own project types.

Control Model

WebLogic Workshop 8.1 introduces a new, extensible model for controls. You can connect to enterprise resources using the built−in controls provided with Workshop, just as you could in Workshop 7.0. And you can also now create your own custom controls, which are reusable and redistributable. Custom controls make it easy to encapsulate business logic within your application.

In WebLogic Workshop 7.0, control files had the CTRL extension. WebLogic Workshop 8.1 uses a different control model, and control files that you create in it have the JCX extension. WebLogic Workshop 8.1

continues to support the CTRL extension, so you can choose whether or not to upgrade your controls to the new model.

If you upgrade your existing controls to the new model, you may need to change some annotations within the control file and within the JWS that calls it. For example, to refer to a JCX control, you must use the @common:control annotation, rather than the @jws:control annotation that was used in WebLogic Workshop 7.0. Also, most annotations within a JCX control file now begin with the @jc prefix.

To find out more about specific annotation changes, place the cursor on the annotation in Source view and press F1 for context−sensitive help.

Autogenerated Service Controls

You may need to update service controls that are autogenerated from a JWS file, because autogeneration for CTRL files is no longer supported.

Controls Must Be in Packages

Controls of type JCX are required to be part of a package in a project. That is, a JCX control file cannot exist at the root of a project within your application. It must be within a folder. A control of type CTRL can continue to exist at the root of the project, but if you update it to a JCX file, you must move it into a package.

Changes to the JMS Control

If you upgrade a JMS control to a JCX file from a CTRL file, you must change a number of annotations on the control. XML maps are no longer directly supported on a JMS control, but there is an additional annotation available for backward compatibility for JMS headers, properties, and messages. The prefix for this set of annotations is @v1jms.

For more information on updating the JMS control for use in WebLogic Workshop 8.1, see JMS Control.

Modules

In WebLogic Workshop 7.0, you included EJB JAR files and other deployable modules used by your project in the project's /WEB−INF/lib folder. In WebLogic Workshop 8.1, you can add these JARs to your application as application−level modules. The Modules folder appears at the root of your application in the Workshop Application pane. When you add a module to the Modules folder, it is automatically deployed to WebLogic Server.

In the file system, the Modules folder corresponds to the application root.

Libraries

In WebLogic Workshop 7.0, you included library JAR files used by your project in the project's /WEB−INF/lib folder. In WebLogic Workshop 8.1, you can add these JARs to your application as application−level libraries. The Libraries folder appears at the root of your application in the Workshop Application pane. Code in library JARs that you add to the Libraries folder is available to all projects in your application.

In the file system, the Libraries folder corresponds to the /APP−INF/lib directory at the root of the application.

External Map Files

External XML map files are no longer supported in WebLogic Workshop 8.1. You can move your XML map information into local XQuery maps or XML maps. XML maps have been deprecated in WebLogic Workshop 8.1 in favor of XQuery maps, but they are still supported.

Migrating Your Existing Application

This section outlines the steps you should take to upgrade your existing application to WebLogic Workshop 8.1.

The first thing to do before you begin migrating your application is to make a backup copy of the full application, in case you need to start again.

Run the jwsUpgrade Command−line Tool

WebLogic Workshop includes a command−line tool to perform a number of migration steps for you. This tool, jwsUpgrade, is located in the BEA_HOME\weblogic81\workshop\upgrade directory. For more information on using the tool, see jwsUpgrade Command.

Note: In order for jwsUpgrade to update your JWS files, the files must be writable. If the WebLogic Workshop project that you are updating is in a source control system, the files may be read−only. In this case jwsUpgrade will write a message to the log file, jwsUpgrade.log, stating that a FileNotFound exception occurred. You should make your project files writable before running jwsUpgrade.

The jwsUpgrade tool will make the following changes to your JWS files:

- Back the original files up to a directory named /bak at the same level as the project directory
- Checks each JWS class definition to ensure that it implements the interface com.bea.jws.WebService.
- Checks that all static inner classes implement the java.io.Serializable interface.

It also detects the following conditions and provides a warning:

- If there are any .jar files in the WEB−INF/lib directory, jwsUpgrade displays and logs a warning telling the user to move the jar file to the APP−INF/lib directory (which appears in the Workshop Application pane as the Libraries folder) in the new application.
- If there are any .class, .java, or JAR files in the WEB−INF/classes directory, jwsUpgrade displays and logs a warning advising the user to move the files to the APP−INF/classes directory (e.g., the application’s Libraries folder) in the new application.
- If your project uses any external map (.xmlmap) files, jwsUpgrade warns that external maps are not supported in WebLogic Workshop 8.1, and recommends that you incorporate the map information directly into the JWS file.

The tool writes output to a file named jwsUpgrade.log in the BEA_HOME\weblogic81\workshop\upgrade directory.

Create a New Workshop Application

After you run the jwsUpgrade tool, the next step is to create a new WebLogic Workshop application and import your project files into it:

1. Launch WebLogic Workshop 8.1.
2. Choose **File−−>New−−>Application** to create a new application.
3. In the **New Application** dialog, select **Default Application**.
4. Specify the directory location and name for the new application. By default the new application will be created in the BEA_HOME\user_projects\applications directory.
5. Select the server on which to create the new application. If you don't have a custom server created, you can create the new application on the default Workshop server. The default home directory for the Workshop server is BEA_HOME\weblogic81\samples\domains\workshop.
6. Click the **Create** button. Your application is created with a basic web project and a Schemas project.
7. In the **Application** pane, right−click the folder for the web project created in Step 5 and choose **Import**.
8. Locate your upgraded project files. Select those folders and files you want to import into the new application. You should exclude the project's /WEB−INF folder and the /bak folder that was generated by the jwsUpgrade tool.
9. Click the Import button.

