Lutris®

Enhydra™

# Developer's Guide

## Acknowledgements

**Lutris Enhydra Development:** Jason Abbott, Kyle Clark, Mark Diekhans, Larry Deran, Michael Gardner, Dick Gemoets, Scott Harrison, Craig Heath, Peter Hearty, Aidan Hosler, Wes Isberg, Andy John, Peter Johnson, Matthew Kalastro, Ray Kiuchi, Paul Mahar, John Marco, Shawn McMurdo, Ioan Mitrea, Paul Morgan, Christophe Ney, Robert Pirani, Scott Pirie, Joseph Shoop, Wayne Stidolph, Josh Sugnet, Simon Tuffs, Mike Ward.

**Lutris Customer Services:** Andy Ames, Debbie Brackeen, Peter Darrah, Jim Dumont, Jason Dunton, Anne Hopkins, Andrew Longsworth, Lindsey Lonne, Livia Peras, John Powell, Christopher Reed, Jane Richter, Katrina Seitz, Steve Slany, Daniel Thomas.

**Lutris Consulting:** Ashley Baumann, David Black, Dennis Chatham, Jon Coyle, Tola Dalton, Jay Gunter, John Hellier, Donna Karolchik, Bill Karwin, Alyssa Lalanne, Graham Moore, Jim Murphy, Thom Nelson, Natasha Perry, Kristen Pol, Lisa Reese, Matt Schwartz, Harvey Thompson, Robert Trama, Shiming Shi, David Simons, Jonathan Webb.

**Lutris Marketing:** Keith Bigelow, Scott Campbell, Lynda Hall, Holly Hamner, Klaus Krull, Helen Meservey, Lynn Renshaw, Greg Schwarzer, Gillian Webster, David Young.

**Lutris Enhydra Documentation:** Teresa Andrews, Ian Evans, Curtis Gavin, Laurel Kline, Michael Maceri, C. Rand McKinney.

**Thanks also to the following Lutris departments:** customer service, consulting, finance, legal, IS, marketing, quality assurance, research and development, sales, technical support, training, and the executive staff.

Printed in the U.S.A.
ENW-US041-35 1E1R1200
0102030405-9 8 7 6 5 4 3 2 1

1200 PACIFIC AVENUE, SUITE 300, SANTA CRUZ, CA 95060 ✦ PHONE 831.471.9753 ✦ FAX 831.471.9754 ✦ http://www.lutris.com

# Contents

# Chapter 1

# Introduction

This book describes the tools included with Lutris® Enhydra,™ and discusses some techniques for developing and deploying Enhydra applications. It provides detailed documentation on the following main components of Enhydra:

- Application Wizard
- Multiserver Administration Console
- Kelp tools
- Enhydra™ XMLC
- Data Object Design Studio (DODS)
- InstantDB
- Enhydra Director

The appendices include information on:

- Integrating SSL with Enhydra
- XMLC metadata files
- The data object markup language (DOML) used by DODS

## What you should already know

This book assumes you have the following basic skills:

- General understanding of the Internet, the World Wide Web (Web), and Hypertext Markup Language (HTML).
- Good working knowledge of the Java programming language. Some knowledge of Java servlets is also helpful.
- Knowledge of basic UNIX commands and the UNIX `make` utility. This is not necessary if you are developing your application with the Kelp toolset in an IDE such as JBuilder.
- Good understanding of relational databases; knowledge of SQL is helpful.

# Conventions used in this book

The typographical conventions used in this book are listed in Table 1.1.

**Table 1.1** Typographical conventions

| Convention | Description |
|---|---|
| *Italics* | Indicates variables, new terms and concepts, and book titles. For example,<br>• A *servlet* is a Java class that dynamically extends the functionality of a Web server. |
| `Fixed-width` | Used to indicate several types of items. These include:<br>• Commands that you enter directly, code examples, utility programs, and options. For example,<br>◦ `cd mydir`<br>◦ `System.out.println("Hello World");`<br>◦ `make` utility<br>◦ `-keep` option<br>• Java packages, classes, methods, objects, and other identifiers. For example,<br>◦ `ErrorHandler` class<br>◦ `run()` method<br>◦ `Session` object<br>**Note:** Method names are suffixed with empty parentheses, even if the method takes no parameters.<br>**Note:** Only specific references to object names are in fixed-width; generic references to objects are shown in plain text.<br>• File and directory names. For example:<br>◦ `/usr/local/bin`<br>**Note:** UNIX path names are used throughout and are indicated with a forward slash (/). If you are using the Windows platform, substitute backslashes (\) for the forward slashes (/). |
| *`Fixed-width italic`*<br>and<br>*`<Fixed-width italic>`* | Indicates variables in commands and code. For example,<br>• `xmlc [`*`options`*`\|`*`optfile`*`.xmlc ...] `*`docfile`*<br>**Note:** Angle brackets (< >) are used to indicate variables in directory paths and command options. For example,<br>• `-class <`*`class`*`>` |
| **Boldface** | Used for the words **Note**, **Tip**, **Important**, and **Warning** when they are used as headings that draw your eye to essential or useful information. |
| *Keycaps* | Used to indicate keys on the keyboard that you press to implement an action. If you must press two or more keys simultaneously, keycaps are joined with a hyphen. For example,<br>• *Ctrl-C*. |

**Table 1.1** Typographical conventions (continued)

| Convention | Description |
|---|---|
| \| (pipe) | Used as a separator in menu commands that you select in a graphical user interface (GUI), and to separate choices in a syntax line. For example,<br>• File\|New<br>• `{a\|b\|c}`<br>• `[a\|b\|c]` |
| { } (braces) | Indicates a set of required choices in a syntax line. For example,<br>• `{a\|b\|c}`<br>means you must choose `a`, `b`, or `c`. |
| [ ] (brackets) | Indicates optional items in a syntax line. For example,<br>• `[a\|b\|c]`<br>means you can choose `a`, `b`, `c`, or nothing. |
| . . . (horizontal ellipses) | Used to indicate that portions of a code example have been omitted to simplify the discussion, and to indicate that an argument can be repeated several times in a command line. For example,<br>• `xmlc [`*`options`*`\|`*`optfile`*`.xmlc ...] `*`docfile`* |
| plain text | Used for URLs and generic references to objects. For example,<br>• http://www.lutris.com/documentation/index.html<br>• The presentation object is in the presentation layer. |
| ALL CAPS | Indicates SQL statements. For example:<br>• CREATE statement |

Table 1.2 lists additional conventions used in this book, including the convention used to describe the Enhydra root directory, platform-related conventions, and so on.

**Table 1.2** Additional conventions

| Convention | Description |
|---|---|
| Enhydra root directory | When you install Enhydra, you install the Enhydra executables and libraries in a directory of your choosing. This directory is referred to as the Enhydra root directory or `<`*`enhydra_root`*`>`. |
| Paths | UNIX path names are used throughout and are indicated with a forward slash (/). If you are using the Windows platform, substitute backslashes (\) for the forward slashes (/). For example,<br>• `/usr/local/bin` |
| URLs | URLs are indicated in plain text and are generally fully qualified. For example,<br>• http://www.lutris.com/documentation/index.html |
| Screen shots | Most screen shots reflect the Microsoft Windows look and feel. |

# Lutris Enhydra document set

The Lutris Enhydra documentation set is an excellent resource for information about Enhydra. The documentation set includes the following printed guides.

**Note** Online versions of these books in both PDF and HTML formats are provided with the purchase of Lutris Enhydra. These online books, along with additional Enhydra online documentation, are located in the `doc` subdirectory of the directory in which you installed Lutris Enhydra. You can also view the online books and installation instructions directly from the product CD.

## Getting Started

*Getting Started with Lutris Enhydra* introduces the fundamentals of Enhydra. The purpose of this book is to introduce Lutris Enhydra and provide a groundwork for understanding and working with Enhydra and its associated tools. It includes a detailed tutorial and an explanation of the Enhydra DiscRack sample application.

**Note** As part of our commitment to support the Enhydra and open-source communities, Lutris Technologies has made the latest online version of *Getting Started with Lutris Enhydra* available for free viewing and download from the Lutris Documentation home page at http://www.lutris.com/documentation/index.html.

## Developer's Guide

The Lutris Enhydra *Developer's Guide* introduces advanced topics and explores key features of Enhydra in detail. The purpose of the *Developer's Guide* is to provide developers with the information they need to create and debug sophisticated Enhydra applications. This guide provides in-depth information on the Lutris Enhydra development tools:

- Application Wizard
- Multiserver Administration Console
- Kelp tools
- Enhydra™ XMLC
- Data Object Design Studio (DODS)
- InstantDB
- Enhydra Director

**Note** The *Developer's Guide* is available only with the purchase of Lutris Enhydra.

## Wireless Application Developer's Guide

The Lutris Enhydra *Wireless Application Developer's Guide* presents information on wireless technologies and describes how to develop wireless applications with Enhydra. It includes a detailed tutorial and an explanation of the Enhydra AirSent wireless sample application.

**Note** The *Wireless Application Developer's Guide* is available only with the purchase of Lutris Enhydra.

# Lutris documentation updates available online

The latest product documentation updates and release notes are available to registered users from the Lutris Documentation home page at http://www.lutris.com/documentation/index.html.

# Contacting Lutris Technical Publications

We strongly encourage you to send us your feedback because it helps us understand your needs and makes our documentation even better. You can submit feedback from the Lutris website at http://www.lutris.com/documentation/feedback/index.html. You can also submit feedback by sending email to documentation@lutris.com.

# Where to find support and training for Lutris Enhydra

Lutris Enhydra includes a package of products for developing Enhydra applications, including open-source products. Lutris Technologies, Inc. provides support and services for Lutris Enhydra.

**Note** Open-source communities or commercial entities support the other products. For detailed information on the available support options for those products, please refer to the appropriate group or company website.

## Lutris support

Lutris offers a variety of support programs designed to assist you with your technical support needs. We can help with installing and using your Lutris product, developing and debugging your code, maintaining your deployed applications, providing onsite consulting services, and delivering enterprise-level support. For more information about any of the Lutris technical support programs, see the Lutris Support home page at http://www.lutris.com/support/index.html or call Lutris Customer Service toll-free at 1-877-688-3724, Monday–Friday, 8 a.m.–6 p.m. Pacific Time (outside of North America, please call 1-831-460-7590).

### Registering your product online

Lutris strongly encourages you to register your product online. Registering your product entitles you to 15 days of free installation support and provides you with the option of purchasing Lutris Support Services.

To register online, browse to the product registration form that is available at http://www.lutris.com/register.html.

### Contacting Lutris Technical Support

For more information about any of Lutris' technical support programs, see the Lutris Support home page at http://www.lutris.com/support/index.html or call Lutris Customer Service toll-free at 1-877-688-3724, Monday–Friday, 8 a.m.–6 p.m. Pacific Time (outside of North America, please call 1-831-460-7590). You can also send email to support@lutris.com.

