BEA WebLogic Workshop™ Help

Version 8.1 SP4
December 2004

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Table of Contents

Command Reference

The following are commands associated with WebLogic Workshop.

Topics Included in This Section

jwsUpgrade Command

Updates web service (JWS) files created in WebLogic Workshop 7.0 to include changes necessary for them to compile in WebLogic Workshop 8.1.

wlwBuild Command

Pre−compiles one or more projects, producing an EAR file that may be deployed to a production server.

startWebLogic Command

Starts WebLogic Server from the command line, allowing specification of options.

xmlbean Ant Task

Compiles a set of XSD and/or WSDL files into XMLBeans.

jwsUpgrade Command

The jwsUpgrade command updates web service (JWS) files created in WebLogic Workshop 7.0 to include changes necessary for them to compile in WebLogic Workshop 8.1.

The jwsUpgrade command is available in the BEA_HOME\weblogic81\workshop\upgrade directory. Note that you must run the command from this directory.

For more information on upgrading your application to WebLogic Workshop 8.1, see Upgrading Workshop Applications.

Syntax

```
jwsUpgrade [−help] [−p{roject} <project−directory>] [−a{ll}] [−x{exclude} <file−name>] [−v{erbose}] [<file−name>]
```

Arguments

−help

Prints help for this command.

−project <project−directory>

Specifies the project directory to upgrade.

−all

Specifies that all JWS files in the project should be upgraded.

−exclude <file−name>

Specifies a file that should be excluded from the upgrade process.

−verbose

Prints upgrade information to the console.

<file−name>

Specifies a specific JWS file to upgrade.

Remarks

You can use the jwsUpgrade command to upgrade JWS files that were created in the previous version of WebLogic Workshop. Before it makes any modifcations to your project, jwsUpgrade creates a /bak directory at the same level as the project and copies the original, unmodified JWS files to that directory. It then modifies the original JWS files. No other file types in the project are modified.

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 command modifies the JWS class file in the following ways:

- It modifies the JWS class definition to inherit from the com.bea.jws.WebService interface.
- It modifies any static inner classes to inherit from the java.io.Serializable interface.

The jwsUpgrade command writes all output, including extended error information, to the jwsUpgrade.log file in the BEA_HOME\weblogic81\workshop\upgrade directory. If you include the −v flag, upgrade information is written to the console, including error information, but you may have to look at the log file to see extended error information.

The jwsUpgrade command also alerts you to the following conditions, in the console if the −v flag is specified, and otherwise in the log file:

- WebLogic Workshop no longer supports external map files. If the project you are upgrading contains an external map file, jwsUpgrade will recommend that you integrate the map directly into the JWS file.
- If there are JAR files in the WEB−INF/lib directory of your project, jwsUpgrade will recommend that you move them to the APP−INF/lib directory in your upgraded application.
- If there are .java files or Java classes in the WEB−INF/classes directory of your project, jwsUpgrade will recommend that you move them to the APP−INF/classes directory in your upgraded application.

The following is an example of how you might run the jwsUpgrade command to upgrade all JWS files in a project created in WebLogic Workshop 7.0, with verbose output:

```
jwsUpgrade −p c:\WebServiceProject −a −v
```

Related Topics

Upgrading Workshop Applications

startWebLogic Command

Starts WebLogic Server in a specific domain. Each domain typically has its own dedicated start up script. For example scripts see the sample domains: [BEA_HOME]\weblogic81\samples\domains\workshop\startWebLogic.cmd, [BEA_HOME]\weblogic81\samples\domains\portal\startWebLogic.cmd, etc. Upon creation of a new domain, startup scripts are provided in the domain directory.

Syntax

startWebLogic ***[noiterativedev]*** [notestconsole] [production]

Parameters

noiterativedev

Optional. When this flag is present, WebLogic Workshop's iterative development mode is turned off. Default: iterative development mode is turned on.

notestconsole

Optional. When this flag is present, the WebLogic Workshop test harness is not available. Also, logging is disabled, which may increase performance depending on logging configuration. Default: the test harness and logging are enabled.

production

Optional. Specifies that WebLogic Server should be run in production mode. Default: development mode.

