Using Kelp with an IDE

Table of Contents

1. Introduction .....
    Kelp features 1
2. Project development overview .....
    Generating or importing source files 3
    Setting project properties 3
    Compiling projects 3
    Building and deploying archive files 3
    Debugging projects 3
3. Using the wizards and tools .....
    Using the Kelp Application Wizard 4
    Enhydra Import Wizard 6
    Using the Kelp XMLC tool 6
        Using mapping tables for generated class names 8
        Setting output options 9
    Using the Kelp Deployer 10
        General Tab 10
        Input Tab 10
        Content Tab 11
        Archive Tab 12
        Run Tab 12
    Using DODS 14
4. Setting project properties in JBuilder .....
    PATH and CLASSPATH settings 16
        Paths page 16
        Output path 16
        Source subtab 16
        Required libraries subtab 17
    Build properties 17
        Generate source to output path option 17
    Run options 18
        Main class option 18
        Application parameters option 18
5. Setting project properties in NetBeans .....
    Classpath 19
    Project directory 19
    Source directory 19
    Output directory 20
    Deploy root 20
    Run options 20
    Setting up a project for use with the Kelp ApplicationWizard 20
6. Setting XMLC properties .....
    XMLC project properties 21
        XMLC node property page 22
        How Kelp sets class names for XMLC 23
        Setting XMLC options for selected files 24
    Input Template node property page 24
7. Kelp sample projects .....
    The sample servlets 25
8. Debugging Kelp projects .....
    Debugging with NetBeans 26
        To debug using SharedMemoryAttach: 26
        To debug using SocketAttach: 26
    Debugging with JBuilder 27

To debug an Enhydra application: 27
9. Kelp for JBuilder differences .....
Using the Enhydra Import Wizard 29
Importing DiscRack 29
Building DiscRack 30
Running DiscRack 30

List of Examples

3.1. Deployment example: deploying a Web application 12

Chapter 1. Introduction

This chapter describes how to use Kelp to develop applications with XMLC for use with Enhydra 5.1. It assumes that you have a basic understanding of Enhydra and either Sun NetBeans for Java, Community Edition, Borland® or JBuilder.

Kelp is a set of tools that extend a Java integrated development environment (IDE) to simplify the development of applications for Enhydra 5.1

Kelp for NetBeans is a set of tools for Sun Netbeans for Java, Community Edition. For information on NetBeans for Java, visit the Sun NetBeans for Java website.

Kelp for JBuilder is a set of tools for Borland JBuilder. For information on JBuilder, visit the Borland website at: http://www.borland.com/jbuilder/ [http://www.borland.com/jbuilder/].

Note: Kelp for NetBeans and Kelp for JBuilder function very similarly. For the sake of clarity, this chapter is written for one IDE, NetBeans for Java, Community Edition. Any notable differences for Kelp for JBuilder are addressed in a separate section at the end of this chapter.

Kelp features

Kelp provides the following tools and features:

- Kelp Application Wizard

  The Kelp Application Wizard generates Web applications using either the Servlet API or the Enhydra programming model.

- XML Compiler integration

  The Kelp XMLC tool lets you set XMLC options, select markup language files (e.g., HTML, XML) to compile, and call XMLC from within the IDE to create classes that generate web content dynamically.

- Kelp Deployer

  The Kelp Deployer allows you to set up your project properties, copy static content to the document root, process templates, create deployable archives, and deploy the archives.

- Enhydra Import wizard

  The Import wizard allows you to import Enhydra projects that use GNU Makefiles into your IDE. The Import Wizard is currently supported in JBuilder only.

- DODS Generator

  Dods generator runs ant based xml files (build_dods.xml and build_java.xml) for generating sql and java files.

- Enhydra XMLC properties

  The XMLC properties give you full control over how XMLC builds Document Object Model (DOM) classes from your HTML files.

- Input Template property pages

  The Input Template property pages let you specify a list of strings to search and replace when you are generating files from templates.

- Build integration

  Through property pages, you can set up JBuilder to invoke XMLC and the Kelp Deployer whenever you make or rebuild your JBuilder project. This feature lets you quickly ensure that your files are updated without having to run the tools individually.

  Note: Build integration is not currently supported by Kelp in NetBeans. To build Enhydra projects with Kelp in NetBeans, you must invoke the Kelp XMLC tool before building the project, and the Kelp Deployer afterwards.

- Kelp sample projects

  These projects demonstrate techniques for creating dynamic Web pages with XMLC Web applications.

Chapter 2. Project development overview

This section outlines how Kelp can be used with an IDE to speed the development process. Using the Kelp tools and wizards together with those of the IDE, you can perform the following basic functions:

Generating or importing source files

Within NetBeans or JBuilder you can generate a new Web Application or Enhydra Application (super-servlet style application), using the Kelp Application Wizard. The generated framework of files and directories provide a useful starting point for developing applications and services for Enhydra 5.1.

If you have an Enhydra application, you can import the application into the IDE using the Enhydra Import Wizard. The Import Wizard sets XMLC properties from the settings in the makefiles. This saves you time and trouble by automating tasks such as setting up the mapping tables to customize generated class names.

Setting project properties

The Kelp project properties consist primarily of XMLC and deployment options. XMLC options can be used to set the name for a generated class file, save the generated Java source-code files, and more. The XMLC options can be specified for a file, a folder, or the project. Project settings will be overridden by folder settings, and folder settings will be overridden by file settings. For additional information about XMLC, refer to Chapter 5, "Enhydra XMLC," of the Developer's Guide.

Deployment properties specify what files to include in the deployable archive file and how to deploy the archive file.

Compiling projects

Once you have generated or added the source files to your project, you can compile the project to generate the necessary classes. If you are using XMLC in your project, you must run the Kelp XMLC tool before compiling the rest of your source files.

Note: Projects compiled or built from within the IDE do not use Ant. If you want to take advantage of some of the advanced Ant capabilities, such as the DODS tasks for automatically generating entity beans, you may want to forget building your project within the IDE.

Building and deploying archive files

After you have compiled your project you can use the Kelp Deployer to create and deploy an archive file. The Kelp Deployer only deploys to mapped drives.

Debugging projects

if you are having trouble running your application, you can use the debugging capabilities of the IDE to isolate the problem. NetBeans for Java supports remote debugging. JBuilder supports debugging for applications running within the IDE. Although the debugging capabilities of the supported IDEs differ, both methods are useful. For additional information refer to "Debugging Kelp projects".

Chapter 3. Using the wizards and tools

Kelp provides wizards and tools that help you develop Enhydra applications from within your IDE.

Using the Kelp Application Wizard

The Kelp Application Wizard, shown in Figure 4.1, speeds up project development by generating the framework and source files for new applications and components. The Kelp Application Wizard uses two generators to create Enhydra projects: the Web Application generator and the Enhydra Application generator.

The Application Wizard uses the following generators:

- Web application (Servlet 2.2 compatible)
- Enhydra application (super-servlet style application)

Creating a new project:

To use the Kelp Application Wizard from within the IDE, create a project beforehand. The NetBeans Project must have a mounted directory into which you will generate your project files.

Important: Running the Kelp Application Wizard while you are using a project that already contains files generated by the Kelp Application Wizard might cause unexpected results. Before you run the Kelp Application Wizard, create a new project for the generated files.

- To generate project file in NetBeans, select File|Kelp Application Wizard to launch the Kelp Application Wizard. Then, choose the component type to generate from the Component Type pull-down menu.

