BEA WebLogic Integration™

Tips and Tricks

Version 9.2
Document Date: April 2007

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

Tips and Tricks for WebLogic Integration

Tips and Tricks for WebLogic Integration

There are a few, significant differences in developing applications between the Workshop-based WebLogic Integration (WLI) 8.x IDE and the Eclipse-based WLI 9.2 IDE. This document provides tips and tricks to users familiar with the WLI 8.x environment, which help them transition to developing applications in the WLI 9.2 IDE.

This document includes the following sections:

Differences between WLI 8.x IDE and WLI 9.2 IDE

The Integrated Development Environment (IDE) for WLI 9.2 is based on BEA Workshop for WebLogic Platform 9.2, which uses Eclipse 3.1.1. It is different from the WLI 8.x IDE as described in the following sections:

Project Structure

WLI 8.x applications included artifacts and other subprojects like schema projects, web projects, and ejb projects and they were hierarchical in nature. In 9.2, WLI applications have a flat organization and consist of an EAR project, one or many Web projects and one or many Utility projects.

Each project maintains references to other related projects in WLI 9.2. When a WLI 9.2 application is created through the Process Application Wizard, these references are already established. If projects are imported in WLI 9.2, then such references should be manually updated (see Managing Project Dependencies).

In a WLI 9.2 application, an EAR project is the central point of the application. It is used to create EAR (Enterprise Archive) files and includes:

- JAR files that are shared by the projects in the enterprise application
- Links to all of the projects in the application used by Workshop

Note: Libraries that need to be available to any project in the application should be stored in `<EAR project>/EarContent/APP-INF/lib`.

In a WLI 9.2 application, a Web project includes Processes, Controls, XQ files, Message Broker, and transformation files. A Utility project includes schemas, xml, and WSDLs.

When a project is created, you are required to do some or all of the following:

- Specify the type of project
- Add standard libraries
- Set compiler options
- Control publishing tasks
- Set build path or add an annotation processor.

The above options are specified by choosing facets during project creation. Basically each project type has its own facets, which assigns the corresponding builders, validators. Facets can also be added and deleted from a project after its initial creation. To edit a project's facets, select **Project > Properties > Project Facets**.

Note: A process can be created only in a WEB project which has Weblogic Integration Process facet (process-enabled) added to it. Similarly a Worklist task plan can be created in an EAR project which has Worklist Integration Worklist Application Module facet (worklist-enabled) added to it.

Table 1 list various artifacts and their locations.

Table 1 Artifacts

WLI Components	Artifacts	Project	Folder/Package
Process	xxxProcess.java	Web	src/processes package
Controls	xxxCtrl.java	Web	src/controls package
Data Transformation	xxxDtf.java	Web	src/processes package
Web Services	xxxJws.java	Web	src/processes package
Message Broker	xxx.channel	Web/Util	src/processes package
Task Plans	xxx.taskplan	Ear	EarContent/<folder>
XML Schema	xxx.xsd	Web/Util	src/schemas package
WSDL	xxx.wsdl	Web/Util	src/schemas package
XQ	xxx.xq	Web	src package

Eclipse-based Projects

WLI 9.2 Applications and their projects depend on Eclipse. Unlike WLI 8.x Applications, you cannot move projects from one system to another by merely copying projects. In 9.2, the projects have to be imported into the workspace using the Import wizard and selecting the **Existing Projects into Workspace** option.

File Extensions

In WLI 8.x, the various types of WLI artifacts had their own file extensions like .jpd, .dtf, .jcx. In 9.2, unique file extensions for specialized java files like .jpd, .dtf, .jcx no longer exist. All file extensions end with .java (for example, process.java). Therefore, while creating WLI 9.2 artifacts it is a good practice to name Process files, controls, and data transformations in such a way that each can be identified easily.

Note: XQuery files have .xq extensions in 9.2.

No Wizard for Some Controls

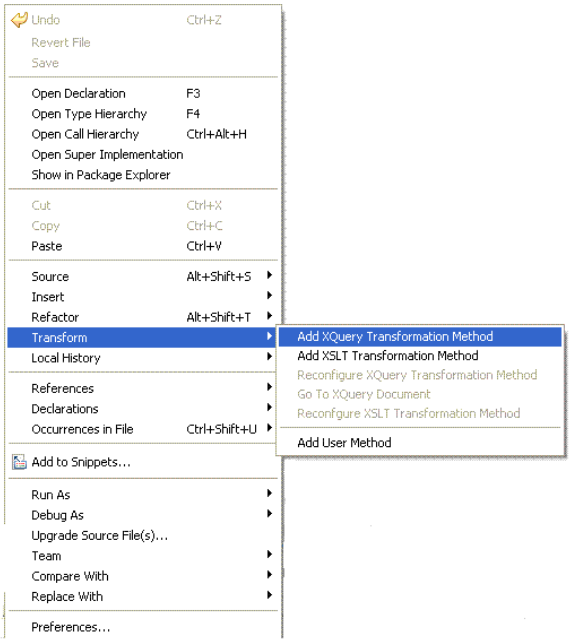
In WLI 9.2, some of the Workshop controls (for example, JDBC or Timer) no longer have wizards. You must use the **Source** view or Annotation view to configure them.