### Submitting bug reports to Lutris Technical Support

To report suspected Lutris Enhydra bugs, fill out the Bug Report form available at http://lutrisbugs.custhelp.com/cgi-bin/lutrisbugs/people. We recommend that you choose the Search Bugs link before submitting a bug report so that you can see if your bug has already been reported. Be sure to include steps-to-reproduce, exact error messages, and code snippets, if applicable, to help us better evaluate your report.

## Lutris training

Lutris wants to ensure your success. Our expert trainers guide participants through hands-on labs designed to provide an intensive learning environment where participants quickly learn how to maximize Lutris Enhydra in development and deployment environments.

### Available training courses

The following courses are currently offered by Lutris Technologies. For more information about training offerings, see the Lutris Training home page at http://www.lutris.com/training/index.html.

#### Lutris Enhydra Fundamentals

Lutris Technologies currently offers a five-day, instructor-led course titled *Lutris Enhydra Fundamentals*. This course is intended primarily for developers, architects, project managers, IT staff, and consultants who will be using Lutris Enhydra or are evaluating it for future projects.

#### Building Wireless Applications

Lutris Technologies currently offers a two-day, instructor-led course titled *Building Wireless Applications with Lutris Enhydra*. This course is intended primarily for Enhydra developers who want to create applications that serve content to cellphones or other wireless devices.

#### Lutris Enhydra Database Techniques

Lutris Technologies currently offers a two-day, instructor-led course titled *Database Techniques with Lutris Enhydra*. This course is intended primarily for Enhydra developers working on applications that require existing database platform support, Java database specialists and DBAs responsible for maintaining the data layer of an Enhydra application, and evaluators interested in seeing the database capability available through Enhydra.

### Contacting Lutris Education Services

For more information about training offerings, see the Lutris Training home page at http://www.lutris.com/training/index.html or call Lutris Customer Service toll-free at 1-877-688-3724, Monday–Friday, 8 a.m.–6 p.m. Pacific Time (outside of North America, please call 1-831-460-7590). You can also send email to training@lutris.com.

# Additional Enhydra information available on Enhydra.org

You can find a variety of information about open-source Enhydra at the Enhydra website: http://www.enhydra.org. The Enhydra website is the home of the Enhydra open-source community, one of Enhydra's greatest assets. The Enhydra community consists of numerous entities, including community sponsors, technology providers, users, and of course developers.

## Enhydra.org mailing lists

The Enhydra.org website includes archives of the various electronic mailing lists that serve as the backbone of the Enhydra community, as well as instructions on how to subscribe to the mailing lists.

Lutris encourages you to join one or more of the following Enhydra email lists:

- **Enhydra@enhydra.org**

  The Enhydra mailing list for developer interaction. The Enhydra project team monitors this list. It is the ideal place to get answers to your questions from fellow Enhydra developers.

- **Enhydra-daily@enhydra.org**

  A daily collection of all mail sent to enhydra@enhydra.org is sent to subscribers of this list.

- **Enhydra-digest@enhydra.org**

  A weekly digest of all mail sent to enhydra@enhydra.org.

- **EnhydraEnterprise@enhydra.org**

  The Enhydra Enterprise mailing list is tailored for those who are developing and deploying Enhydra applications on a large scale. Here you can find answers to the more detailed Enhydra questions, such as those on Enterprise Java Beans (EJB) and the Common Object Request Broker Architecture (CORBA).

- **EnhydraEnterprise-digest@enhydra.org**

  A weekly digest of all mail sent to EnhydraEnterprise@enhydra.org.

- **Enhydra-announce@enhydra.org**

  The mailing list for receiving Enhydra announcements.

For information and instructions on joining one or more of these lists, go to http://www.enhydra.org/community/mailingLists/index.html.

### Mailing list archives

You can search the combined Enhydra mailing list archives at http://www.enhydra.org/community/mailingLists/index.html.

## Enhydra.org working groups

Enhydra *working groups* bring together developers interested in creating new Enhydra applications and contributing new technologies or bug fixes for Enhydra.

Each working group provides access to the current project source code and to the project email list. This lets you communicate with the project leaders and other developers.

For information and instructions on joining one or more of these groups, go to http://www.enhydra.org/project/workingGroups/index.html.

### Documentation working group

The documentation working group is focused on facilitating developer-created documentation for open-source Enhydra and related technologies. The working group also provides a central point for documentation discussions and proposals.

Community members are encouraged to submit and collaborate on articles of any length, on topics of general interest to all Enhydra developers—from beginning to advanced.

For information and instructions on joining this groups, go to http://www.enhydra.org/project/workingGroups/index.html.

## Enhydra.org community documentation

The Enhydra website also has documentation provided by members of the community. For more information on this documentation, see http://www.enhydra.org/software/documentation/enhydra/index.html.

## Open-source software downloads

You can download the latest version of open-source Enhydra and other related software at: http://www.enhydra.org/software/downloads/index.html.

# Acknowledgments

As an open-source product, Enhydra benefits from the contributions of many developers around the world. Lutris would like to thank the members of the Enhydra community, particularly those who contributed information used in some form in this book.

# Chapter 2

# Using the Application Wizard to create Enhydra applications

The Application Wizard is a utility that creates a set of skeleton files for new Enhydra applications and components. It provides a command-line tool, a graphical wizard, and an API for integration with Kelp. For information on using the Application Wizard with Kelp, see "Using the Enhydra Application wizard" on page 52.

## Application Wizard generators

The Application Wizard uses two generators to create Enhydra projects—the Web Application generator and the Enhydra SuperServlet generator.

### Web Application generator

The Web Application generator produces the skeleton for an application that uses XMLC, Java Servlets using the Servlet API, business objects (BOs), and data objects (DOs).

### Enhydra SuperServlet generator

The Enhydra SuperServlet generator produces the skeleton for an application that uses XMLC, presentation objects (POs), business objects (BOs), and data objects (DOs).

## Using the Application Wizard command-line interface

The `appwizard` command takes predefined code-generator names as parameters and creates the skeleton code. The `appwizard` syntax is:

`appwizard <generator name> <generator options>`

The following table lists `appwizard`'s valid generator names:

**Table 2.1** Generator names for the Application Wizard

| Generator | Generator name | Description |
|---|---|---|
| Web Application | `webapp` | A servlet-based Web application that contains a welcome servlet and a redirect servlet. The welcome servlet uses XMLC to show the current time. The redirect servlet redirects the user to the welcome page. |
| Enhydra SuperServlet | `en3app` | An Enhydra SuperServlet-based application that uses the Presentation Object API. |

## Generator options

The following table lists the options for the application generators.

**Table 2.2** Options for application generator

| Option | Description |
|---|---|
| `client <Client_type>` | The application client type. Valid types are HTML, WML, cHTML, and XHTML. |
| `copyright <Copyright_string>` | Copyright string to add to the headers of generated files. |
| `copyrightfile <File>` | Text file containing the copyright string to add to the headers of generated files. |
| `nomake` | Do not generate GNU Makefiles for your project. |
| `overwrite` | Overwrite any existing files when generating files. |
| `project <Project_name>` | The project directory name. You must specify a project. |
| `package <Package_name>` | The top-level package for the generated Java files. You must specify a package. |
| `root <Root_path>` | The root path for your project. You must specify a root path for your application. |
| `noscript` | Do not generate shell scripts to start your application. |

For example, to generate an Enhydra super-servlet application for HTML clients from the command line, enter the following:

```
appwizard en3app -project myProjectName -package org.myorg.mypackage -client HTML -root
/home/user/enhydraApps/
```

# Using the Application Wizard GUI

The command `appwizard` with no arguments brings up the Application Wizard GUI for invoking the generators.

**Figure 2.1** Application Wizard GUI

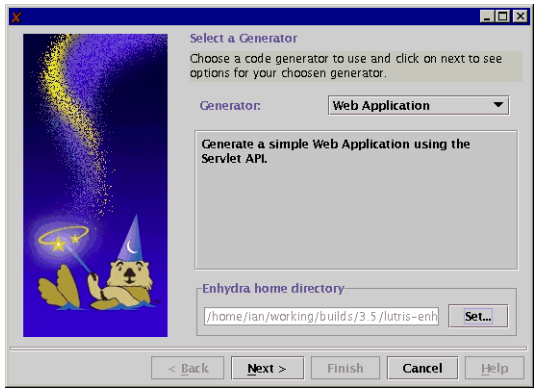



Figure 2.1 shows the generator selection screen in the Application Wizard GUI.

1 Select the appropriate generator from the drop-down list and click Next.

The valid generators are Web Application, a servlet-based application, or Enhydra super-servlet, a PO/BO/DO-based application.

**Note** The Enhydra Home Directory box displays the Enhydra installation the generators will use to find the templates. Click Set to navigate to the correct Enhydra installation.

2 Select the appropriate client type from the drop-down list.

The valid types are:

- HTML
- WML
- cHTML
- XHTML

3 Enter the project directory name in the Project Directory Name edit box.

4 Enter the package name for your project in the Package edit box.

5 Enter the root path for your Enhydra applications in the Root Path edit box. Click Set to set your Enhydra application root to something other than `$HOME/enhydraApps`. Click Next.

6 Select the appropriate radio button for inserting copyright text in the files generated by the Application Wizard.

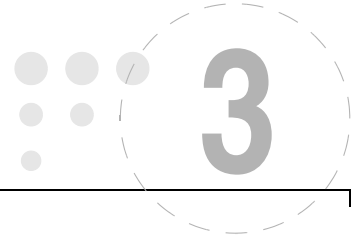
Choose File lets you specify a template file whose contents will be added to each generated file. Click Set to navigate to your template file.

Enter Copyright Text lets you enter the text in the edit box below.

Choose No Copyright if you do not want to insert copyright text in the generated files.

Click Next.

7. To generate `Makefile`s for use with the GNU `make` utility, check Create Makefiles.
8. To have the Application Wizard create shell startup scripts, check Create Shell Scripts.
9. Click Finish to generate your project.



# Chapter 3

# Using the Multiserver Administration Console

## Overview

The Enhydra Multiserver is a servlet runner that runs Enhydra super-servlet applications, Java servlets, and Java ServerPages (JSPs). The Multiserver has the following features:

- The Apache Tomcat servlet container, which makes the Multiserver a Servlet 2.2-compliant servlet container.
- Enhydra Director support, allowing the Multiserver to take requests from the most popular Web servers, including Apache, Netscape iPlanet, and Microsoft Internet Information Server.
- Supports multiple connection methods, allowing your application to take requests from a Web server or direct HTTP requests at the same time.
- A graphical administration console.
- Built-in HTTP debugger console, allowing you to monitor and debug requests for your Web application.
- Standalone HTTP Web server capability that supports the `http-basic-auth` protocol.
- Application partitioning within the Java Virtual Machine (multiple class loaders).
- Pure Java implementation.
- Runs on most UNIX systems, Windows 95, 98, NT, and 2000.
- Scales from single server to clustered server environments.