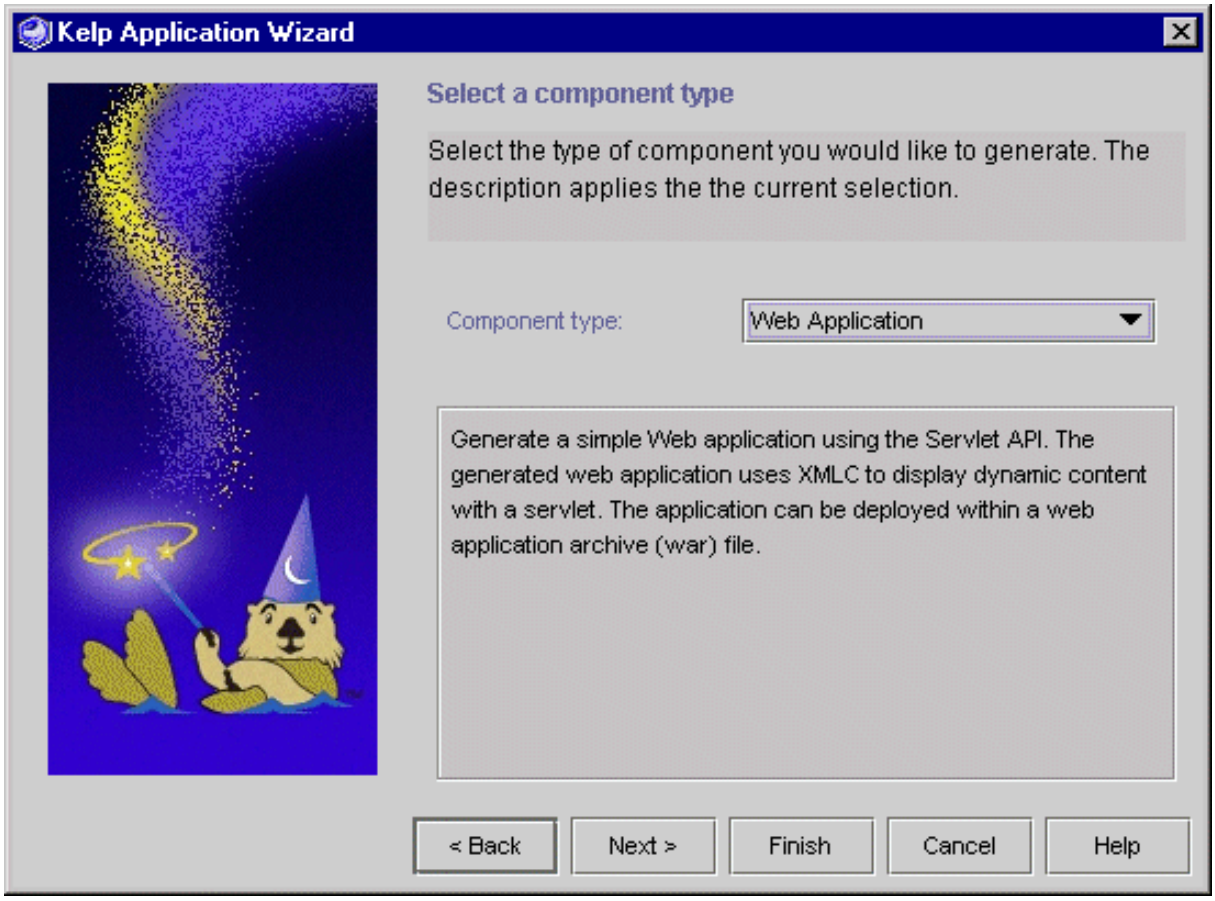

Figure 4.1 Kelp Application Wizard, launched from NetBeans

Note: To create a new project in JBuilder, select File|New to open the Object Gallery as shown in Figure 4.2 . Click the Enhydra tab, select the component type to generate, and click OK. The Object Gallery in JBuilder replaces the first page of the Kelp Application Wizard. The rest of the wizard is unchanged.

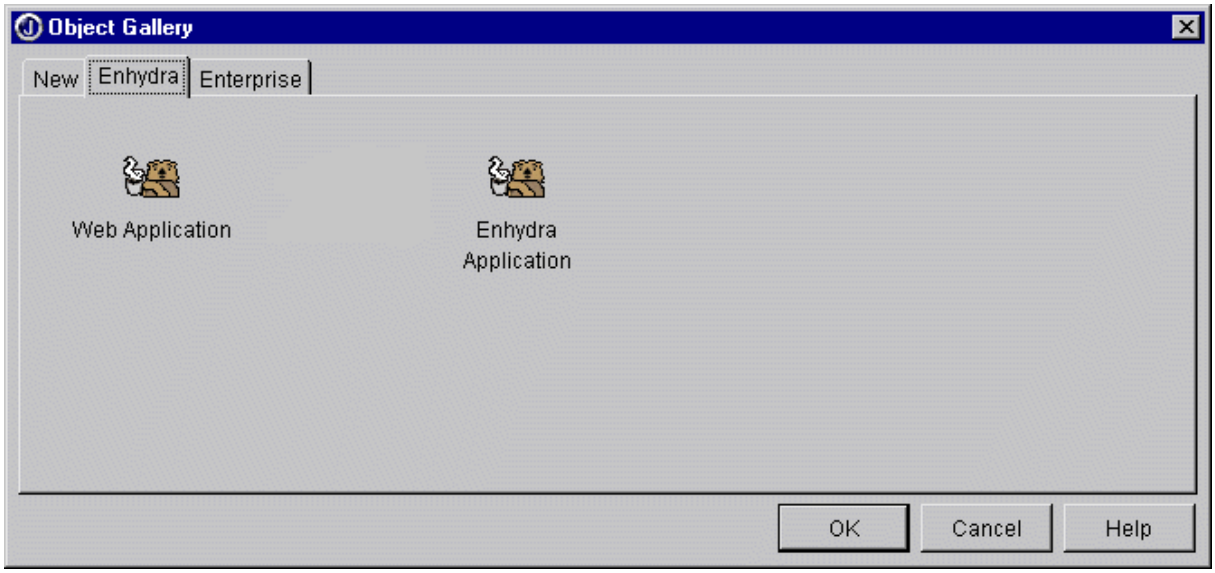

Figure 4.2 Object Gallery in JBuilder, showing choices for generating Enhydra compontents

- Navigate through the wizard and select the options and naming conventions for your genenerated files.
- When you have finished setting the options you want, click Finish to generate your application or service. When generation is complete, view the newly generated Readme.html file that has been added to your project. Readme.html shows the steps you need to build and run the application or service. The steps will vary, depending on which IDE you are using.

For more information on the Kelp Application Wizard, see Chapter 2, "Using the Kelp Application Wizard," of the Developer's Guide. Once you create a project, you can use the Kelp XMLC tool and the Kelp Deployer tool to work with it.

In both the Kelp XMLC tool and Enhydra Deployment wizards, you use a tabbed dialog box to select files, set options, and view the build process.

Enhydra Import Wizard

The Enhydra Import wizard is similar to the Application Wizard. This wizard allows you to convert a GNU Makefile project for an Enhydra Application to an IDE project.

Note: The Enhydra Import Wizard is only supported by JBuilder. For additional information, refer to "Using the Enhydra Import Wizard".

Using the Kelp XMLC tool

The XML Compiler dialog calls XMLC, which compiles HTML and XML files into DOM classes. To open the dialog, select Tools|XML Compiler. The tool is available only when a project is open.

The dialog box has three primary tabs: Selections, Options, and Output. The Selections tab, shown in Figure 4.3, displays all files in the currently selected project that are recognized as compilable by XMLC. You can use the single arrow buttons (< and >) to add or remove files from the list selected to be compiled. Use the double arrow buttons (< < and > >) to add or remove all files from the selection list. Select the Show Full File Path check box to display the full path of the files.

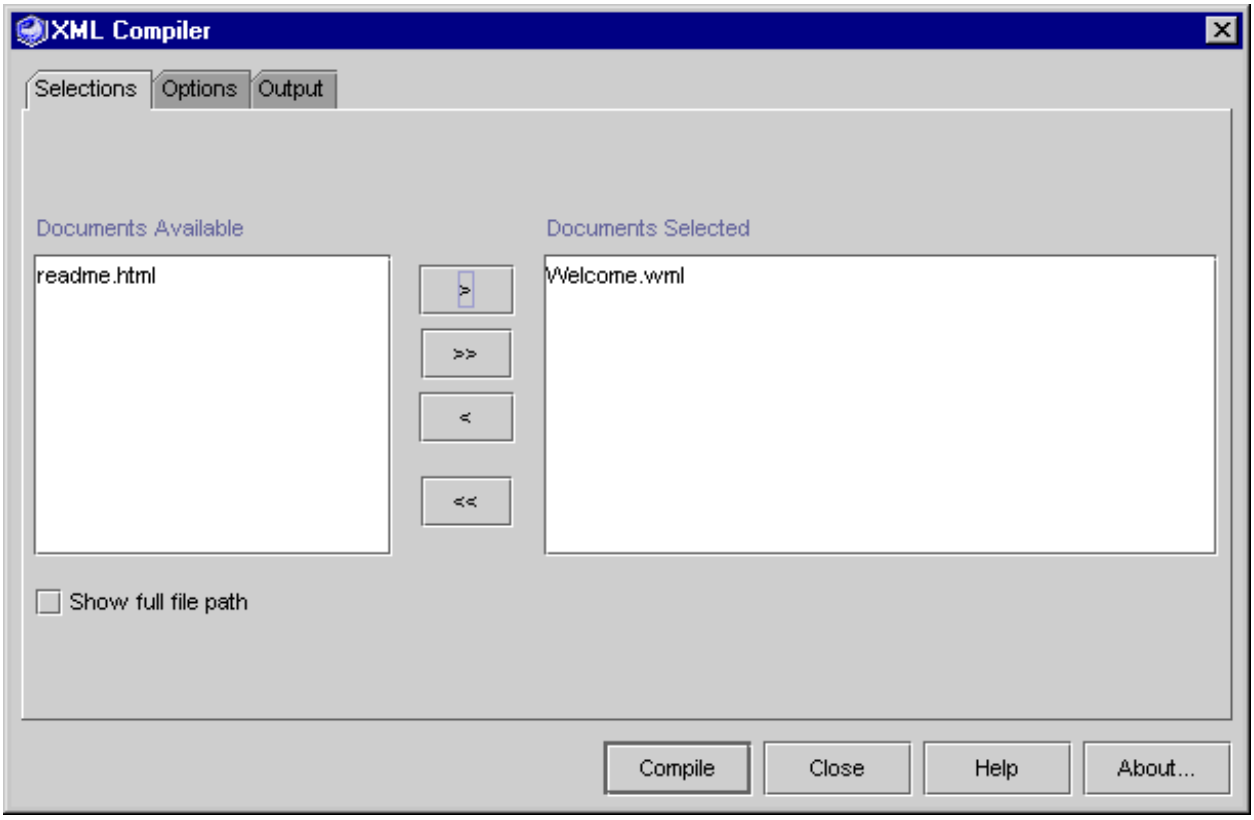

Figure 4.3 Selections tab of the XML Compiler dialog

The Options tab displays additional tabs for compile, XMLC types, and trace options.