WebLogic Workshop Modes

WebLogic Workshop can be run with *iterative development mode* set to on or off. When iterative development mode is off, only applications that are packaged into a deployed EAR file may be accessed by clients. Applications that are in exploded form in the Application pane are not accessible. When iterative development mode is on, the opposite is true: applications in exploded form are accessible and applications in deployed EAR files are not. In addition, when iterative development mode is off the web page for each web service is limited to just the Overview pane; the Console, Test Form and Test XML panes are not available.

WebLogic Server Modes

WebLogic Server may be run in either *development mode* or *production mode*. This mode is controlled by the weblogic.ProductionModeEnabled Java property. In development mode, WebLogic Server's application poller is active, meaning WebLogic Server will automatically discover new applications (new WebLogic Workshop projects) that are created and populated. In production mode, the application poller is disabled.

Note: You can read more about starting WebLogic Server in the topic Starting and Stopping Servers: Quick Reference in the WebLogic Server 8.1 documentation.

Remarks

The startWebLogic command starts WebLogic Server in a specific domain. A typical WebLogic Server installation may contain many configured domains. Each domain's configuration is controlled by the config.xml file. startWebLogic should be run from the directory containing the config.xml file for the domain whose server you wish to run.

wlwBuild Command

Builds an application into an EAR file for deployment to a production server. Optionally, this command can build a single project into a JAR file, provided the project type does not include web application resources.

Syntax

```
wlwBuild
[−p <project−name>]
[−od <output−directory>]
[−of <output−file>]
[−scp <server−classpath>]
[−f <build−properties−file>]
[− clean]
[<application−file>.work]
```

Arguments

−p, −project

Specifies the name of the project in the application that should be built. If this flag is omitted, the entire application will be built.

−od, −outputdirectory

Specifies the directory where the output EAR file will be placed. If this flag is omitted, the EAR is placed in the application's root directory.

−of, −outputfilename

Specifies the name of the file that should be produced.

−scp, −serverclasspath

Specifies the server classpath to use in the build. Defaults to the classpath of the application's selected server. This flag is required if the the application's server's home directory cannot be found.

−f, −file

Specifies the property file containing build options. The property file should use the full names of the flags above, e.g., "−serverclasspath=C\:bea\weblogic...".

−clean

Cleans all previous build output for a single project or the full application.

<application−file>.work

Specifies that application configuration parameters from the specified .work file should be used.

Remarks

wlwBuild is located in the BEA_HOME\weblogic81\workshop directory.

Related Topics

Deploying Applications

How Do I: Call wlwBuild.cmd from an ANT build.xml file?

How Do I: Use a Custom Ant Build for a Project?

xmlbean Ant Task

Compiles a set of XSD and/or WSDL files into XMLBeans. This is useful for building an XMLBean JAR from XSD and WSDL files. If desired, the task can also generate the source code that makes up the XMLBean type system specified by the schema files.

Note: This task depends on an external library not included in the Ant distribution called xbean.jar. It can be found in the XMLBeans developer kit at http://dev2dev.bea.com/technologies/xmlbeans/. The build script will need to include a taskdef for xmlbean, which could look like this:

```
<taskdef name="xmlbean" classname="com.bea.xbean.tool.XMLBean" classpath="path/to/xbean.
```

Note: Note that this task is an Ant alternative to compiling schemas using a WebLogic Workshop schemas project. For more information about using a schemas project, see How Do I: Use XML Schema in WebLogic Workshop?

It is possible to refine the set of files that are being processed. This can be done with the includes, includesfile, excludes, excludesfile and defaultexcludes attributes. With the includes or includesfile attribute you specify the files you want to have included by using patterns. The exclude or excludesfile attributes are used to specify the files you want to have excluded. This is also done with patterns. And finally with the defaultexcludes attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks in the Ant documentation, on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet and supports all attributes of `<fileset>` (`dir` becomes `basedir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