### Multiserver architecture

The Enhydra Multiserver uses three types of configuration files:

- The configuration file for the Multiserver itself, `multiserver.conf`, which contains configuration information listing for the applications running on the server. The file `<enhydra_root>/multiserver.conf` is an example that ships with the server.
- The configuration file for an Enhydra super-servlet application, normally called `<app>.conf`, where *app* is the name of the application. Copying an Enhydra

application's configuration file to `<enhydra_root>/apps` will make the application visible to the demonstration Multiserver installation that comes with the server.

- The `web.xml` file as defined in the Servlet 2.2 specification. To run servlets or JSPs on the Multiserver, they must be placed in a directory structure commonly referred to as a Web Application Archive, or WAR. The configuration file for a WAR is defined as the file `WEB-INF/web.xml`. WARs are discussed in more detail in "Adding a Web application archive" on page 27.

The Multiserver logs errors, warnings, and debugging messages to a file defined in `multiserver.conf`. In the demonstration installation, this file is `<enhydra_root>/log/multiserver.log`.

## Connection methods

The Enhydra Multiserver allows a Web application to take requests from multiple sources at the same time. For example, you can configure your application to accept Web server requests and direct HTTP requests at the same time. With that configuration, you can access your application through the Web server, or directly from the Multiserver, all at the same time. This capability is valuable for debugging configuration and load balancing problems.

Each method of listening for client requests is called a *connection method*. The Multiserver supports three connection methods: HTTP, HTTPS, and Enhydra Director. In the example above, the Multiserver has an HTTP connection method running on a port, like 8000, listening for HTTP traffic. It also has an Enhydra Director connection running on another port, like 8080, listening for requests forwarded from a Web server. Enhydra Director has its own protocol.

The connection methods are built on an underlying architecture that uses *channels* and *filters.* Each channel defines a connection method, a URL prefix, and an application. Filters are an added layer of abstraction used by the server to monitor incoming requests. The traffic debugger uses the Multiserver's filter capability to display requests for an application. Filters also enable logging for your application.

## Enhydra class loader

To avoid class loading problems, you must understand the Multiserver's `CLASSPATH` and the Enhydra class loader.

Any Java application is written as a set of classes. Each time a previously unreferenced class is accessed, the Java Virtual Machine (JVM) automatically loads the class through the class loader. The class loader looks for the class file in a sequential list of directories or archives (ZIP files or JAR files). This sequential list is known as the *`CLASSPATH`*.

In the Enhydra Multiserver, each application receives its own class loader. The Multiserver uses class loaders for a number of reasons. Separate class loaders let the server start and stop applications without stopping the server. Class loaders also prevent applications from conflicts over class ownership, and lets them run autonomously.

When an Enhydra super-servlet application requires a class, the following sequence of events takes place:

1. The application's class loader tries to find the class in the directories or archives defined in the Enhydra application's configuration file. If the class is found, a private version of that class is loaded for the application.
2. If the class is not found, the class loading responsibility is passed to the system class loader. This class loader looks in all directories and archives defined in the system `CLASSPATH`. The system `CLASSPATH` is a combination of the default Java `CLASSPATH` and the `CLASSPATH`s added by the Enhydra Multiserver start script, typically defined by the `CLASSPATH` environment variable. Any class loaded by the system class loader is shared by all applications, and therefore cannot be easily modified.

With a servlet application deployed as a WAR (a Web application archive, as discussed in "Creating a WAR file" on page 44), the class loading sequence is similar, but the `CLASSPATH` is constructed differently. The Servlet 2.2 specification defines two subdirectories of the directory `WEB-INF`. The first is called `classes` where expanded class files are placed, and the second is called `lib` where JAR files are placed. On startup, the `classes` directory and every JAR file in the `lib` directory are automatically added to the application's `CLASSPATH`. See "Creating a WAR file" on page 44 for more information.

**Note** Some JDBC drivers can only be loaded by the system class loader. If you encounter JDBC problems, try adding the location of the JDBC classes to the system `CLASSPATH`, and remove the locations from the application's `CLASSPATH`.

# Launching the Admin Console

You can start the Multiserver and its Admin Console simultaneously from the command line.

1. Type the following command at the shell prompt to start the Multiserver:

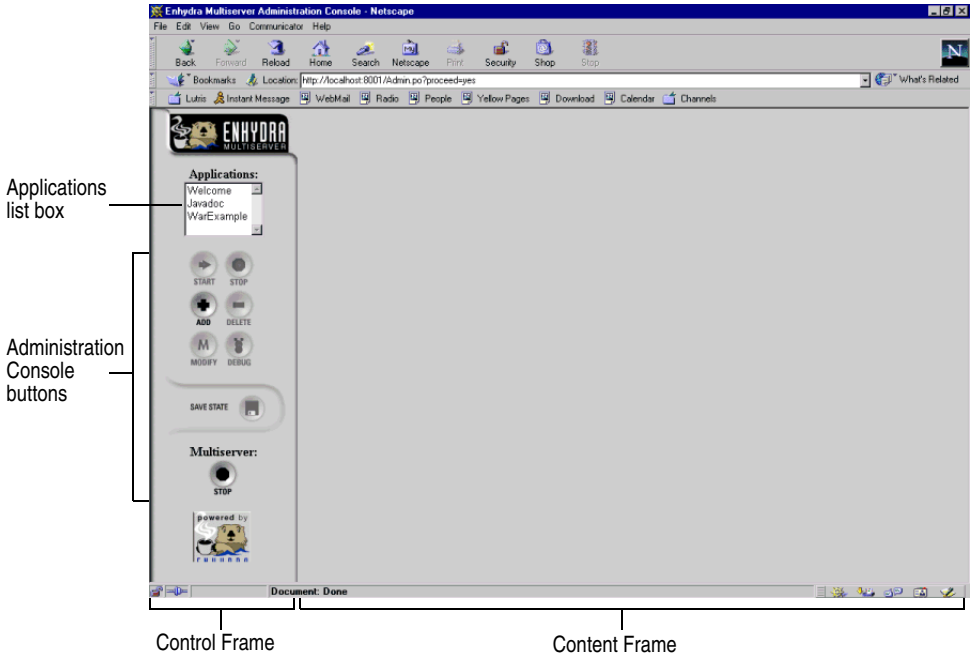
   `<`*`enhydra_root`*`>/bin/multiserver`

   where *`enhyra_root`* is the root of your Enhydra installation. Invoking the multiserver command without specifying a `multiserver.conf` as an argument brings up the server in the demonstration installation.

   If this fails, your `PATH` is probably not configured properly. See Chapter 2, "Installation," of the *Getting Started with Lutris Enhydra*.
2. In your Web browser, open the URL `http://<`*`machine_name`*`>:8001`, where *`machine_name`* is the name of the server where you started the Enhydra Multiserver. This default URL may be modified by the user.
3. You will be prompted for a user name and password. The default username is `admin` and the password is `enhydra`. To change these settings, see "Modifying an application" on page 29.

4 Your browser connects to the Enhydra Multiserver Admin Console and provides access to the console tools, as shown in Figure 3.1, "Multiserver Administration Console display":

**Figure 3.1** Multiserver Administration Console display




The control frame on the left contains the Enhydra Multiserver Admin Console tools, which are used to select, start, stop, add, delete, modify, or debug an application. There is also a button, Save State, for saving the current state of the server.

The content frame on the right displays status screens or dialog boxes for the selected application.

# Viewing application status

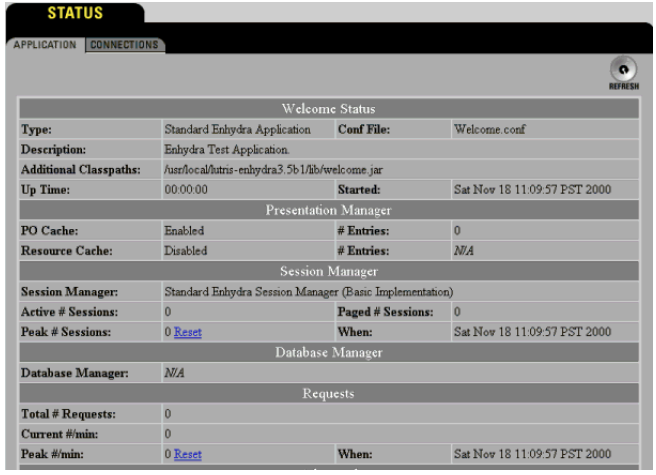
The Applications window, located at the top of the control frame in the Admin Console, contains a list of all applications that have been added to the Enhydra Multiserver. Upon initial start-up of your server, the Welcome, JavaDoc, and WarExample samples appear in this window as the only installed applications.

**Note** There are two kinds of Web applications:

- Enhydra super-servlet applications, created with Enhydra development tools
- Servlet applications, otherwise known as WAR files (see "Adding a Web application archive" on page 27).

To manage an application, select its name in the Applications window. This displays its status information in the content frame, and any of the console tools that represent valid operations for that application are activated at this time.

**Figure 3.2** Application Status window




## Understanding the Application tab

The Status window in the content frame provides basic information for the selected application, such as the number of requests if the application is currently running, requests per session, and uptime. Some of these are read from `<appName>.conf`. Other controls in this window provide for refreshing the screen and accessing online help.

### Enhydra super-servlet Application Status window

The following tables represent the standard fields in the Application status display.