- The Compile options let you customize the generated DOM classes.
- The XMLC Types tab allows you to add new file extensions to associate types of files with XMLC. Files with the specified extentions display on the Selections tab and can be selected if you want them to be processed by XMLC.
- The Trace options let you display detailed information during the compile process without affecting the generated classes.

Note: In JBuilder, the Options tab also has an Invoke XMLC during Project Make/Rebuild check box. Select this check box to call XMLC without opening the XML Compiler dialog. This calls XMLC when you build or make a project node that contains a selected XMLC document type file.

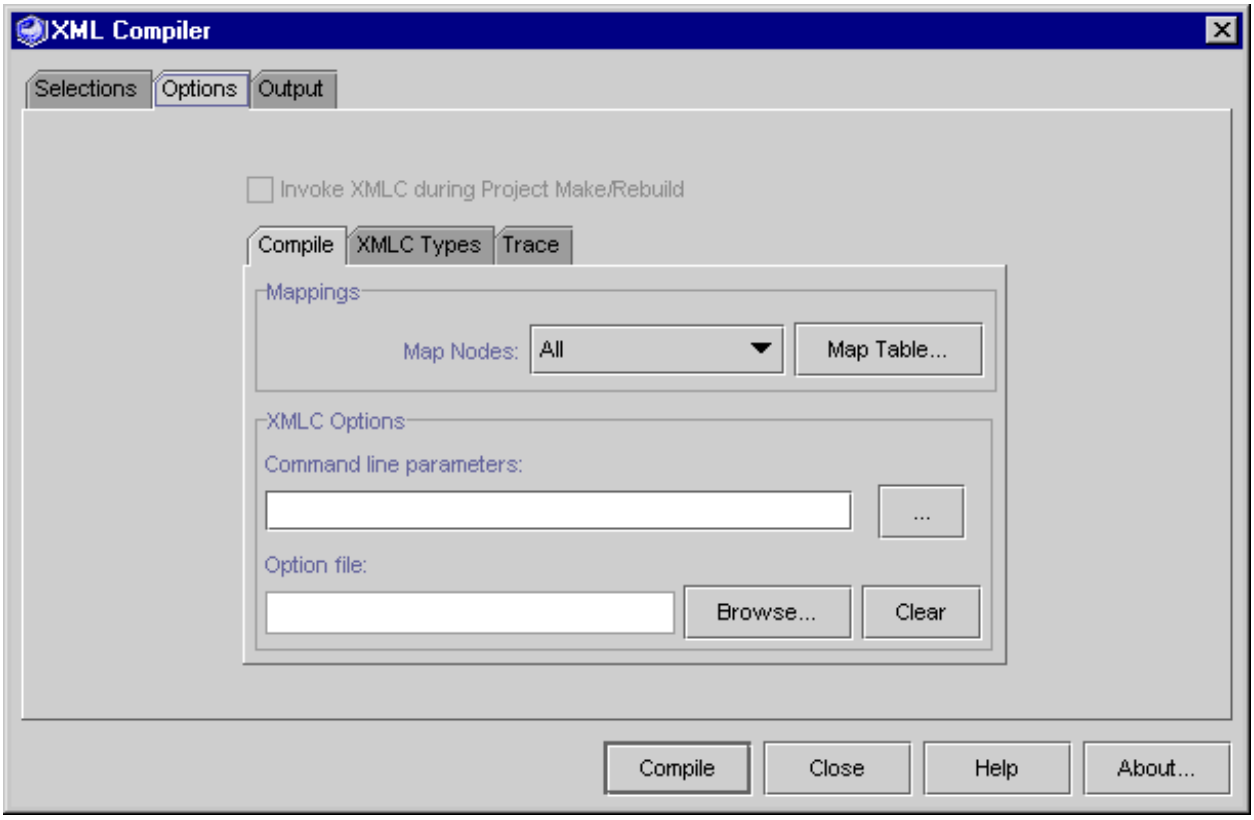

Figure 4.4 Options tab of the Kelp XMLC tool

The Output tab shows the results of running XMLC based on the files you selected on the Selections tab. For information on the Output tab, see "Setting output options" .

Using mapping tables for generated class names

One common option is to use a mapping table to customize generated class names. By default the XML Compiler dialog uses the current directory name to determine the package name in the generated source. Unless your HTML files are stored in the same directory as your presentation package Java source files, you need to use the mappings option to map the resource document directory to the presentation package name.

Note: The Enhydra guidelines recommend that you keep your XMLC documents in a resources directory and map them to the presentation package when you are generating the DOM classes. Both the Kelp sample project and the DiscRack example store HTML files in a resources directory.

To create mappings:

- Click Edit in the Mappings section of the Make tab. This opens the Project Map editor, which lets you associate source directories with package names.
- Click Add to create a new mapping.

  You can enter a source directory or use the Set button to navigate to one.

The dialog box in Figure 4.5 shows how to set up the compiler to use the kelp.webapp.presentation package name when you are compiling HTML pages stored in

`D:/jbuilder5/kelp5/samples/webapp/src/kelp/webapp/resources.`

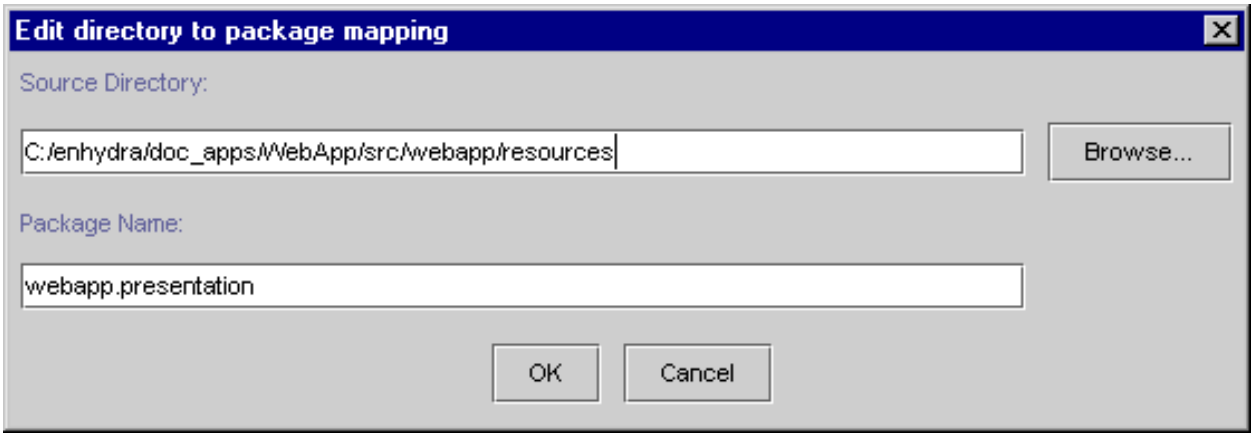

Figure 4.5 Mapping a source directory to a package

Setting output options

The Output tab (Figure 4.6) is automatically selected when you click Compile in the XML Compiler dialog. This tab contains a scrollable text area that displays the results of the compile. If you have any errors in your HTML files, the problems appear on this tab. You can optionally save the output to a text file by selecting Output To Log File and entering a file name.