DTF Design View not Available

In WLI 9.2, there is no DTF design view. Only a **Source** view is available.

To configure a Transformation method, right-click **Source** view, then select **Transform**, and then select one of the options (see Figure 1).

Figure 1 Configure Transformation Method



To reconfigure a Transformation method, open the transformation file and in **Source** view, select the transformation method and then right-click, select **Transform > Reconfigure XQuery Transformation Method**.

XQuery

WLI 8.x supports XQuery 2002. WLI 9.2 supports XQuery 2004, and XQuery 2002 support is extended for backward compatibility. Upgrade of an Xquery file from 8.x to 9.2 may not be successful due to the incompatibility between the XQuery specifications. In such cases, you are

expected to manually correct the upgraded xquery file (see Updating XQuery Use to Support XQuery Implementation).

Iterative Development

In WLI 9.2, if a Process file's interface is modified, then you need to recreate all the other dependant artifacts (such as process controls that are generated out of this Process file, WSDLs generated from this Process file). In addition, iterative development does not automatically redeploy the application as in WLI 8.x.

In the case of Task Controls, it is very necessary to regenerate the task control each time you make a change to the Task Plan and reuse it in the Process.

Standards

WLI 8.x uses the Javadoc-comment style annotations. In WLI 9.2, Java 5 annotations are used; therefore, when upgrading a WLI 8.x application, the annotations in the application are converted to 9.2 annotations. Some of the 8.x annotations are deprecated, because of which the conversion to WLI 9.2 annotations may cause error. You have to manually correct them (see Upgrading Annotations).

Upgrading Sample Application System Schemas

The WLI 8.x Schema Builder allows you to a build sample application even though the namespace is not specified. The WLI 9.2 Schema Builder does not support empty namespaces. If you import the `envelope.xsd` from WLI 8.x, it will lead to some errors.

To fix these errors do one of the following

- Import the schema from the WLI 9.2 Sample Application
- After importing the 8.x schema, change it from
  `xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">` to
  `xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">`
  `schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"/>`

  Note: Import namespace statements should be added.

Mapper Test View

The XQuery Mapper test view in WLI 8.x is not a standalone tester and always requires a running WLI server. Where as, the XQuery Mapper test view in WLI 9.2 is a standalone tester and does not runs inside a Process file container. (It does not matter if there is a WLI 9.2 server running, the XQuery Mapper tester does not use it). This is by design, because the XQuery Mapper is used in other BEA products outside of WLI.

Performance Setting for Complex Application

If you are developing a complex application with high consumption of memory, change your JVM setting by changing the memory parameters in the file **`workshop4wp.ini`** available at your *BEA_HOME* directory (for example, `C:\bea\workshop92\workshop4WP.ini`) as follows:

- Xms384m: Do not change any setting
- Xmx768m: Change to 1280m (1.25 GB)

Note: The above settings are only a recommendation; you can calculate settings best suited for your machine (see Configuration Requirements).

Guidelines

Guidelines While Creating WLI Application

1. In case you have to work on a large Process, it is a good practice to divide it into a number of subprocesses (Each subprocess is also a process in itself). Individual developers can work on each subprocess. Later, all subprocesses can be called through a central process.
2. WLI 9.2 provides a project wizard which helps you to create projects required for Process file and Worklist applications. In many applications, your business process may interact with a task plan. In such scenarios, it is a good practice to create a project and then add a Worklist facet to your project as follows:
   - Right-click on your Ear Project, go to **Properties > Project Facets > Add and Remove Project Facets**, and select **WebLogic Integration Worklist Application Module** and then click **Finish**.
   - Right-click on your Web Project, go to **Properties > Project Facets > Add and Remove Project Facets**, and select **WebLogic Integration Worklist Application Module** and **Beehive NetUI 1.0** and then click **Finish**.

- Create a folder under your **Ear Project > EarContent**, and create the task plans inside the EarContent folder.

Once you add a worklist facet to a process project, you can create a Process file and a task plan in the same application.

3. Whenever you move to a specific artifact (Process file, XQ, or Worklist), ensure that you are in the appropriate perspective (Process, XQuery Transformation, and Task Plan respectively). For example, navigating to an XQuery file from a Process file does not automatically change the perspective to the XQuery Transformation perspective.

4. If you are working with CVS, ensure the following:

   Do not check in the following directories to source control. (Not all of these directories and files will exist in every project.)

   - .metadata (workspace-level)
   - build (project-level folder; In every web and util project, the build folder should be excluded)
   - templib (project-level; in every web and util project, the temp lib folder, if present, should be excluded)
   - .apt_src (project-level; in every web and util project, the .apt_src folder should be excluded)
   - .xbean_src and .xbean_bin (project-level; only for projects with XMLBeans Builder enabled)

   The following directories and files should be included in source control. (Not all of these directories and files will exist in every project.):

   - .settings/* (project-level)
   - .classpath (project-level)
   - .factorypath (project-level)
   - .project (project-level)
   - .datasync-project (project-level)

   When you try to use this project from another machine, perform the following steps,

   - Open an empty workspace.
   - Import the Application by right clicking and selecting **Import > Existing projects into workspace**.