**Table 3.1** Application Status: General information

| Field | Description |
|---|---|
| Type | Type of application, usually Standard Enhydra Application. For a custom application, its type will be displayed in this field. |
| Conf File | Name of the configuration file, found in `/<enhydra_root>/apps/`, for example, `Welcome.conf`. |
| Description | Modifiable description of the application. |
| Additional Classpaths | Location of additional `CLASSPATH`s necessary for the successful instantiation of this application. For example, `/usr/local/<enhydra_root>/`. |
| Up Time | Elapsed time since application startup. If the application has not been started, a `Not running` message appears here. |
| Started | Time of the most recent application startup. If the application has not been started, a `Not running` message appears here. |

**Table 3.2** Application Status: Presentation information

| Field | Description |
|---|---|
| PO Cache | Indicator of whether presentation objects are being cached or not. Values are `enabled` or `disabled`. |
| # Entries | Number of entries currently in the PO cache. |
| Resource Cache | Indicator of whether resources are being cached or not. Values are `enabled` or `disabled`. |

**Table 3.3** Application Status: Session information

| Field | Description |
|---|---|
| Session Manager | Type of session manager in use, usually Standard Enhydra Session Manager. If you have installed a custom session manager, its type is displayed. If there are no active sessions, `N/A` is displayed. |
| Active # Sessions | Number of currently active sessions. If there are no active sessions, `N/A` is displayed. |
| Peak # Sessions | Peak number of sessions attained by this application during its current instantiation. If there are no active sessions, `N/A` is displayed. The Reset link allows you to reset this value to 0. |
| When | Time and date when the peak number of sessions occurred. |

**Table 3.4** Application Status: Database information

| Field | Description |
|---|---|
| Database Manager | Type of database manager in use, usually Standard Enhydra Database Manager. If you have installed a custom database manager, its type is displayed. If the application does not require the use of the database manager, `N/A` is displayed. |
| Active # Connections | Number of active connections to the database manager, during the application's current instantiation. If the application does not require the use of the database manager, this field is not displayed. |
| Peak # Connections | Peak number of connections to the database manager during the application's current instantiation. If the application does not require the use of the database manager, this field is not displayed. The Reset link allows you to reset this value to zero. |
| When | Time and date when the peak number of connections occurred. If the application does not require the use of the database manager, this field is not displayed. |

**Table 3.5** Application Status: Request information

| Field | Description |
|---|---|
| Total # Requests | Total number of requests of this application during its current instantiation. If the application is not running, `N/A` is displayed. |
| Current #/min | Current number of requests of this application per minute, during its current instantiation. If the application is not running, `N/A` is displayed. |
| Peak #/min | Peak number of requests of this application per minute, during its current instantiation. If the application is not running, `N/A` is displayed. The Reset link allows you to reset this value to zero. |
| When | Time and date when the peak number of requests per minute occurred. |

The Advanced section of the application status display is an optional, application-specific area. Information is only displayed here if the application has a `toHtml` method implemented.

## Servlet Status window

If your selected application is a servlet, the following tables represent the standard fields in the Servlet Status window.

**Figure 3.3** Servlet Status window


The following tables provide information about the Servlet Status window.

**Table 3.6** Servlet Status: General information

| Field | Description |
|---|---|
| Type | By default, displays `Servlet`. |
| Doc Root | File system location to associate with the document root of the servlet. For example, `/tmp`. |
| Classname | Class name of the servlet. |
| Description | Modifiable description of the servlet. |
| Initial Arguments | Initial arguments passed to the servlet, represented in name/value pairs, for example, `max = 256`. The text `no initial arguments` is displayed if there are none. |
| Additional Classpaths | Location of additional `CLASSPATH`s necessary for the successful instantiation of this servlet. |
| Up Time | Elapsed time since servlet startup. If the servlet has not been started, the text `Not running` appears in this field. |
| Started | Time of the most recent servlet startup. If the servlet has not been started, the text `Not running` appears in this field. |

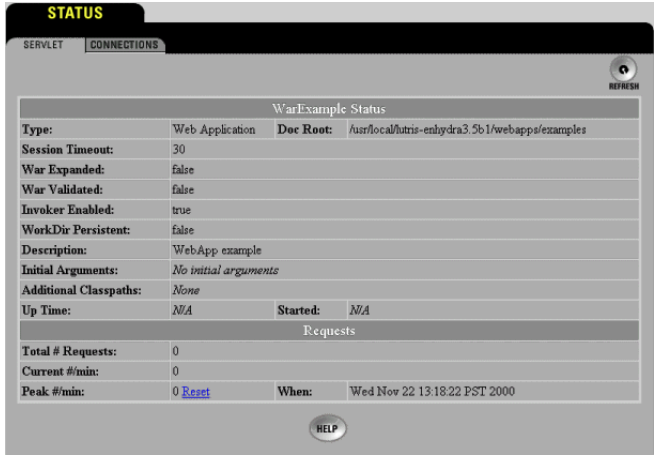
**Table 3.7** Servlet Status: Request information

| Field | Description |
|---|---|
| Total # Requests | Total number of requests of this servlet during its current instantiation. If the servlet is not running, `N/A` is displayed. |
| Current #/min | Current number of requests of this servlet per minute, during its current instantiation. If the servlet is not running, `N/A` is displayed. |
| Peak #/min | Peak number of requests of this servlet per minute, during its current instantiation. If the servlet is not running, `N/A` is displayed. The Reset link allows you to reset this value to zero. |
| When | Time and date when the peak number of requests per minute occurred. |

## WAR Status window

If your application is a WAR (Web application archive, see "Creating a WAR file" on page 44) file, the Status window, as shown in Figure 3.4, displays fields as described in Table 3.8.

**Figure 3.4** WAR Status window




The following table provides information about the Status window.

**Table 3.8** WAR status

| Field | Description |
|---|---|
| Type | By default, displays `Web Application`. |
| Session Timeout | Period of time, in minutes, for which the session may remain idle before timing out. |
| WAR Expanded | Whether the WAR file has been extracted. |
| WAR Validated | Whether the WAR file has been validated. |
| Invoker Enabled | Indicator of whether `/servlets/*` syntax is in use, `true` or `false`. |
| WorkDir Persistent | Indicator of whether to the work directory is saved after the termination of the current Multiserver session, `true` or `false`. |
| Description | Plain English description of this archive. |

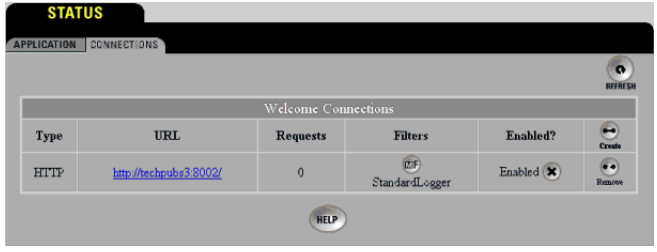
**Table 3.8** WAR status (continued)

| Field | Description |
|---|---|
| Initial Arguments | Initial arguments passed to the servlet, represented in name/value pairs, for example, `max = 256`. The text `no initial arguments` is displayed if there are none. |
| Additional Classpaths | Path to the root directory of the Web archive, after it is decompressed. |
| Uptime | Period of time, in seconds, for which the session may remain idle before timing out. |

## Understanding the Connections tab

Selecting the Connections tab in the Status window displays the connection status for the selected application. This section also contains controls for adding new connections, removing existing connections, enabling and disabling connections, and modifying the filters associated with any of the connections.

**Figure 3.5** Connections Status window




The following table provides information about the Connections Status window.

**Table 3.9** Connections Status window

| Field | Description |
|---|---|
| Type | Type of connection.<br>• HTTP: Connect through a Web browser. No additional steps are needed.<br>• HTTPS: Connect through secure HTTP.<br>• Enhydra Director: Connect through a Web server configured to use Director. |
| URL | URL that may be used to access the application through the particular connection type. These are live links through which you may access the application, when both the application and the Multiserver are running. |
| Requests | Number of requests that have been received by the application through the specified connection. |
| Filters | Filters associated with the current connection. These may be used to filter selectively for certain types of data or conditions, each viewable in its own log file. |
| Enabled? | Status of the current connection, `Enabled` or `Disabled`. |
| Create | Button for adding new connections. |
| Remove | Button for removing each of the existing connections. |

# Starting an application

Use the Start button in the control frame to start an installed application. If the application is already running, the Start button is disabled. Since the Start operation changes the state of the application, it activates the Save State button. If you want this application to be automatically started the next time the Enhydra Multiserver is started, use the Save State button to save this setting, as described in "Saving the current state of the console" on page 43.

To start an application:

1 Select an application from the Applications window by clicking on its name.
2 Click the Start button.

# Stopping an application

Use the Stop button in the control frame to stop an installed application. If the application is not currently running, the Stop button is disabled. Since the Stop operation changes the state of the application, it enables the Save State button. If you want this application to be stopped the next time the Enhydra Multiserver is started, use the Save State button to save this setting, as described in "Saving the current state of the console" on page 43.

To stop an application:

1 Select an application from the Applications window.
2 Click the Stop button.

If an application has active users, you are prompted to confirm you want to stop the application.

# Adding an application

For the most part, all types of applications are handled in the same way in the Admin Console. However, in the case of adding an application, the console provides different windows for adding an Enhydra application, a single servlet, or a WAR file.

Whichever one you add, though, the process consists of two parts: adding the application, and creating at least one connection for it.

## Adding an Enhydra super-servlet application

Use the Add button in the control frame to add an application to the Enhydra Multiserver. Before adding it, its application configuration file must be made accessible to the console, as described below.

## Preparing the configuration files

1 Copy the application's configuration file, `application.conf`, to `/usr/local/<enhydra_root>/apps/`.

2 In the configuration file, comment out the first line of the `Server.Classpath` variable.

3 Directly beneath this, uncomment the second line.

4 Set the `Server.Classpath` variable equal to the absolute path to the application's JAR file:

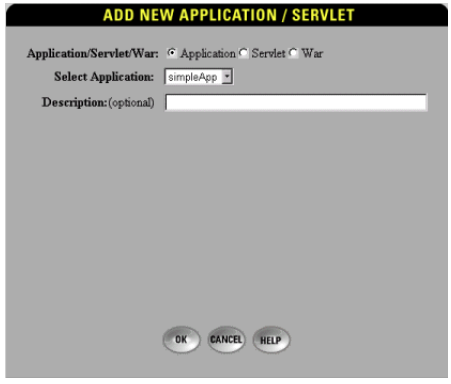
```
Server.Classpath[] = "/<application_root>/output/lib/<appName>.jar"
```

5 Save and close the configuration file.

## Adding the application in the console window

1 In the Admin Console window, click the Add button to display the Add New Application/Servlet window.

**Figure 3.6** Add New Application window




2 Select the Application radio button at the top of the window to display the Application tab section, if it isn't already displayed.

3 Select an application from the pull-down Select Application menu. The names in the menu represent applications with configuration files in the `/usr/local/<enhydra_root>/apps/` folder, but which have not already been added to the console. For each such file, the associated application name appears in the menu. The menu is empty if there are no more applications available to add.

4 Optionally, complete the Description field.

5 Click OK twice to return to the admin console. The Applications window is updated to reflect the new application, which is selected by default. Its status appears in the content frame.

6 Create a connection for the application according to the instructions in "Connecting the new application" on page 28.

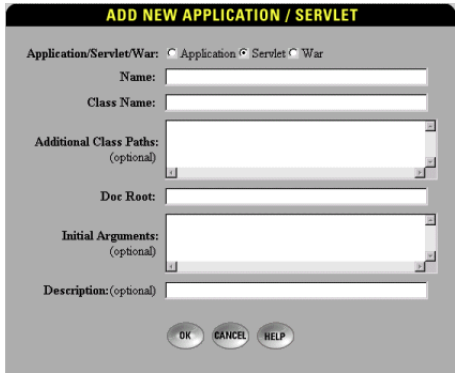
## Adding a single servlet

Although Enhydra provides the capability for adding a single servlet to the console, as described below, users will typically find it more efficient to add all of their servlets in the form of a Web archive (WAR), as discussed in "Adding a Web application archive" on page 27.

Follow these steps to add a servlet application to the Console:

1 Click the Add button to display the Add New Application/Servlet window and select the Servlet radio button to toggle to the Servlet tab section.