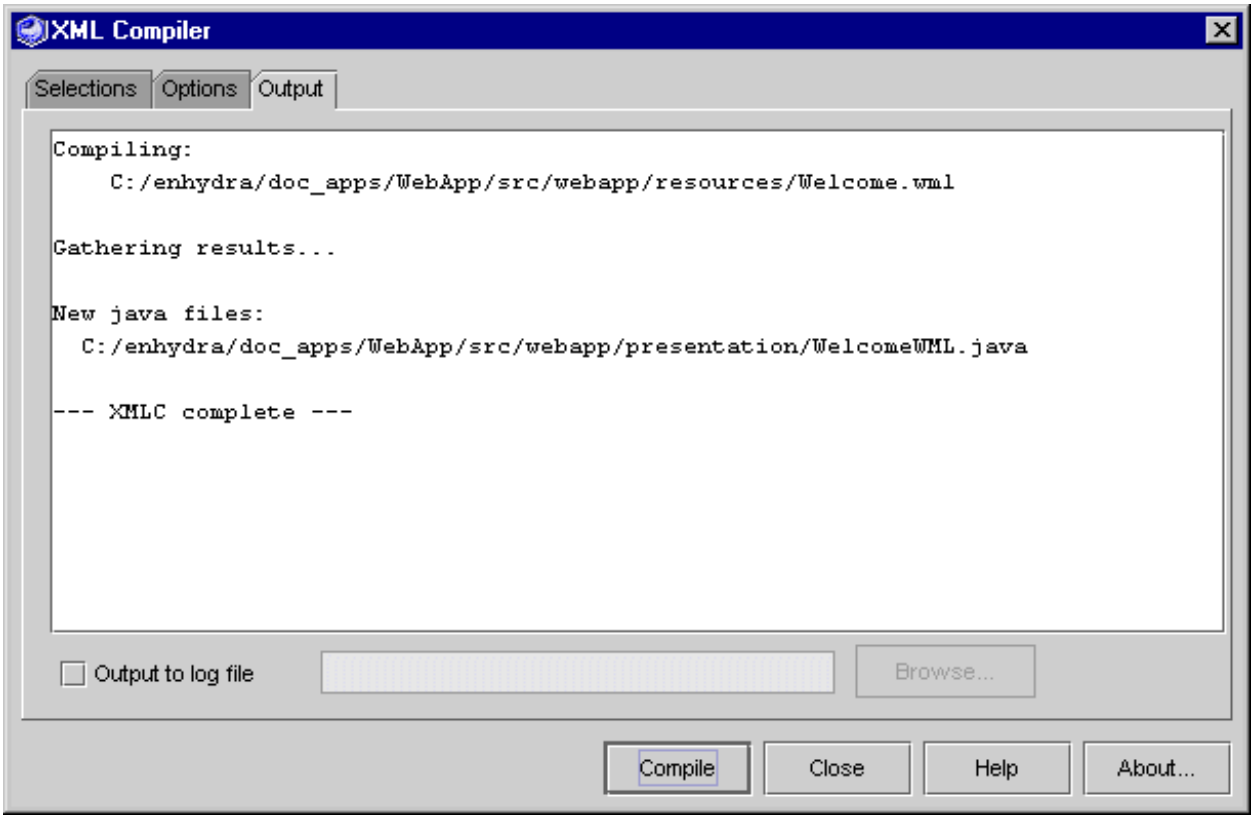

Figure 4.6 Output tab of the XML Compiler dialog

The Output page displays the files that are created during the compilation process. At the start of the compilation process, the dialog erases any files it created during a previous compile. If an HTML file contains a new error, the

associated Java and class files are erased and not regenerated.

Using the Kelp Deployer

The Kelp Deployer lets you quickly configure projects for your current environment and directory structure. The Kelp Deployer helps you perform the following four tasks:

- Generate configuration files and deployment descriptors from input templates
- Copy static content to your document root
- Create a deployable archive, based on the type of application or component
- Set up your project so you can launch Enhydra with your application

General Tab

This tab sets and displays basic information about deployment. The messaging options control how status messages are handled during deployment.

- Deploy Root--Deploy Root is used for setting the archive document root directory. Click the Ellipses (...) button to navigate to a folder.
- Deploy Type--Select the Enhydra component to deploy from this pull down menu.
    - Web Application--If your project uses the Java servlet API, select Web Application. Servlet archives are Web Application Archive (WAR) files that contain the output classes, static content files, and a deployment descriptor.
    - Enhydra Application--If you are using the Enhydra super-servlet API, select Enhydra Application. Enhydra super-servlet archives contain the classes in your output directory.
- Deploy Messages
    - Messages--Shows status messages during deployment.
    - Options

      Display During Project Make/Rebuild--Used in JBuilder when the Kelp Deployer is set to invoke during project building.

      Write to File--Allows you to save all messages to a log file.
- Deploy During Project Make/Rebuild check box--Select Deploy During Build to deploy your application on each project rebuild. The Show Messages box is enabled if Check Deploy During Build is selected. If Show Messages is selected, deployment messages will be shown in the Output tab.
- Overwrite Without Warning--Normally, you want this checked. Otherwise, a warning dialog box is opened for every file that is redeployed.

Input Tab

Template Path points to the location of your project's .in files, which are template files. These files are processed and copied to your Deploy Root folder using relative paths.

- Selections--The Selections tab lets you select which of the available template files in your project will be processed. The left pane shows available templates, and the right pane shows the selected templates. Use the arrow buttons to move files from one pane to the other. Double arrow buttons (<< and >>) move the entire contents of one pane to the other. Single arrow buttons (< and >) move the selected file or files from one pane to the other.

  Select Show Full File Path to show the full path of each template file.

- Replacements--The Replacements tab lets you adjust the template settings for your project. Templates have the extension .in. Templates are text files with placeholders that are replaced with system-specific information by the Kelp Deployer. One of the default placeholders, @CLASSES@, is replaced with the current class output directory specified by your project.

  You can customize the search and replace mechanism by editing the data in the Replace Text table on the Options tab of the dialog. The default option lets you quickly restore the default project settings. The Replace With values can refer to relative paths. If you enter a value starting with a period, the Kelp Deployer substitutes the directory containing the project file for the period. For example, if you open the Kelp Web Application project from:

  ```
  /home/user/jbprojects/samples/webapp/KelpWebApp.jpr
  ```

  and have a Replace Text table containing Text to Find: @CLASSES@ Replace With: ./classes

  The resulting configuration files (.conf) will contain /home/user/jbprojects/samples/webapp/classes in place of the @CLASSES@ placeholders that are in each of the templates (.conf.in).

  For applications deployed on Windows, some Text to Find values of the form @*_PATH@ are handled differently than typical Text to Find values. They are expanded according to the following find values to deal with the different forms of Windows paths:

  - @SHELL_*_PATH@ expands to UNIX-style paths with the system drive as the root of the file system and the //<drive_letter>/<dir> format for other drives. For example, //d/myDir
  - @JAVA_*_PATH@ expands to Java-style Windows paths with the <drive_letter>:/<dir> format. For example, d:/myDir.
  - @OS_*_PATH@ expands to Windows-style paths with the <drive_letter>:\<dir> format. For example, d:\myDir.

  You can change the order for replacements using the Move Up and Move Down buttons. Items at the top are replaced first. The Reset button sets the Replace With text back to the default values.

- Show Full File Path--Displays the full path for files
- Input Templates (.in) Only--If not checked, you can have the Kelp Deployer copy (without searching and replacing strings) all non-template (.in) files from the input root directory to the deploy root directory.
- Enable Input Deployment--Enables or disables input deployment. If you are using auto-deploy, you can normally disable input deployment.

Content Tab

The Kelp deployer will copy content types from the content source directory to the archve document root. Use the archive document root to select content when building Web application archives.

- Paths--If you Select Enable Content Deployment, you can select the directory containing the static content files you want to deploy with your Web application.

    - Content Source--The directory that contains the static content files for your project. These static files will be copied to your document root using relative paths.

    - Archve Document Root--This field cannot be edited. It lists the directory where the archive file will be created.

- Types--The Content Types subtab shows the extensions of static resource files that will be copied from the Content Source directory to your document root directory. Click Add to enter a new extension. Select an extension and click Remove to keep files with that extension from being copied. Click Reset to go to the default values of recognized extensions.

Archive Tab

The Kelp Deployer can create and recreate archive files that you can auto-deploy to a running Enhydra server. The table lists the archives that have been or will be generated during deployment. If the Built check box is selected for an item, then a deployable archive already exists.

Clicking Edit opens the Kelp Archive Wizard. The wizard lets you create an archive. For more information about the Kelp Archive Wizard, refer to the Chapter 3, "Using the Kelp Archive Wizard," of the Developer's Guide.

Run Tab

The deployer can deploy archives to a Enhydra server running locally or set up your project so you can launch it from the IDE.

- Auto-Deploy to Locally Running Server--

- Confgure Project for Starting Server from IDE--

Example 3.1. Deployment example: deploying a Web application

After you have generated and built a Web application, you can deploy the Web application archive (WAR) in a few different ways using the Kelp Deployer.