Parameters

Attribute	**Description**	**Required**
schema	A file that points to either an individual schema file or a directory of files. Not a path reference. If multiple schema files need to be built together, use a nested fileset instead of setting schema.	Yes, unless a fileset element is nested.
destfile	Define the name of the jar file created. For instance, "myXMLBean.jar" will output the results of this task into a jar with the same name.	No, default is "xmltypes.jar".
download	Set to true to permit the compiler to download URLs for imports and includes. Defaults to false, meaning all imports and includes must be copied locally.	No, default is false.
failonerror	Determines whether or not the ant target will continue if the XMLBean creation encounters a build error.	No, default is true.
verbose	Controls the amount of build message output.	No, default is true.
typesystemname	The name of the package that the TypeSystemHolder class should be generated in. Normally this should be left unspecified. None of the XMLBeans are generated in this package. Use .xsdconfig files to modify XMLBean package or class names.	No

classgendir	Set a location to generate CLASS files into.	No
srconly	A value of true means that only source will be generated.	No, default is false.
srcgendir	Set a location to generate JAVA files into.	No
classpath	The classpath to use if schemas in the fileset import definitions that are supplied by other compiled XMLBeans JAR files, or if JAVA files are in the schema fileset. Also supports a nested classpath.	No
classpathref	Adds a classpath, given as reference to a path defined elsewhere.	No
includes	Comma− or space−separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	The name of a file. Each line of this file is taken to be an include pattern.	No
excludes	Comma− or space−separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	The name of a file. Each line of this file is taken to be an exclude pattern.	No
defaultexcludes	Indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
debug	Indicates whether source should be compiled with debug information; defaults to `off`. If set to `off`, `−g:none` will be passed on the command line for compilers that support it (for other compilers, no command line argument will be used). If set to `true`, the value of the `debuglevel` attribute determines the command line argument.	No
debuglevel	Keyword list to be appended to the `−g` command−line switch. This will be ignored by all implementations except `modern` and `classic(ver >= 1.2)`. Legal values are `none` or a comma−separated list of the following keywords: `lines`, `vars`, and `source`. If `debuglevel` is not specified, by default, nothing will be appended to `−g`. If `debug` is not turned on, this attribute will be ignored.	No
optimize	Indicates whether source should be compiled with optimization; defaults to `off`.	No
includeAntRuntime	Whether to include the Ant run−time libraries in the classpath; defaults to `yes`.	No
includeJavaRuntime	Whether to include the default run−time libraries from the executing VM in the classpath; defaults to `no`.	No
fork	Whether to execute `javac` using the JDK compiler externally; defaults to `yes`.	No, default is true
executable	Complete path to the `javac` executable to use in case of `fork="yes"`. Defaults to the compiler of the Java version that is currently running Ant. Ignored if `fork="no"`	No
memoryInitialSize		No

	The initial size of the memory for the underlying VM, if `javac` is run externally; ignored otherwise. Defaults to the standard VM memory setting. (Examples: `83886080`, `81920k`, or `80m`)	
memoryMaximumSize	The maximum size of the memory for the underlying VM, if `javac` is run externally; ignored otherwise. Defaults to the standard VM memory setting. (Examples: `83886080`, `81920k`, or `80m`)	No
compiler	The compiler implementation to use. If this attribute is not set, the value of the `build.compiler` property, if set, will be used. Otherwise, the default compiler for the current VM will be used.	No

Example

Be sure to define the task in your script, like this:

```
<taskdef name="xmlbean" classname="com.bea.xbean.tool.XMLBean" classpath="path/to/xbean.jar"
```

The following builds all the schemas in the schemas directory and creates a jar called "Schemas.jar".

```
<xmlbean schema="schemas" destfile="Schemas.jar"/>
```

The following compiles the schema "ourSchema.xsd" into the default jar "xmltypes.jar". If any imports and includes are defined by remote URLs, they are downloaded during the build.

```
<xmlbean schema="schemas/ourSchema.xsd" download="true"/>
```

Using a fileset

```
<xmlbean classgendir="${build.dir}" classpath="${class.path}"
      failonerror="true">
  <fileset basedir="src" excludes="**/*.xsd"/>
  <fileset basedir="schemas" includes="**/*.*"/>
</xmlbean>
```

Gathers all the files in the src directory except XSD files, along with every file in the schemas directory, and compiles them. The fileset can include schema files that refer to previously compiled schema components. The fileset can also contain JAVA files. The classpath parameter defines the classpath necessary to resolve compiled schema and java references.

The built classes will go into ${build.dir}.

Related Topics

How Do I: Use XML Schema in WebLogic Workshop?