**Figure 3.7** Add New Servlet window




2 Complete the fields as described in the following table.

**Table 3.10** Adding Servlets

| Field | Description |
|---|---|
| Name | Identifier string used to refer to this servlet. Required. |
| Class Name | Name of the class to instantiate for this servlet. This class must implement the servlet interface. Required. |
| Additional Class Paths | Any additional `CLASSPATH`s required for this servlet. Multiple class paths, each on its own line separated by carriage returns, can be specified in this field. Optional. |
| Doc Root | Root of the servlet's filesystem on disk. Required. |
| Initial Arguments | Any initial arguments required for this servlet. These are made available to the servlet's `init()` method. Multiple initial arguments, each on its own line separated by carriage returns, can be specified in this field. Optional. |
| Description | Description to be associated with the servlet being added. Optional. |

3 Click OK to return to the console window. In the updated Applications window, the new servlet is selected by default. Its status appears in the content frame.

4 Create a connection for the servlet according to the instructions in "Connecting the new application" on page 28.

## Adding a Web application archive

A Web archive (WAR) is a collection of servlets bound together for convenient administration. Once configured, a WAR may be moved from one server to another without the need for further modification.

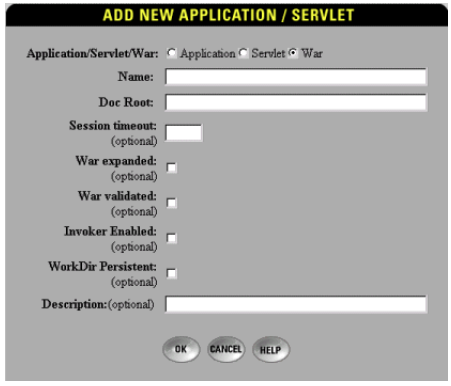
For information on creating a WAR file, see the section "Creating a WAR file" on page 44.

Follow these steps to add your WAR file to the Console.

1 In the Administration Console window, click the Add tool to display the Add New Application/Servlet window.

2 Select the WAR radio button to switch to the WAR tab section.

**Figure 3.8** Add New WAR window




3 Complete the fields as described in the following table.

**Table 3.11** Adding WARs

| Field | Description |
|---|---|
| Name | Name of the archive, not necessarily the file name. |
| Doc Root | Path to the root directory of the Web archive, after it is decompressed. |
| Session Timeout | Period of time, in minutes, for which the session may remain idle before timing out. |
| WAR Expanded | Leave selected. |
| WAR Validated | Leave selected; see Tomcat documentation for details. |
| Invoker Enabled | Select if you want to use `/servlets/*` syntax. |
| WorkDir Persistent | Select if you want to save the work directory after the termination of the current Multiserver session. |
| Description | Plain English description of this archive. |

4 Click OK to return to the console. The Applications window is updated to reflect the new application, and it is selected by default. Its status appears in the content frame.

5 Create a connection for the WAR according to the instructions in “Connecting the new application” on page 28.

## Connecting the new application

After adding an application to the console, you must specify how it should connect to the outside world. The Enhydra Multiserver lets you choose:

- One-to-one: one port to one application.
- One-to-many: one port to many applications.
- Many-to-many: multiple ports to multiple applications.

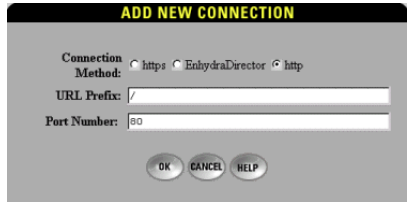
Use the Add New Connection window to add and remove connections.

1 Select the desired application in the Applications window.

2 Click the Connections tab in the Status window.

3 Click Create to display the Add New Connection screen. In this screen, select the desired connection method.

**Figure 3.9** Add New Connection window




- **HTTP:** This method opens a socket in the Multiserver on the local machine using HTTP, connecting through a Web browser. HTTP is a good choice for a development environment because of its simplicity.
- **HTTPS:** Connect through secure HTTP. This option is only available if you have configured your Enhydra installation with Sun's Java Secure Socket Extension Kit. For more information, see “Using SSL with Enhydra” on page 179 for more information.
- **Enhydra Director:** This works in conjunction with a Web server configured to use Enhydra Director. If you use this option, make sure to set the URL prefix and port number to those used in your Director configuration file. See “Configuring your application” on page 167 and “Editing enhydra_director.conf” on page 168 in Chapter 9, “Using Enhydra Director” for more information.

4 Enter the URL prefix—the part of the URL that immediately precedes the file name. Or, leave it set to the default, the forward slash character.

5 Enter the port number. If you do not have root access on your system, you must pick a number above 1000. The highest valid port number is 65535. If the specified port is already being used, the attempt to add the connection fails. The `netstat` command shows the state of the ports on UNIX systems.

6 Click OK to return to the Connections screen.

To run your newly added application, see "Starting an application" on page 24.

### Removing a connection

Click Remove Connection button to remove an application. A pop-up window will ask for confirmation before the connection is removed. Click OK to remove the connection.

## Deleting an application

Use the Delete button to remove an application from the Enhydra Multiserver. If the selected application is currently running, the Delete button is disabled. You must stop a running application prior to removing it, as described in "Stopping an application" on page 24.

1 Select the application from the Applications window.

2 Click the Delete button in the console tools section.

3 Click OK to confirm.

4 The Remove Application (or Servlet, or WAR) popup appears, confirming the deletion.

When an application is deleted, its entries are removed from the Enhydra Multiserver configuration file. The application itself, however, is not removed from the file system, and its configuration file is not removed from `<enhydra_root>/apps/`. You may add the application again at any time, using the Add button, described in "Adding an application" on page 24.

Since the Delete operation changes the state of the application, performing this operation activates the server Save State tool. If you want this application to not be present the next time the Enhydra Multiserver is started, click Save State to save this setting.

## Modifying an application

The Enhydra Multiserver Admin Console provides a graphical user interface and "one-stop shopping" for modifying your Enhydra applications, Java servlets, and WAR files. The fields displayed in the console reflect properties found in two configuration files:

- `/usr/local/<enhydra_root>/multiserver.conf`, the basic Multiserver configuration file
- `/usr/local/<enhydra_root>/apps/<appName.conf>`, the configuration file specific to the application designated by its name.

However, the configuration file for the console itself, `/usr/local/<enhydra_root>/multiserverAdmin.conf`, must be modified by hand in any text editor.

Additionally, some of the variables in the other two configuration files are *not* reflected in the console, and must also be modified manually, as necessary.

## Modifying configuration files by hand

The following tables identify the modifiable variables found in each of the files. You can edit these files in any text editor or word processing program.

### Multiserver configuration file

The file `multiserver.conf` is a simple Enhydra Multiserver configuration file that describes applications running on the server. Many of these properties are modifiable in the console, which overwrites this file when you save changes you make in the Modify screens.

**Table 3.12** Basic Multiserver configuration file (multiserver.conf)

| Parameter | Description |
|---|---|
| `Server.ConfDir` | Where the config files are. |
| `Server.LogFile` | Set up logging. |
| `Server.LogToFile[]` | Specifies logging levels. |
| `Server.LogToStderr[]` | Specifies logging levels. |
| `Application.simpleApp.ConfFile` | Specify only one application (no graphical administration). |
| `Application.simpleApp.Description` | Plain English description of the application. |
| `Application.simpleApp.Running` | Indictor of whether the application is running. |
| `Connection.p9000.Type` | Connection type (HTTP, HTTPS, Enhydra Director). |
| `Connection.p9000.Port` | Port to bind. |

### Admin Console configuration file

The file `multiserverAdmin.conf` contains properties used to configure the console itself, including the password and user name properties. For additional application-specific properties, see “Enhydra application configuration file” on page 31.

**Table 3.13** Admin Console configuration file (multiserverAdmin.conf)

| Parameter | Description |
|---|---|
| `Admin.Username` | User name required to access the admin app. This is used in conjunction with the “Basic Auth” HTTP authentication method. If both the user name and password are "" (blank), then no authentication is required. |
| `Admin.Password` | Same as above. |

**Table 3.13** Admin Console configuration file (multiserverAdmin.conf) (continued)

| Parameter | Description |
|---|---|
| `Admin.DebugQueueSize` | Maximum number of transactions (a request and the resulting response) to store when debugging. |
| `Admin.SaveResponseData` | Indicator of whether each of the stored transactions should keep a copy of the data written out to the net when debugging. Consumes additional memory. Be careful when using this option and large values of `Admin.DebugQueueSize` simultaneously. Values are `true` or `false`. |

## Enhydra application configuration file

The file `<appName>.conf` contains properties for the specified Enhydra application.

**Table 3.14** Application configuration file (<appName>.conf)

| Parameter | Description |
|---|---|
| `Server.ClassPath[]` | Comma-separated `CLASSPATH` directories and files used by this application. Assumes run from the `output` directory for debugging. If you run from the JAR, you must rebuild after every change to the app. |
| `Server.AppClass` | The fully qualified name of the application class. |
| `Server.PresentationPrefix` | Prefix used to derive presentation object class names and paths from URLs. Assumes run from the `output` directory for debugging. |
| `Server.AutoReload` | Flag to indicate that application classes and resources should be reloaded automatically if *any* file in the `CLASSPATH` changes.<br>**Note:** This is a debugging option and may slow down the performance of your application. The `CLASSPATH` should *not* contain any directories (or underlying directories) that contain constantly changing files, such as log files, for example, the application's `output` directory, which contains the application log files in the underlying log directory. |
| `Application.DefaultUrl` | If the URL `/` (Web server root) for this application is accessed, the user will be redirected to this URL. This should be a relative URL. |
| `Server.XMLC.AutoRecompilation` | Enable or disable recompilation of XMLC document classes when the source files are out of date. For this to work, XMLC document classes must have been compiled using the `-for-recomp` flag. The application must be running with its classes (at least XMLC classes) in directories rather than in a JAR. The HTML files must be on the `CLASSPATH` in the same package as the class that was generated from them. |
| `Server.XMLC.AutoReload` | Enable or disable automatic reloading of XMLC document classes when the class file has changed. This is a subset of the `Server.XMLC.AutoRecompilation` and checks for XMLC document classes that have been loaded being out of date, however it doesn't check or recompile source files. If `Server.XMLC.AutoRecompilation` is set, this option is ignored. |
| `SessionManager.Lifetime` | Maximum number of minutes a user session can last, after which the session is terminated regardless of activity. Setting the value to less than or equal to 0 gives an infinite lifetime. The default is 1440 minutes (24 hours). |

**Table 3.14** Application configuration file (<appName>.conf) (continued)

| Parameter | Description |
|---|---|
| `SessionManager.MaxIdleTime` | Maximum number of minutes a user may be idle before being logged off. Setting the value to less than or equal to 0 gives an infinite idle time. You should not set this and SessionLifetime both to zero, else sessions will never expire. The default is 30 minutes. |
| `SessionManager.MaxNoUserIdleTime` | Maximum number of minutes a session not associated with a logged in user (perhaps login pending) may be idle before the session is deleted. Setting the value to less than or equal to 0 will give an infinite idletime. You should not set this and SessionLifetime both to zero else sessions will never expire. The default is 30 minutes. |
| `SessionManager.IdleScanInterval` | Seconds between idle user scans in the Session Manager. The default is 30 seconds. |
| `SessionManager.RandomizerIntervals[]` | Set of varying intervals, in seconds, to use for adding user-generated entropy to the random number generator. It is a good idea to use several different prime numbers. Setting this key is only necessary in very security-conscious environments. Suitable defaults are used if this is not specified. |
| `PresentationManager.CacheClasses` | Enables or disables caching of presentation object classes in memory. Optional; the default is `true`. |
| `PresentationManager.CacheFiles` | Enables or disables caching of files (HTML, GIF, etc.) that are served as part of the application. Optional; the default is `false`. |

### Changing the console username and password

The user name and password properties deserve special mention because they are buried in the `/usr/local/<enhydra_root>/MultiserverAdmin.conf` file. Simply modify the variables `Admin.Username` and `Admin.Password` as desired.