To run the application manually on a restricted instance of the Enhydra server:

Kelp can generate scripts for running your application locally in an isolated instance of the Enhydra server. This instance of the Enhydra server loads the minimum number of services required to run your application. This configuration of the Enhydra server does not support the use of the LMC or the Web console.

- Select the Run tab in the Kelp Deployer dialog.

- Select Configure Project For Starting Server >From IDE.

- Click the Deploy button.

- In a shell window, change directories to <project_root>/output.
- Execute the start script, run or run.bat, to launch a local instance of the Enhydra server.

  This instance of the Enhydra server loads only the services required to run your application. The application is started automatically.

  Note: You may need to set the script to be executable.
- When Enhydra server is finished booting, access the application with a browser at the following URL:

  `http://localhost:8002`

  You should see the Welcome.html page with a redirect and a dynamic time stamp.

To hot deploy your application:

Hot deployment consists of deploying an application or component to a running instance of Enhydra. The Kelp Deployer can only deploy to a Enhydra server running locally (i.e., running on a mapped drive).

- Execute the Enhydra server start script, multiserver or multiserver.bat in <Enhydra5.1_root>/bin to start the server

  note: It is important to launch the Enhydra server from the /bin directory.
- Select the Run tab in the Kelp Deployer dialog.
- Select Auto-deploy to Locally Running Server.
- Click the Deploy button.

  The archive file is copied to <Enhydra5.1_root>/deploy directory. The Enhydra server will deploy the file automatically. When the file is deployed by Enhydra, the WAR file is removed from the /deploy directory.
- Start the Enhydra Multiserver Admin Console.
- To start your application, log in to the Enhydra server to which you deployed your application.

  The default username and password are "admin" and "enhydra" respectively.
- In the LMC Browser, select the Web application, and click Start in the Operations section displayed in the Workspace.
- Access the application with a browser at the following URL:

  `http://localhost:8002/appname`

  Where appname is the name of your WAR. The name of the WAR is used as the URL prefix by default. You should see the Welcome.html page with a redirect and a dynamic time stamp.

To run your application from within the IDE:

- Select the Run tab in the Kelp Deployer dialog.
- Select Configure Project For Starting Server >From IDE.

- Select Create StartServer.java
- Click the Deploy button.
- Find the generated StartServer.java file in the project source directory.
- Right-click the node and select Compile from the context menu.
- Right-click the StartServer node again and chose Execute from the context menu.

  This starts the Enhydra server from within the IDE.
- When Enhydra server is finished booting, access the application with a browser at the following URL:

  `http://localhost:8002`

  You should see the Welcome.html page with a redirect and a dynamic time stamp.

Using DODS

You can start dods in Dods Generate window with one of the following parameters:

- [doml]

  The doml input file describing data object mapping. Required: Yes
- [outdir]

  Target for generated classes, expressed as a directory path. Required: Yes
- [task]

  Name of Ant task from generate.xml. Required: No.

  This parameter can have one of the following values:

  - without parameters - to create all sql files and java classes and to compile them.
  - dods:sql - to create only sql files.
  - dods:javaNoCompile - to create only java files without compiling them.
  - dods:noCompile - to create SQL files and java files without compiling them.
  - dods:sqlsplit - to create only sql files and separate them in different files using SQLSplitter.
  - dods:noCompileSplit - to create SQL files and separate sql commands using SQLSplitter and java files without compiling them.
- [extensions]

  Template extensions for generating java code. Required: No.

  This parameter can have one of the following values:

- mdb - generate java code with multi database support.
- wdwf - generate java code with WebDocWF support.
- mdbwdwf - generate java code with multi database and WebDocWF support.

Sql and java files will be generated into current project source directory.

Chapter 4. Setting project properties in JBuilder

JBuilder provides many properties that let you customize your projects. After opening a project that you want to use with Enhydra, choose Project|Properties to configure path, compiler, and run settings. You do not need to enter anything in the Servlets tab to work with applications for Enhydra. Most of the settings you will use are found on the Paths tab.

Property pages are dialog boxes in which you set build options for the entire project or for a selected node in JBuilder. A node is an object in the project tree of a JBuilder project. For the purposes of this section, a node refers to a file in a project. Project property pages appear as tabs in the Project Properties dialog box. Node property pages let you customize options for a specific file in your project.

The Kelp project property pages add to JBuilder's existing Code Style, Paths, Run, and Compiler property pages. The Paths property page is where you set the library files your project requires.

- To open the Project Properties dialog box, choose Project|Project Properties.
- To open a Node property page, right-click a file in your project and choose Properties from the right-click menu.

JBuilder opens a property page specific to the file you have selected. If you right-clicked a Java source file, the property page will have RMI (Remote Method Invocation) and JNI (Java Native Interface) settings. To open the Input Template node property page, right-click any file with a .in extension in the template directory, /input. When you right-click an HTML file and select Properties, the XMLC node property page appears.

PATH and CLASSPATH settings

The PATH and CLASSPATH settings are critical for being able to compile and run your applications

Paths page

Configuring your project to use the correct paths for file output, project files, and libraries will simplify the development of applications for Enhydra. This section describes the options on the Project Properties Paths page that are relevant to using Kelp.

Output path

The output directory specifies where you want class files to be created. It is also where the Enhydra server looks for images that have relative references to the presentation objects. For example, the Kelp Application Wizard creates images under:

```
<source_directory>/<package_directory>/presentation/media
```

When you build the project inside JBuilder, the image is copied to the output directory along with the compiled class files. For the image to be displayed properly, the output directory must be set to a classes subdirectory under the source directory.

The Kelp Deployer also uses this setting when it creates deployable archives.

Source subtab

The source path is where the compiler looks for Java files and packages to compile. When you compile package names, the source files will be in subdirectories under the source directory. For example, if your highest-level package is com, the com directory will be located directly under a source path.

You can set the Source path to the same location as the project file. When you create a project using the Project wizard in JBuilder Foundation, it creates a directory for your project under jbprojects in your home directory. You can use this directory as your source.

JBuilder lets you select multiple source paths. This can be useful when you are working on several modules from source code. For example, if you have checked out the Enhydra source code from CVS, you'll see that it is divided into several modules. Each module contains its own source directory.

Note: If you are using automatic source packages in JBuilder 5, make sure that your output path is not a subdirectory of any of your source paths.

Note: Remove any source paths that your project does not require.

Required libraries subtab

The application CLASSPATH must contain the following JARs in order to build and run Enhydra applications:

- `<Enhydra5.1_root>/lib/enhydra.jar`
- `<Enhydra5.1_root>/lib/xmlc.jar`
- `<Enhydra5.1_root>/lib/xerces.jar`
- `<Enhydra5.1_root>/lib/tomcat.jar`
- `<Enhydra5.1_root>/lib/core.jar`
- `<Enhydra5.1_root>/lib/xhtml.jar`
- `<Enhydra5.1_root>/lib/wireless.jar`
- `<Enhydra5.1_root>/lib/build/toolbox.jar`

You can add JARs to an application CLASSPATH by using JBuilder libraries. If you installed Kelp using the Windows installer, you already have an Enhydra library defined for you. If you are using JDBC in your application, you will need to add the JDBC driver as a library.

If you need to define an Enhydra library in JBuilder Foundation, click Add and then click New. Name your library and click Add to navigate to the JAR file you want to add.

Note: You may need to modify the Enhydra library if you have upgraded to a newer version of Enhydra. Make sure the JARs specified for the library match the JARs referenced in the IDE CLASSPATH. Use the About button in the XML Compiler dialog to determine the version of Enhydra that is in the IDE CLASSPATH.

Build properties

This section describes the options on the Project Properties Build page that are relevant to using Kelp.

Generate source to output path option

On the Build page, you can select Generate Source To Output Path to keep the generated XMLC Java files separate

from your HttpPresentation implementations. If you select this option, the Java source files for the DOM classes will be created in a Generated Source directory under your output classes directory.

Run options

This section describes the options on the Project Properties Run page that are relevant to using Kelp.

Main class option

JBuilder Foundation lets you specify which class to run when you run the project. This class does not need to be part of your project's source files.

If you have an Enhydra library selected under the Required Libraries option, you can use the main class to launch the Enhydra server. Use the Set button to select the Enhydra server startup class, com.lutris.multiServer.Multiserver.

Application parameters option

If you have set the main class to launch the Enhydra server, you can use the application parameters to pass in a configuration file. For example, if you create a Web application using the Kelp Application Wizard, the application parameter is set to:

```
<source_directory>/output/conf/servlet/servlet.conf
```

Chapter 5. Setting project properties in NetBeans

After creating a NetBeans project to manage your source, you can edit Kelp project properties by selecting Project|Settings from the main menu. The project Settings window contains two Kelp-specific nodes: XML Compiler and Kelp Deployment. By selecting either node, one can set "simple" properties via the property sheet. For more robust settings, however, one should use the customizer, accessible via the node's Customize context menu option.