Your upgraded files are now copied into the new application. If you are importing a lot of files, it may take a few minutes for WebLogic Workshop to import them all and fix up their package names. You may also see a number of warnings stating that control files in the project cannot be updated. This is addressed below in the Update Controls section.

Import Deployable Modules

Next, import any EJBs or deployable modules from the project you are upgrading into the new application:

1. In the **Application** pane, right−click on the **Modules** folder and choose **Add Module**.
2. Navigate to /WEB−INF/lib within your existing project and select the EJB jar files.
3. Click the **Open** button.

The EJB jars will be added to your application as modules. If WebLogic Server is running, they will be immediately deployed to the server; if it is not, they will be deployed the next time the server is started.

If you try to import a JAR file that is not a deployable module, you will get an error stating that it wasn't a J2EE module and couldn't be imported. In this case, import that JAR file as a library rather than as a module.

Import Local Library JAR Files

Next, import any local libraries (i.e., jars that were not deployed to the server) from the project you are upgrading into the new application. In the previous version of WebLogic Workshop, you could include library jar files directly within a package in your project as well in the /WEB−INF/lib folder. In this version, you should import these libraries into the Libraries folder:

1. In the **Application** pane, right−click on the **Libraries** folder and choose **Add Library**.
2. Navigate to /WEB−INF/lib within your existing project and select any library jar files.
3. Click the **Open** button.
4. If you have library jars stored in folders other than WEB−INF/lib, import these in the same way.

If the project you are upgrading has any class files in the WEB−INF/classes directory, you should copy these files manually to the new application.

Upgrade Controls

In WebLogic Workshop 7.0, control files had the CTRL extension. WebLogic Workshop 8.1 uses a different control model, and control files that you create in it have the JCX extension. WebLogic Workshop 8.1 continues to support the CTRL extension. However, it is recommended that you upgrade your existing controls to the new JCX model, so that you can take advantage of this model's greater flexibility.

Note that in WebLogic Workshop 8.1, all control files must live in a package; a control cannot reside at the root of the project. If you have control files at the project root, move them into a folder beneath the project root.

The following section describes the steps to upgrade a CTRL file to a JCX file.

To upgrade a CTRL file to a JCX file:

1. Rename the extension of the file from .ctrl to .jcx. To rename, right−click the file in the **Application** pane and choose **Rename**.
2. A JCX file must extend the interface com.bea.control.ControlExtension. You may need to modify the JCX control file in Source View to ensure that it extends this interface. You can simply add extends com.bea.control.ControlExtension to the class signature .
3. Most annotations in a JCX file have changed to start with the prefix @jc:. You will need to modify annotations in your control file to use the @jc: prefix rather than the @jws: prefix.
4. The annotation used to designate to a control variable has changed to @common:control. In all JWS files that refer to the control class, change the annotation above the declaration from @jws:control to @common:control.

The following sections provide specific information for upgrading built−in controls:

Database Controls

If a Database control returns an object of type java.util.Iterator, and if that type is defined in a separate class rather than in an inner class, you need to qualify the type specified in the iterator−element−type attribute of the @jc:sql annotation with the package name for that type. For example, if your method returns an object of type Customer which is defined in the database package, you need to qualify it with the package name, as shown in the following fragment:

```
/**
* @jc:sql statement="SELECT * FROM customer"
* iterator−element−type="database.Customer"
*/
public java.util.Iterator findAllCustomersIterator();
```

Web Service Controls

WebLogic Workshop no longer supports autogeneration for CTRL files. If you have a Web Service control that is auto−generated from a JWS file, you must upgrade the control in order for auto−generation to work. When you import an autogenerated Web Service control, or when you modify the JWS to which it is linked, Workshop will warn you that these files no longer support autogeneration.

You can upgrade an autogenerated Web Service control simply by right−clicking the JWS file in the **Application** pane and choosing **Generate Service Control**. You can then delete your existing CTRL file.

JMS Controls

The JMS control has changed substantially in WebLogic Workshop 8.1. The following points summarize the changes to the JMS control. For more information on these changes, see JMS Control.

- XML maps are no longer supported on a JMS control. You can now use XMLBeans to construct the body of a JMS message. For more information, see Specifying the Message Body.
- Only one parameter can be used to specify the JMS message body in a method that sends a JMS message, and only a single parameter can be used to receive the message body in a callback that receives a JMS message. You can use an XMLBeans type for this parameter. The @jws:jms−message annotation is no longer valid. For more information, see Specifying the Message Body.
- The @jws:jms−header annotation is now the @jc:jms−headers annotation. The @jc:jms−headers and @jc:jms−property annotations have new syntax for specifying JMS headers and properties. See Specifying Message Headers and Properties for more information.
- You must now specify the message style (topic or queue) separately for outgoing and incoming JMS messages. The @jc:jms annotation has a send−type attribute and a receive−type attribute; you must specify one or both for the JMS control. In WebLogic Workshop 7.0, the @jws:jms annotation had a single type attribute which you used to specify both the send and receive message style; this attribute is no longer valid.

Move Inner Classes into Separate Class Files

If you have inner classes in a JWS file that are called from a JCX control or a JSX script file, you should move these into separate Java class files. Inner classes in a JWS are no longer visible to JCX and JSX files.

In general it's good programming practice to move your inner classes into separate free−standing classes.

Incorporate External Map Files into JWS Files

WebLogic Workshop 8.1 no longer supports external map files (.xmlmap files). If you have external map files, incorporate the map logic directly into the JWS file by adding it to the @jws:parameter−xml or @jws:return−xml annotation of the web service operation that uses the map.

Build the Project

You can find remaining any syntax errors by building the project. To build, right−click the project folder in the Application pane and choose Build <*projectname*>.

Related Topics

jwsUpgrade Command