## Modifying configuration files in the console

Use the Modify button to change some of the configurable attributes of your application. In the Modify screens, you can modify attributes associated with the selected application, which appear in the application's own configuration file, `/usr/local/<enhydra_root>/apps/<appName>.conf` or in the Enhydra Multiserver configuration file `/usr/local/<enhydra_root>/multiserver.conf`.

For properties in the `multiserverAdmin.conf` file, or for properties from the other files that are not reflected in the console, you must make modifications manually in any text editor.

If the application you have selected is currently running, the Modify button is disabled.

1. Select an application from the Applications window.
2. Click the Stop button to stop running the application and enable the Modify button.

3 Click the Modify button to display the Status screen in the content frame. It contains several configuration tab sections: Application/Servlet, Sessions, Database, Advanced.

   If modifying servlets or WAR files, there will only be one tab, Servlet or WAR, respectively, displayed. The other tabs are specific to Enhydra applications and only appear when modifying one of them.

4 Select the tab containing the information you wish to modify, and consult the appropriate section(s) below for further information.

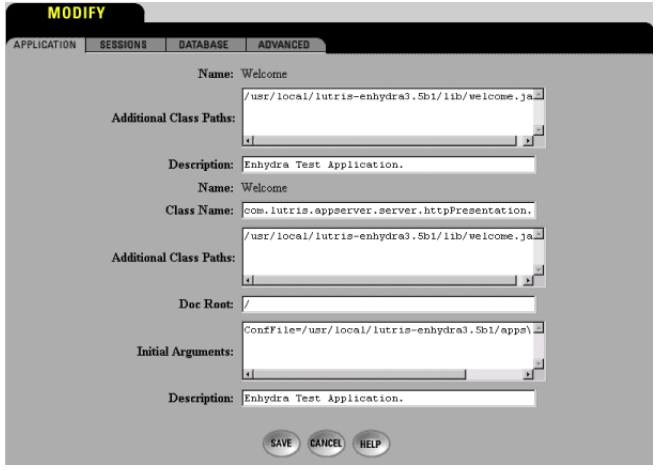
In most cases, specific parameters available in each tab section depend upon the individual application.

On each tab, click the Save button to write back modifications to the appropriate configuration file, or Cancel to cancel any changes and return to the Status screen for the selected application. There is no need to use the Save button in every tab section; clicking it once when you've finished with all of the sections is sufficient.

## Application tab

This tab represents the general configurable parameters for the selected application. The Additional Classpaths field is read from and saved to the application's configuration file, `/usr/local/<enhydra_root>/apps/<appName>.conf`. The Description field is read from and saved to the Enhydra Multiserver configuration file, `/usr/local/<enhydra_root>/multiserver.conf`.

**Figure 3.10** Application tab section



The following table provides information about application form fields.

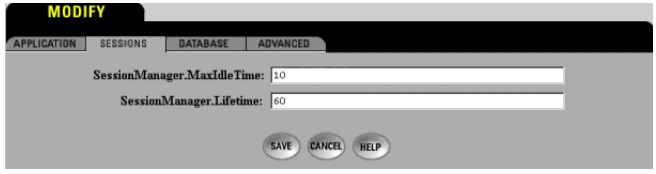
**Table 3.15** Modifying applications: Application form fields

| Field | Description |
|---|---|
| Name | Identifier string used to refer to this application. It is not modifiable once set, and appears here for reference only. |
| Additional Class Paths | Any additional CLASSPATHs required for this application. Multiple class paths, each on its own line separated by carriage returns, can be specified in this field. |
| Description | Description to be associated with the application being added. Optional. |

## Sessions tab

This tab section presents session-specific parameters for the selected application. All of the fields are read from and saved to the application's configuration file, /usr/local/<*enhydra_root*>/apps/<*appName*>.conf.

**Figure 3.11** Sessions tab section




Not all of these attributes may appear for your selected application. The following table represents the list of all available attributes.

**Table 3.16** Modifying Application: Sessions form fields

| Field | Description |
|---|---|
| IdleScanInterval | Frequency, in seconds, at which the session manager tests whether any sessions should be expired. Must be greater than zero. |
| SessionManagerLifetime | Maximum number of minutes a session is valid. If this value is zero, then there is no lifetime limit. Required. |
| SessionMaxNo-UserIdleTime | Maximum number of minutes a session that does not have a User object associated with it may be idle before the session is expired deleted). If this value is less than or equal to zero, then the session may be idle indefinitely. Default is the same as MaxIdle. |
| SessionMaxIdle-Time | Maximum number of minutes a session may be idle before the session is expired (deleted). If this value is less than or equal to zero, then the session may be idle indefinitely. |
| Randomizer-Intervals | Set of varying intervals, in seconds, to use for adding user-generated entropy to the random number generator. It is a good idea to use several different prime numbers. Multiple intervals, each on its own line separated by carriage returns, can be specified in this field. |

## Database tab

This tab presents database-specific parameters for the selected application. All of the fields are read from and saved to the application's configuration file, `/usr/local/<enhydra_root>/apps/<appName>.conf`.

There may be no modifiable database attributes, depending on the specific application selected. If so, the text `Nothing to modify` appears under the Database tab.

**Table 3.17** Modifying Application: database form fields

| Field | Description |
|---|---|
| `Debug` | Specify true to enable Query and Transaction logging, false to disable it. Optional; false if not specified. |
| `DB.<dbname>.Jdbc-Driver` | JDBC driver to use to access the database. Required. |
| `DB.<dbname>.Class-Type` | Class of the logical database implementation or a symbolic name if one of the standard types is selected. This is recommended because, although JDBC abstracts the data access, the functionality of each database is slightly different, and this parameter allows for optimized usage. Optional. Standard types are:<br>• Oracle: For Oracle 7/8 usage<br>• Informix: For Informix usage<br>• Sybase: For Sybase usage<br>• Msql: For Microsoft MSQL usage<br>• Standard: For all other JDBC databases |
| `DB.<dbname>.ObjectId.CacheSize` | Number of object IDs to cache between database queries. Optional. If not specified, defaults to 1024. |
| `DB.<dbname>.ObjectId.MinValue` | Starting number of Object ID allocation. This is only used if the Object ID table is empty, and thus is useful in development and testing. Optional. If not specified, it defaults to 100000000000000000. Note that the largest number that can be associated with an OID in the Enhydra Multiserver is `database:DECIMAL(19,0)`. |
| `DB.<dbname>.Connection.Url` | JDBC URL of the database. Mandatory. e.g., `jdbc:sequelink://dbHost:4000/[Informix]; Database=dummy` |
| `DB.<dbname>.Connection .MaxPoolSize` | Maximum number of open connections to the database. Optional. If not specified, defaults to 0. A value of 0 means that connections are allocated indefinitely or until the database (JDBC) refuses any new ones. |
| `DB.<dbname>.Connection .QueryTimeout` | Amount of time, in seconds, that a query blocks before throwing an exception. If less than or equal to 0, then the query will not block. Optional. If not specified, defaults to 0. This is not implemented by all logical databases. |
| `DB.<dbname>.Connection.User` | Database user used to access the database. Mandatory. |
| `DB.<dbname>.Connection .Allocation-Timeout` | Maximum amount of time, in milliseconds, that a thread waits for a connection from the connection allocator before an exception is thrown. This prevents possible deadlocks. If less than or equal to zero, the allocation of connections will wait indefinitely. Optional. If not specified, defaults to 1000 (ms). |

**Table 3.17** Modifying Application: database form fields (continued)

| Field | Description |
|---|---|
| DB.<*dbname*>.Connection .Transaction Timeout | Amount of time, in seconds, that a transaction blocks before throwing an exception. If less than or equal to zero, the transaction will not block. Optional. If not specified, defaults to 0. This is not implemented by all logical databases. |
| DB.<*dbname*>.Connection.Logging | Specify true to enable SQL logging, false to disable it. Optional. Defaults to false if not specified. |
| DB.<*dbname*>.Connection.Password | Database user's password. Required. |
| DefaultDatabase | Default logical database used by this application. Optional. If not specified, then the first entry in the Databases field is used. |
| Databases | List of logical SQL database names. |

**Figure 3.12** Database tab section


## Advanced tab

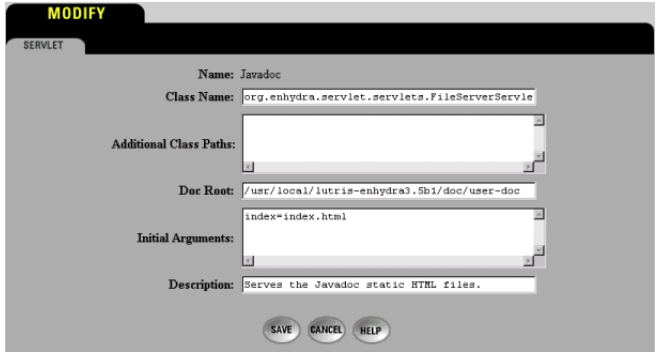
This tab section presents all of the application-specific parameters for the selected application that are not related to sessions or databases. Consult your application's documentation on this form's fields. All fields are read from and saved to the application's configuration file, /usr/local/<*enhydra_root*>/apps/<*appName*>.conf.

**Figure 3.13** Advanced tab section


## Servlet tab

This tab section represents the only configurable parameters for servlets. All of the attributes available for modification here are read from and saved to the Enhydra Multiserver configuration file, /usr/local/<*enhydra_root*>/multiserver.conf.

**Figure 3.14** Servlet tab section




The following table provides information about modifying servlets.