Node properties may be set on recognized markup language files (e.g., HTML, WML) and template files by right-clicking on the node and selecting the Customize option. The selected property can be set more quickly by using the node's property sheet, which is viewed by selecting the Properties context menu option.

Classpath

The classpath is set by mounting directories and JARs to the project's Filesystems collection. By default, some of the Enhydra JARs are mounted when NetBeans starts, and do not appear in the Filesystems tab. These JARs include:

- `<Enhydra5.1_root>/lib/enhydra.jar`
- `<Enhydra5.1_root>/lib/xmlc.jar`
- `<Enhydra5.1_root>/lib/xerces.jar`
- `<Enhydra5.1_root>/lib/tomcat.jar`
- `<Enhydra5.1_root>/lib/core.jar`
- `<Enhydra5.1_root>/lib/gnu-regexp-1.1.4.jar`
- `<Enhydra5.1_root>/lib/xhtml.jar`
- `<Enhydra5.1_root>/lib/wireless.jar`
- `<Enhydra5.1_root>/lib/build/toolbox.jar`

When using the Kelp Application Wizard, additional JARs may be added to the mounted Filesystems if they are required to build the default configuration.

Project directory

The project directory is where Kelp will look for the default input, output and source directories, or their equivalents. As NetBeans does not create a tangible project directory the way JBuilder does, the user will have to provide this directory. By default, the Project directory is assumed to be the first directory added to the project (not to be confused with the first mounted directory). This default can be overridden via the XMLC "Project Directory" property.

Source directory

Kelp uses the source directory as the parent for all generated packages. The source directory defaults to <project_dir>/src, but this can be overridden by adjusting the "Source Directory" XMLC property.

Output directory

Kelp for NetBeans does not have an output directory, per se. The output directory is the same as the source directory, and this is where the class files should be generated on compilation.

Deploy root

The deployment root directory can be set through the Kelp deployer property sheet.

Run options

When deploying your app, if you have chosen to generate a StartServer.java file, you can run your application from the IDE. In this case, a pre-configured external executor should be attached to the StartServer file, so you can right-click the node and select Execute from the context menu.

Setting up a project for use with the Kelp ApplicationWizard

If the Kelp Application Wizard is used to jumpstart your development effort, we recommend the following project setup:

- Create the project directory.
- In NetBeans, select Project|New Project from the main menu.
- In the Create New Project dialog, enter a name for your project.

  Kelp does not use this project name internally, so there are no naming restrictions.
- When asked if you would like to start with a new Filesystem configuration, choose the New option.
- When prompted to mount a directory, select your project directory.
- Once the project is created, select the Project tab in the Explorer window.
- Right-click on the project root and select Add Existing from the context menu.
- Select your project directory and click the "OK" button.

  You are now ready to generate a new application. See "Using the Kelp Application Wizard" for more details.

Chapter 6. Setting XMLC properties

XMLC properties can be set for the project, a folder, or individual files. XMLC properties set for a file always override properties set for a folder, and folder properties always override properties set for the project.

XMLC project properties

You can view the XMLC project properties page by opening the Project|Project Properties dialog box and selecting the Enhydra XMLC tab (Figure 4.7). This page shows the same make and output options that you can set on the Options tab of the XML Compiler dialog.

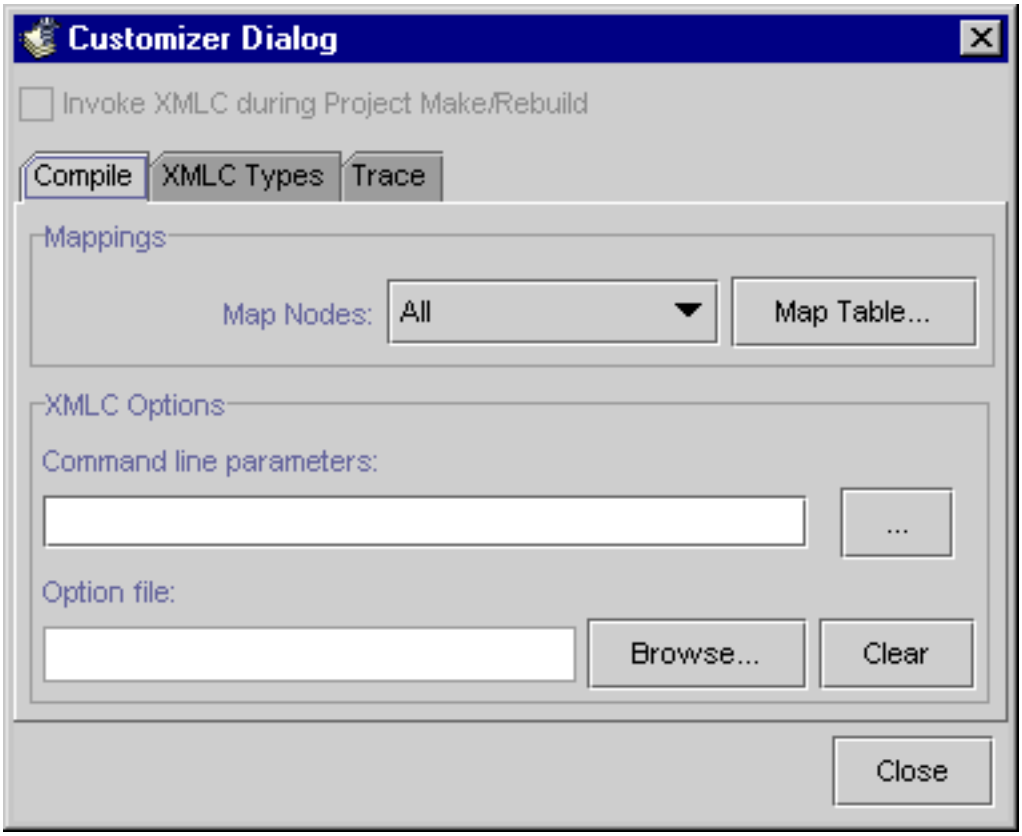

Figure 4.7 Project Properties page

The Compile tab includes three sections:

- Generated Java Source: Tells you which XMLC options need to be set for the current IDE.
- Mappings: Determines how class names are generated.
- XMLC Options: Lets you specify XMLC command-line options or select an XMLC Options file to set XMLC options. For more information on XMLC command-line options, see Chapter 5, "Enhydra XMLC,"of the Developer's Guide.

Click Edit to open the XMLC Parameter Editor. The XMLC Parameter Editor lets you organize complex XMLC command-line options.

In the Mappings section, you can define a mapping table and set the nodes that you want to use with the table. The

mapping table lets you specify package names to use when generating DOM classes for a given directory. For example, the sample Kelp Web application project uses a mapping table to map all the HTML files in a resources directory into the kelp.webapp.presentation package.

To view or edit the package name mapping:

- Open the XML Compiler dialog
- Select the Options tab.
- Click the Edit button.

This opens the mapping table, where you can add a new entry that maps a directory to the correct presentation package. Here is a sample mapping for the Kelp sample project:

Source directory:

```
/user/local/jbuilder/kelp2/samples/webapp/src/kelp/webapp/resources
```

Package name:

```
kelp.webapp.presentation
```

Select the Output tab on the XMLC page in Project|Properties. These settings cause the XMLC to stream out additional information when compiling the HTML files. The Output settings do not affect the generated DOM classes.

Note: For faster compiles, deselect all the Output options.

XMLC node property page

You can open the XMLC node property page by right-clicking an HTML or WML file in the Project pane and selecting Properties. This opens the Properties page for the selected file. As Figure 4.8 shows, this page has four options:

- Copy to Output as a Static Resource: Select if the file will be served as a static file. The file will be copied to the output directory.
- Generate DOM Class: Select if you want to select this file for compilation with XMLC.
- Generated Class Names: Lets you configure the names of generated classes. Click Map Table to open the Map editor.
- XMLC Options: Lets you enter XMLC command-line options for the selected file, or for your entire project. This option is available only when Generate DOM Class is selected.