- Select the workspace (which is the top folder), all project folders would be listed in the dialog-box.
- Select the required folders and click **Ok**, all folders will be imported, and then start building.

  Note: Occasionally you might see some errors like the library module reference: `beehive-controls-1.0` is on the classpath of a dependent project, but it is not included in this EAR project. In this case, clean and build again. If it does not works, switch to the same workspace and build, the errors will go off.

5. During the development process, it is quite possible that you may have to transfer resources from one developer's system to another developer's system. For such transfers, you make a portable ZIP file. If you are making a Portable Workspace ZIP file (contains workspace and project) manually or through an Ant task, make sure to exclude these directories:
   - .metadata (workspace-level)
   - build (project-level folder; in every web and util project, the build folder should be excluded)
   - templib (project-level; in every web and util project, the temp lib folder, if present, should be excluded)
   - .apt_src (project-level: in every web and util project, the .apt_src folder should be excluded)
   - .xbean_src and .xbean_bin (project-level; only for projects with XMLBeans Builder enabled)

   If you are making a portable ZIP file with Workshop for WebLogic, select **File > Export > Archive file > Next**. In the left-hand pane, select the projects you want to include, but unselect the following directories within each project:
   - build
   - templib
   - .apt_src
   - .xbean_src and .xbean_bin (only for projects with XMLBeans Builder enabled)

   To retain the original directory structure when your workspace has multiple projects, make sure to place a checkmark next to **Create directory structure for files**.

   To uncompress the ZIP file and use the workspace, select **File > Import > Existing Projects into Workspace**, then select **Archive file** option and provide the ZIP file (see Opening a Sample Workspace).

Guidelines During Application Development Stage

1. Occasionally, the XML Validator and WSDL validator associated with WTP might show errors on the XMLs/WSDLs available in the projects. Clear the XML Validator and WSDL validator check boxes on WLI-enabled utility project and Web project to disable them.

2. Clicking on the **Transformation > Create Transformation** option in node editors generates a transformation file with a default name (for example, RequestQuoteTransformation.java) and a method with a default name (for example, availProcessorGetAvail). Such transformation files are difficult to share as they have system generated names. If you, want to edit the system generated names, ensure that all references to them are modified accordingly.

   In a node editor, each time you edit the transformation, by modifying its inputs or outputs, a new Transformation method with a new signature and a new XQ file is generated. If you intend to reuse these transformations across processes, it is good practice to develop a separate transformation file containing all node editor-related transformations, then, you can reuse the transformation methods through the **Advanced Options** dialog in the node editor.

3. If some annotation of a particular node is not visible in the Annotation View even after selecting that node in the Design View, try selecting the source code of that node in the source view. This should reveal the expected annotation in the annotation table.

4. Try not to open more than one instance of BEA Workshop for WebLogic Platform.

5. Try using XQuery transformations instead of relying on the XMLBeans API for simple transformations. For example if you want to retrieve EmpID from an EMP XmlBean, instead of using `getEmpID()` get method on that XML Bean object, create a simple XQuery transformation, using the node editor, for extracting EmpID from EMP.

6. If your application is transformation centric, processing large (ie. ~1 MB) schemas files, the XMLBeans Builder can result in unacceptably long build times due to the performance of both the XMLBeans compiler and the Eclipse Java builder. XMLBeans compiler performance can be improved by disabling assertions for the XMLBeans code. Assertions can be disabled by adding the line "`-da:org.apache.xmlbeans`" to the file workshop92/workshop4WP/workshop4WP.ini.

7. There is a known issue when you drag and drop custom methods from the Task Control into the Process Canvas. The suggestion is to go to **Source** view and change `eventSet = tmp.TaskC.Callback`.class to `eventSet = tmp.TaskC.CustomCallback`.class in the `EventHandler` annotation.

Guidelines While Building an Application

1. In the WLI IDE 9.2, the Build process has two modes – Auto and Manual. In Auto Build mode, resources (Projects, folders, files) are built as they are changed and saved. The default option is **Build automatically**. To improve performance you can switch off this option. To disable **Build automatically**, select **Project tab** on your menu bar, and from the drop-down list clear **Build automatically**.
2. It is a good practice to perform a clean build operation once you have completed your development work or if you have build-related problems.
3. Be sure to save your project before you build an application, changes are not saved by default.
4. Closing projects which are not of immediate relevance may improve build performance.

Recommended Reading

For more information, refer to the following documents:

- Introducing BEA WebLogic Integration
- Guide to Building Business Process
- Using Integration Controls
- Guide to Data Transformation
- Building Web Services with Workshop for WebLogic
- Using Worklist

See edocs, for more WLI related information.