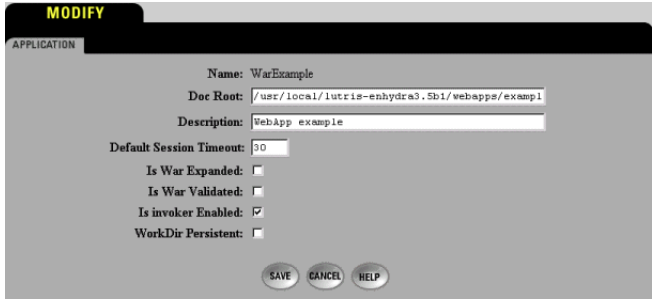
**Table 3.18** Modifying servlets

| Field | Description |
|---|---|
| Name | Identifier string used to refer to this servlet. It is not modifiable once set, and appears here for reference only. |
| Class Name | Name of the class to instantiate for this servlet. This class must implement the servlet interface. Required. |
| Additional Class Paths | Any additional CLASSPATHs required for this servlet. Multiple classpaths, each on its own line separated by carriage returns, can be specified in this field. |
| Doc Root | Root of the servlet's file system on disk. Required. |
| Initial Arguments | Any initial arguments required for this servlet. These are made available to the servlet's init() method. Multiple initial arguments, each on its own line separated by carriage returns, can be specified in this field. |
| Description | Description to be associated with the servlet being added. Optional. |

## WAR tab

This tab section represents the only configurable parameters for WAR files. All of the attributes available for modification here are read from and saved to the Enhydra Multiserver configuration file, /usr/local/<*enhydra_root*>/multiserver.conf.

**Figure 3.15** WAR tab section



The following table provides information about modifying WARs.

**Table 3.19** Modifying WARs

| Field | Description |
|---|---|
| Doc Root | Path to the root directory of the Web archive, after it is decompressed. |
| Description | Plain English description of this archive. |
| Default Session Timeout | Period of time, in minutes, for which the session may remain idle before timing out. |
| Is War Expanded | Whether the WAR file has been extracted. |
| Is War Validated | Whether the WAR file has been validated. |
| WorkDir Persistent | Select if you want to save the work directory after the termination of the current Multiserver session. |

## Finalizing application modifications

1. After making the desired modifications to your application, select Save. An alert appears, confirming that the changes have been saved to the corresponding configuration file.
2. If you made changes that affected the Enhydra Multiserver configuration file, another alert appears to confirm this modification.
3. Click OK to return to Status information for the selected application. Any modifications that have been made to the application-specific configuration files take effect upon starting the application.
4. To save changes that affect the Enhydra Multiserver configuration file, click the Save State button in the control frame.

# Monitoring traffic to the application

The debugging utility is used as a window into the operation of a running application to aid in debugging or maintaining an application. Enabling this feature displays a separate debugging control panel which is used to control debugging-specific functions. Debugging information is then displayed in the content frame area of the display.

If the selected application is not currently running, or has no active connections, the Debug button is disabled.

1. Select an application from the Applications window.
2. Click the Debug button in the console tools section. The debugging control panel, containing the debugging controls and an event viewing space, appears.

**Figure 3.16** Debugging control panel


The name of the application is appended with the designation `D` in the Applications window to designate that this application is currently being debugged.

You can now view events that are being handled by the application being debugged in the event viewing space.

Events appear here only if the application is actively being used. If no events are displayed, you can generate some activity in the selected application, which is reflected in the debugging control panel.

**Table 3.20** Debug control panel buttons

| Button | Description |
|---|---|
| Pause | Pauses the debugging function, and stops the accumulation and scrolling of events. This is useful while debugging an application with a fast-scrolling event list that needs to be paused for perusal. Disabled if when the Pause function is already on. |
| Resume | Resumes the debugging function, and restarts the accumulation and scrolling of events. Disabled until the Pause button is activated. |
| Clear | Clears all of the current accumulated events listed in the event viewing space. |
| Finish | Halts the debugging function, and closes the debugging controls window. The application being debugged loses its `D` designation in the Applications window. |

## The active event list

Once an application or servlet is in debugging mode, any event handled by this application appears in the debugging controls window's event viewing space. Each line has a format similar to the following:

```
200 host1 GET Welcome.po
```

The first column designates the status code returned to the browser as a result of this request, in this case, `200`. These status codes are color-coded to designate different types of events that might need attention.

The second column refers to the name of the remote host that originated this request.

The third column refers to the type of event. Here, we have a `GET` request. The other option is a `POST` request. In this column, the name of the request type (i.e., the word `GET`) is a clickable link. Clicking on this link displays the Debug screen in the Admin Console's content frame. "Debugging event details" explains the information in this screen.

The fourth column refers to the resource being requested by the browser at the remote host, in this case, `Welcome.po`.

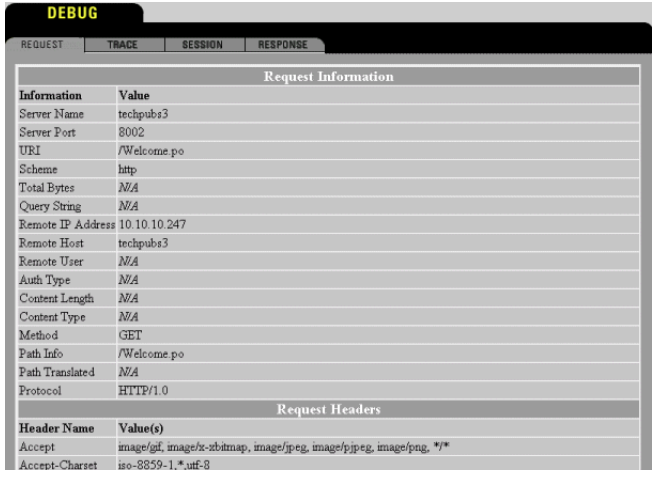
## Debugging event details

Clicking the name of the type of event (`GET` or `POST`) in the third column of the debugging active event list, displays detailed information about that event. This information appears in the content frame of the console, split between three tabs: Request, Trace, and Response.

### Request tab

This tab represents information about the request sent from the remote browser to the Enhydra Multiserver. It contains areas for general information, headers, cookies, and parameters associated with this request.

**Figure 3.17** Debug Request tab



The following table provides information about Request tab fields.

**Table 3.21** Debugging detail: Request tab fields

| Field | Content |
|---|---|
| Server Name | Name of the server hosting the Enhydra Multiserver. |
| Server Port | Port number on the server used to access the Enhydra Multiserver. |
| URI | URI used to request a resource for this event. |
| Scheme | Scheme used to make the request. |
| Total Bytes | Number of bytes (if applicable) of this request. |
| Query String | Query string (if any) of this request. |
| Remote IP Address | IP address of the remote client making this request. |
| Remote Host | Host name of the remote client making this request. |
| Remote User | Remote user name (if available) making this request. |
| Auth Type | Authentication type (if applicable) used for this request. |
| Content Length | Content length (if applicable) of this request. |
| Content Type | Content type (if available) of this request. |
| Method | Method (get or post) applicable to this request. |
| Path Info | Path information for the resource being requested. |
| Path Translated | Translated path information for the resource being requested. |
| Protocol | Protocol and version being used for the communication of this request. |
| Cookie | Name and value of the header cookie associated with this request. |
| Accept-Charset | Acceptable character set specified in the header of this request. |
| Pragma | Pragma header for this request. |
| Accept | Accept header values for this request. |
| Referer | Referer URL header for this request. |
| Accept-Language | Header setting the acceptable language parameter. |
| User-Agent | Browser originating this request. |
| Host | Server host name and port number. |
| Connection | Connection type header setting. |
| Accept-Encoding | Acceptable encoding types for this request. |
| Cookie Name | Cookie name (if applicable) for this request, and its value. |
| Parameter Name | Parameters (if any) for this request. |

## Trace tab

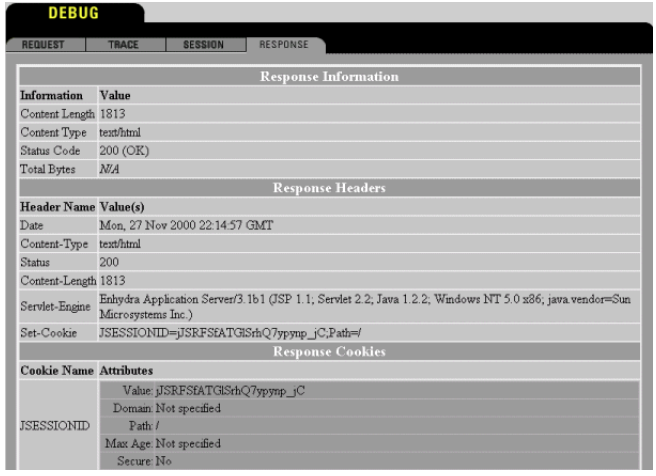
This tab represents a trace listing of calls made to the servlet API by the application as a result of this event. The last item on this list is the amount of elapsed time for the application to handle this request. The diagram below is an example of a trace tab.

**Figure 3.18** Debug Trace tab


## Response tab

This tab represents information about the response sent back to the remote browser from the Enhydra Multiserver as a result of this request. It contains areas for general information, headers, cookies, and the actual response data. This last item is of note as the actual HTML source sent back to the browser will be displayed for HTML responses.

**Figure 3.19** Debug Response tab



The following table provides information about Response tab fields.

**Table 3.22** Debugging detail: Response tab fields

| Field | Content |
|---|---|
| `Content Length` | Content length (if applicable) of this response. |
| `Content Type` | Content type (if available) of this response. |
| `Status Code` | Status code being returned to the browser with this response. |
| `Total Bytes` | Number of bytes (if applicable) of this response. |
| `Expires` | Expiration time for the response data if cached. |
| `Cache-Control` | Cache control setting. |
| `Cookie Name` | Name of the cookie (if any) for this response. |
| `Response Data` | Data being returned as part of this response. |

# Saving the current state of the console

The Save State button lets you save the current state of the Enhydra Multiserver to the Enhydra Multiserver configuration file `/usr/local/<enhydra_root>/multiserver.conf`, and is read the next time the server is started.

The Save State button starts out in a disabled state. However, the first change to the state of the server causes the button to become enabled. The following operations cause a change to the state of the server, and thus cause the Save State button to become active.

- Starting or stopping an application via the Start or Stop buttons
- Adding a new application via the Add button
- Deleting an existing application via the Delete button
- Modifying any of the fields for a servlet or WAR file, using the Modify button
- Modifying the Description field for an application, using the Modify button
- Adding a new connection to an application using the Add Connection button
- Removing an existing connection from an application using the Remove Connection button
- Enabling a disabled connection of an application using the Enable Connection button
- Disabling an enabled connection of an application using the Disable Connection button
- Adding a new filter to an existing connection of an application using the Modify Filters button
- Removing an existing filter from a connection of an application using the Modify Filters button.