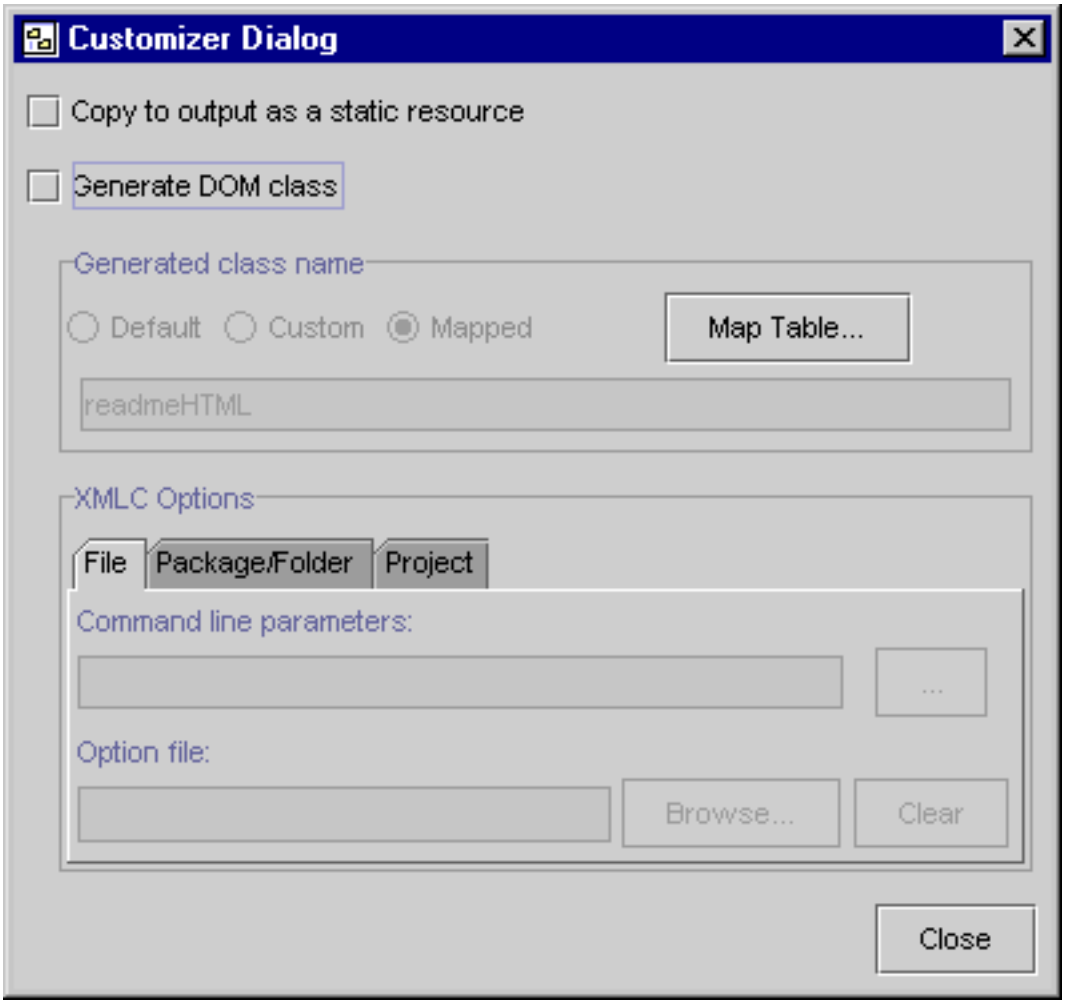

Figure 4.8 XMLC Node Properties page

How Kelp sets class names for XMLC

You can use the Generated Class Name section to select how you want to generate the class name. There are three choices: default, custom, and mapped. The file name, directory location, and project source path determine the default name.

For example, if you create a new application in /home/bob/jbprojects/untitled1, the source path for the Welcome.html file is /home/bob/jbprojects/untitled1, the directory location is /home/bob/jbprojects/untitled1/untitled1/presentation and the file name is Welcome.html. The package name is untitled1.presentation.WelcomeHTML.

The source path is removed from the directory location to create the new package name. The new class name is formed by removing the .html extension and adding "HTML". Select the Custom option if you want to enter your own class name. When using custom class naming, make sure the name you enter conforms to a valid Java identifier.

Note: If the selected HTML file is not located in a subdirectory of one of the Project source path settings, Kelp may not be able to generate a valid default class name.

To map your HTML files to user-defined class names, select Mapped and click Map Table to open the Edit Project Map dialog box. XMLC uses this mapping table to set custom package names based on the directory of the HTML

file. This mapping table is normally used to map a resources directory to a presentation package. There is only one mapping table per project. For more information, see "XMLC project properties" and "Using the Kelp XMLC tool.".

Setting XMLC options for selected files

To set additional XMLC options for the selected file, use the XMLC Options section in the XMLC node property page (see Figure 4.8).

Command-line parameters let you add command-line options to XMLC from within your IDE. Click Edit to open the XMLC Parameter Editor. The Parameter Editor allows you to organize complex command-line parameters. If you have an options file for XMLC, click Set and select the options file. To clear a previously selected options file, click Clear. For more information on using XMLC, see Chapter 5, "Enhydra XMLC," of the Developer's Guide.

Input Template node property page

The Input Template node property page contains the same settings that appear on the Template tab of the Kelp Deployer project properties page. Right-click the .in file and choose Properties to open the Input Template node property page for a .in file in your project.

In addition to the project properties, there is a Generate .conf check box you can select to generate configuration files for the current template.

Chapter 7. Kelp sample projects

Kelp includes two sample projects that demonstrate how to use XMLC and Enhydra to create Web and wireless applications within JBuilder. If you are new to both Enhydra and Kelp, you can start learning Enhydra by running the Kelp sample application. The Kelp sample demonstrates only a small part of Enhydra's capabilities.

Once you feel comfortable with the Web application sample, examine the DiscRack example that comes with Enhydra, located in <Enhydra5.1_root>/examples/DiscRack. The DiscRack example provides a more complete application that incorporates JDBC access. You can run the DiscRack example using most JDBC-compliant data sources, including Oracle, Microsoft SQL Server, PostgreSQL, and InstantDB.

If you are already familiar with Enhydra, you can use DiscRack to see how to set up a project for your own applications. For more information on DiscRack, see Chapter 8, "DiscRack sample application," of Getting Started with Enhydra.

The sample servlets

The sample project consists of four presentation objects that show some of the common uses of XMLC. The sample is simplified in that it only contains a presentation layer. Production Web applications normally contain at least three packages, including presentation, business, and data. For a detailed explanation on how you can separate applications into these three functional areas, see Getting Started with Enhydra.

Note: If you have moved a sample project after running the Kelp Deployer, run the Kelp Deployer again before using the sample.

Each servlet dynamically generates HTML files using XMLC. The HTML source files are located in a resources directory. Each HTML file is compiled into a class file using XMLC. There is a corresponding Java file that implements HttpServlet for each HTML file. These files work with the generated HTML classes to create and process input from the Web pages. The Java files are located in the kelp.webapp.presentation package. The four servlets demonstrate the following:

- Greetings Servlet: This is similar to a traditional Hello World example. This servlet contains one HTML element that is set through XMLC to greeting the user with a phrase contained in the Java source.
- Table Servlet: One of the most common tasks in dynamic HTML generation is populating an HTML table. This servlet shows you how to define a table as a template in HTML and then populate it through Java when a user requests the page. In a real-world application, the data would most likely come from a JDBC data source. For the sake of simplicity, this example populates the table with an array of values that are hard coded into a Java file.
- New Node Servlet: XMLC allows you to insert HTML blocks from external sources into an existing page. This example shows a page containing a span of HTML that is read in from a text file. You can modify the text file to alter the page without recompiling the HTML or Java files.
- Form Servlet: You can use XMLC with HTML input fields to create data entry forms. This servlet shows you how to update a file on the server with input values retrieved from a Web browser. For simplicity, this example uses a property file to store displayed values. Normally, values are stored in a JDBC data source that is accessed through the business and data layers of an application.

Chapter 8. Debugging Kelp projects

One of the advantages of using an IDE for developing applications and components for Enhydra, is that the IDEs provide tools for debugging.

Debugging with NetBeans

NetBeans supports remote debugging using the Java Platform Debugger Architecture (JPDA). The JPDA enables you to set breakpoints on classes, threads, and variables; to set conditional breakpoints; and to examine the value of an expression. The JPDA is the default debugger if you are running the Java 2 Platform, Standard Edition (J2SE), SDK 1.3.

The Debugging workspace automatically loads when you start a debugging session. By default, this workspace includes three windows:

- The Debugger window with separate tabs for managing breakpoints, variables, watches, and threads
- The Output window for displaying messages from the debugger
- The Source Editor for showing the line in the source code where the program is stopped

To debug using SharedMemoryAttach:

The SharedMemoryAttach method may be faster

- To debug an application using SharedMemoryAttach, launch an instance of Enhydra locally with the following command:

  ```
  multiserver -debug -jdwp-args "transport=dt_shmem,server=y,suspend=n,
  address=localdebug"
  ```

- In NetBeans, attach to the VM by choosing Debug|Attach to VM from the menus.
- In the Attach to VM dialog, set the following options to start debugging:
  - Debugger Type: Default debugger (JPDA)
  - Connector: SharedMemoryAttach
  - Transport: dt_shmem
  - Name: localdebug (this is specified in the command argument used to launch the Enhydra server)
- Deploy and start the application to debug.
- Open the application code in NetBeans to add watches or breakpoints.
- Access the application in a browser to exercise the code you are debugging.

To debug using SocketAttach:

By default, when you start the Enhydra server with the -debug argument, the server is configured for debugging via a socket attachment. This may be slower.

- To debug an application using SharedMemoryAttach, launch an instance of Enhydra locally with the following command:

  ```
  multiserver -debug
  ```

- In NetBeans, attach to the VM by choosing Debug|Attach to VM from the menus.
- In the Attach to VM dialog, set the following options to start debugging:

  - Debugger Type: Default debugger (JPDA)
  - Connector: SocketAttach
  - Transport: dt_socket
  - Host: Enter the name of the host upon which the Enhydra server was started (e.g., localhost)
  - Port: Enter the port assigned for debugging

- Deploy and start the application to debug.
- Open the application code in NetBeans to add watches or breakpoints.
- Access the application in a browser to exercise the code you are debugging.

Debugging with JBuilder

The only requirement for debugging applications from JBuilder is that you must start the Enhydra server from your JBuilder project. Before you debug an application, make sure you can run the Kelp sample project successfully. To do this, you will need to run the XML Compiler and the Kelp Deployer.

To debug an Enhydra application:

- Make sure the Enhydra server is not already running.
- Open the Kelp sample project.
- Select TableServlet.java in the kelp.webapp.presentation package.
- Locate the for loop of the clearTable() method and click the left edge of the editor to put a breakpoint on the line that reads table.appendChild(newRow);.

  After you set the breakpoint, the line appears in red.

- From the menu, choose Run|Debug.

  This starts the debugger. The debugger opens a command window even if you have selected Execution Log for console I/O.

- Open the table page in your browser by opening http://localhost:9000, where 9000 is the specified port and clicking Table Page.

  This triggers the breakpoint, as shown in Figure 4.9.

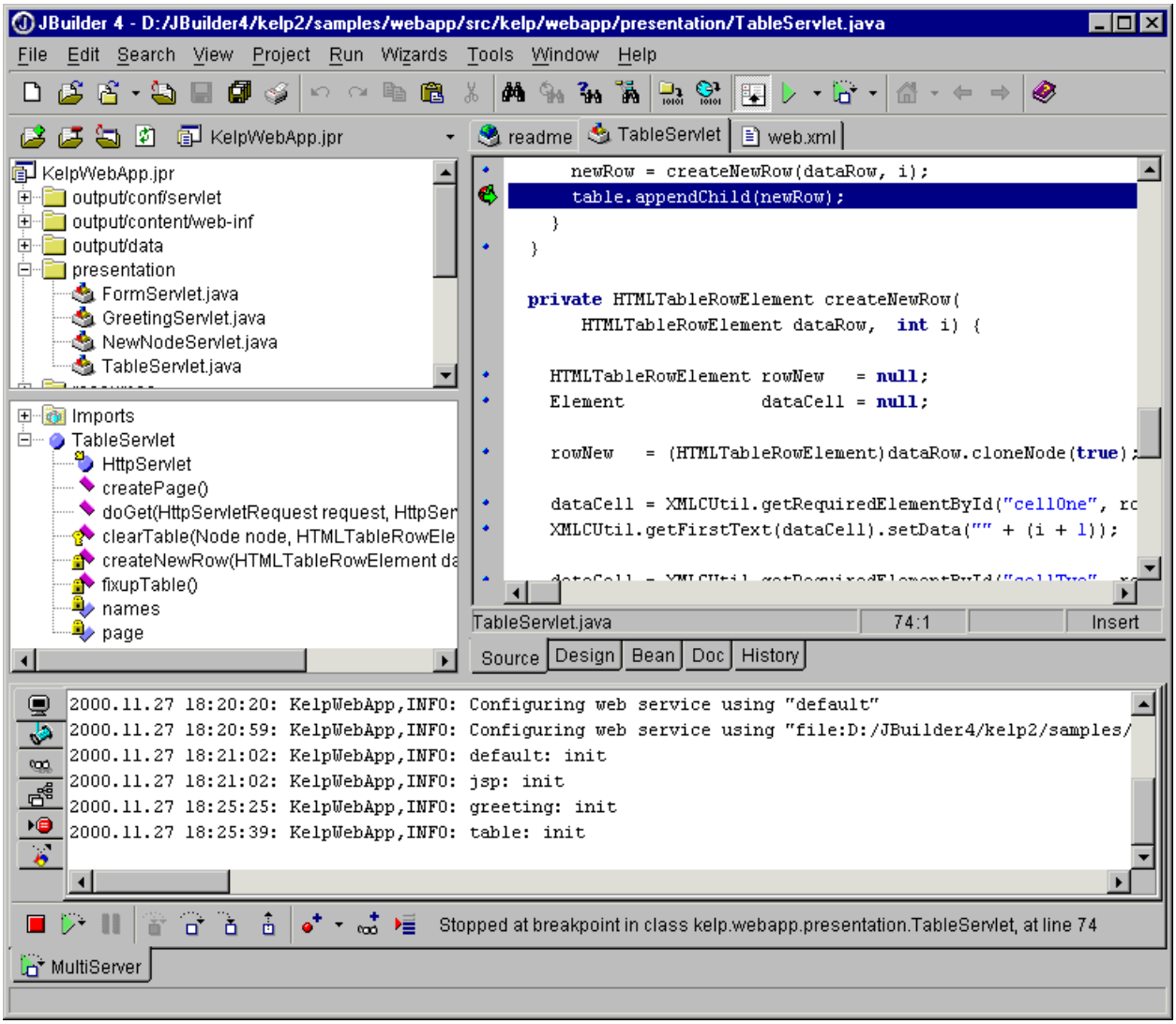

Figure 4.9 Debugging an application in JBuilder

You can set the value of i to 4 to change the number of rows generated in the table. Your browser might time-out if you keep the program suspended for too long.

Chapter 9. Kelp for JBuilder differences

The primary functions of Kelp are the same regardless of the IDE used. However, there are some differences in how Kelp items are accessed from within JBuilder.

Using the Enhydra Import Wizard

The Enhydra Import wizard lets you quickly import source files from an existing Enhydra GNU Makefile project into a NetBeans or JBuilder project, capturing any XMLC options specified in the Makefile system.

As an example, this section describes how to import the DiscRack sample project to JBuilder.

Note: Applications generated by the Kelp Application Wizard from within the IDE do not use the Makefile system or Ant. Applications generated using the Kelp Application Wizard launched from the command line include a build.xml and are built with Ant.

The following steps explain how to import and run the DiscRack example using Kelp with JBuilder. The steps are divided into three sections:

- Importing DiscRack
- Building DiscRack
- Running DiscRack

If your application is running outside of JBuilder, you should be able to skip "Running DiscRack" .

Importing DiscRack

To import DiscRack into JBuilder, follow these steps:

- Build DiscRack as described in the README file. In addition to running XMLC and compiling the java source code, this generates the data package source code.
- Create a new JBuilder project file.

    - In JBuilder, select File|New Project.

      This opens the Project Wizard dialog box.

    - In the Project Wizard dialog box, set Project Name to DiscRack.

      This name has no impact on the project or application, but we recommend that you use DiscRack for consistency.

    - Set the Root Path to <Enhydra5.1_root>/examples, where <Enhydra5.1_root> is the directory where Enhydra is installed.

      For example, the Root Path might be C:/Enhydra5.1/examples. You can click the button to the right of the field to navigate to the directory.

    - In the Project Directory Name field, enter DiscRack.
    - Accept the defaults for the rest of the fields.

    - Click Finish to generate the new project.
- Choose Wizards|Enhydra Import.
- Set the project directory to <Enhydra5.1_root>/examples/DiscRack.
- Click Next to navigate through the wizard, accepting all default settings.
- Click Finish to import the project.

Building DiscRack

To build DiscRack, follow these steps:

- Open the XML Compiler.
- Click Compile to generate the DOM Java source files using XMLC.
- When XMLC is finished compiling, click Close.
- Choose Project|Make to compile the Java source files.
- Choose Tools |Kelp Deployer.
- Click Deploy to copy static content files, process configuration templates, and create a deployable archive.

Running DiscRack

To run DiscRack, follow these steps:

- Choose Project|Project Properties.
- Add the InstantDB library to the required library list:

    - Select the Paths tab.
    - Select the Required Libraries subtab.
    - Click Add.
    - Click New.
    - Name the library "InstantDB".
    - Click Add.
    - Navigate to the location of idb.jar, select it, and click OK.
    - Click OK until the Project Properties dialog box is closed.
- Choose Run|Run Project.
- Open a browser to http://localhost:5555 where 5555 is the specified port, to view the application.