To save the current state of the Enhydra Multiserver:

1 Perform one or more of the operations listed above, and arrive at the server state that you wish to save.

2 Select the Save State button from the console tools section.

The content frame refreshes, presenting you with a confirmation alert.

3 Click OK to proceed with the save, or Cancel to abort. If you select OK, the window refreshes and displays notification of a successful save operation.

4 Click OK to return to the Admin Console.

# Creating a WAR file

This section describes how to set up the directory structure of a WAR file to deploy Java servlets, JSPs, and static content. Under the Servlet 2.2 API, JSPs and servlets are assembled into a directory structure referred to as a Web application archive, or WAR. When the WAR is finished, you can compress it into a file with a `.war` extension. Once set up, it can be moved from server to server without further configuration. A good way to understand how to construct a WAR is to look at a simple example.

### A simple WAR example

Suppose you want to deploy a JSP called `Hello.jsp` and a servlet called `Hello.java`. Set up the following directory structure:

```
/tmp/webApp
   myJspDir
      Hello.jsp
   WEB-INF
      classes
         Hello.class
      web.xml
```

The directory structure begins with a document root directory that has an arbitrary name, in this case, `webApp`. The JSP page can be placed either in the document root directory or in any of its subdirectories. In this example, it is placed in an arbitrary subdirectory named `myJspDir`. `WEB-INF`, the one required subdirectory, contains the configuration file `web.xml` and a directory called `classes` which in turn contains the compiled servlet classes. If you are only using JSPs, you do not need a `classes` subdirectory. The Multiserver adds the `classes` directory to its `CLASSPATH`, so the server automatically finds servlets placed in that directory.

The last required file, `web.xml`, contains deployment information like name mappings, parameters, and default file mappings. The `web.xml` file in this simple example contains the following essentially empty file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
   "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
```

```
<web-app>
   <!-- configuration options would go here -->
</web-app>
```

Once you register your Web application with the Multiserver, you access the above pages with the following URLs:

- `http://<your_host>/myPrefix/myJspDir/Hello.jsp`
- `http://<your_host>/myPrefix/servlet/Hello`

The `http://<your_host>` section of the URL represents your host machine. When you configure the Multiserver to run the servlet, you tell it that the path to the document root is `/tmp/webApp`, and the URL prefix is some arbitrary string like `/myPrefix`. Therefore, every request with the prefix `myPrefix` is forwarded to the Multiserver which in turn runs your application.

The remainder of the URL for the JSP page corresponds to the directory structure. You can put HTML files in the same directory and request them in a similar manner. Calls to the servlet require the reserved word "servlet" in the URL. When the servlet server sees it, it knows to look for the corresponding class in the `classes` subdirectory of the `WEB-INF` directory.

### For more information

For a somewhat more complex example, see the `WarExample` application that ships with Enhydra. Beyond that, Sun's Servlet 2.2 specification provides more information about configuring a WAR, containing both instructions and examples. You can download it from http://java.sun.com/products/servlet/download.html.

### A more complex web.xml file

As mentioned above, the `web.xml` file describes a number of configuration parameters. The `web.xml` file in the foregoing example is empty. It contains no configuration parameters. The following `web.xml` file shows how to perform some common configuration tasks, described by comments in the code.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
   "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
   <!-- The next two sections define name/class mappings. -->
   <servlet>
      <servlet-name>BasicHello</servlet-name>
      <servlet-class>Hello</servlet-class>
      <init-param>
         <param-name>SomeParameter</param-name>
         <param-value>Parameterized hello from the web.xml file.</param-value>
      </init-param>
   </servlet>
   <servlet>
      <servlet-name>PackagedHello</servlet-name>
      <servlet-class>a.b.c.HelloPackage</servlet-class>
   </servlet>
```

```
    <!-- The next two sections define name/url mappings. -->
    <servlet-mapping>
       <servlet-name>BasicHello</servlet-name>
       <url-pattern>/Hello</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
       <servlet-name>PackagedHello</servlet-name>
       <url-pattern>/Howdy/*</url-pattern>
    </servlet-mapping>
    <!-- The next two sections define welcome files and the error page. -->
    <welcome-file-list>
       <welcome-file>index.jsp</welcome-file>
       <welcome-file>index.html</welcome-file>
       <welcome-file>myFunkyIndex.htm</welcome-file>
    </welcome-file-list>
    <error-page>
       <error-code>404</error-code>
       <location>/aDirectory/myFavoriteErrorPage.html</location>
    </error-page>
    <!-- This application wide parameter mapping is used to map the / to a page. This
    parameter is used by the index.jsp page in this example.  -->
    <context-param>
       <param-name>welcomeServlet</param-name>
       <param-value>/Hello</param-value>
    </context-param>
</web-app>
```

### Mapping servlets to URLs

The first lines in the `web.xml` file map servlet classes to arbitrary names. In the above example, the `Hello` servlet class is mapped to the name `BasicHello`. In another example, the mapped class is in a package. The class file for the `HelloPackage` servlet is located at `/someDir/webApps/webApp/WEB-INF/classes/a/b/c/HelloPackage.class`. Once the servlet classes are mapped to names, the names can link the classes with arbitrary URL mappings. For example, the `HelloPackage` servlet is mapped to `/Howdy/*`, so the following URL brings up the servlet:

`http://localhost:8010/myPrefix/Howdy/aBigHello`

Notice there is no use of the reserved word “servlet” in the URL. Frequently, every servlet in a Web application is mapped to a corresponding URL to avoid using the word “servlet.” The example uses the wildcard * (asterisk), but you can map to specific names. The pattern-matching rules are described in more detail in the servlet specification. For security reasons, a single * (asterisk) is the only supported wildcard.

JSPs can be mapped to URLs in the same way. Just use the `jsp-file` tag instead of the `servlet-class` tag when you define the servlet.

### Mapping default and error pages

The `welcome-file-list` and `error-page` sections of the `web.xml` file defines default pages and the error page. The `welcome-file-list` section lists the default pages. There are three entries: `index.jsp`, `index.html`, and `myFunkyIndex.htm`. If a URL is called without a page specified, these are the default pages. For example, assume a WAR has the following configuration:

A `/tmp/webApp/aDirectory` containing a file `index.html`.

The root of the Web application is mapped to `/tmp/webApp`. The URL prefix is mapped to `"/myPrefix"`.

In this case, a call to the following URL would return the contents of `index.html`:

```
http://localhost:8010/myPrefix/aDirectory
```

If there was another subdirectory `webApp/anotherDir` with the file `myFunkyIndex.htm`, a call to http://localhost:8010/myPrefix/anotherDirectory returns the contents of `myFunkyIndex.htm`.

The `error-page` section defines the default error page. In this case, the 404 File Not Found error is mapped to a page called `myFavoriteErrorPage.html` in the directory `aDirectory`. Nonexistent URLs will map to this page.

### Defining parameters used by the application

Virtually every application uses parameters that are set in a configuration file. In the Servlet 2.2 model, the parameters used by an application are included in the `web.xml` file. The above example `web.xml` file includes one parameter set for a servlet and one for the entire Web application. The `Hello` servlet includes a parameter defined in the block beginning with the `init-param` tag. The servlet then gets that value with the following call:

```
getInitParameter("SomeParameter")
```

The application-wide parameter is defined in the block beginning with the tag `context-param`. This value can be accessed by any servlet or JSP with the following method call:

```
getServletContext().getInitParameter("welcomeServlet")
```

Servlets talk to each other through the servlet context in a similar manner. For example, if you have a factory class you instantiate on startup and place in the servlet context, then all other servlets can access that factory object through the servlet context. A full discussion of JNDI, security, and servlet/EJB communication is beyond the scope of this document, but they are an integral part of Servlet 2.2.

### Using the load-on-startup tag

While it is not used in this example, the `load-on-startup` tag is very useful. Many large Web applications start at least one servlet that manages services for the rest of the application. That central servlet might start a database connection pool manager, verify that an email server is working, test a credit card verification service, or instantiate objects used by the rest of the servlets. This tag takes a positive integer as an argument. The number determines what order the servlets are started in. For more information, see the `web.xml` DTD in the servlet specification. The DTD includes descriptions of other tags not used here, including tags that control the security context and references to Enterprise Java Beans.

### Mapping the slash to a servlet welcome page

Mapping the slash (`/`) to a servlet is an apparently simple task that requires a somewhat complicated solution. It is common to map the opening page of an application to a servlet, in which case a request to the URL `http://foo.com/myPrefix` is

forwarded to the servlet mapped to `http://foo.com/myPrefix/Hello`. Note that your application prefix can just be `/` if you want `http://foo.com` to go to `http://foo.com/Hello`.

The most intuitive answer is to simply define a URL mapping to the URL `/`. However, if you do this, the mapping acts like mapping `/*` and every URL not defined in the `web.xml` file to the default page. Files like `foo.html` are no longer served. This occurs because mapping `/` takes over the default servlet. This behavior is defined in the servlet specification. Because `*.html`, `*.wml`, and other static files are handled by the default servlet, the returned request looks like the page you mapped to the slash.

One good solution to this problem is to use a JSP page in the document root that forwards the request to your welcome servlet. The server receives the request for the slash and looks for the `welcome-file` mappings. It finds an `index.jsp` file and calls it. That JSP file then forwards the request to your `welcome` servlet. The advantage to this approach is that it leaves the default servlet that comes with the server in place. It also takes advantage of the `welcome-file` approach defined in the Servlet 2.2 specification. The `index.jsp` file can be copied to any subdirectory and it forwards the request to the default servlet defined in the `web.xml` file.

An important point to note with this approach is the possible security risk of the default servlet showing directory listings. If a directory listing will compromise your site's security, put an `index.jsp` or `index.html` in every directory. For example, assume you have a `webApp/media` directory containing your site's images. Without a welcome file, a call to `http://foo.com/myPrefix/media` returns a directory listing of all the images. If you do not want a user to see that listing, put an `index.jsp` or `index.html` file in the `media` subdirectory.

### The default web.xml file

The Multiserver has a `web.xml` file that defines default mappings, located in `enhydra.jar`. You can extract the file from the JAR file using a ZIP utility or other extraction utility. Reading this `web.xml` file is instructive. For example, there is a mapping for `"/servlet/*"` to the `ServletInvoker` servlet. The default `welcome` mappings are listed at the bottom of the file.

### Adding MIME types for WAP/WML

While the default `web.xml` file defines hundreds of MIME mappings, it does not define the MIME mappings used by the WAP protocol, the protocol used to transmit content to the micro-browsers in cellular telephones. If your WAR contains `*.wml` and `*.wbmp` files, you need to add the wireless MIME types to your `web.xml` file.

### Adding other JAR files to a Web application

So far, this section has only described how to add servlets or other classes to the `classes` subdirectory in the `WEB-INF` directory. Any JAR files your application uses are put in the `lib` subdirectory of the `WEB-INF` directory. The name `lib` is a reserved word. The Multiserver adds the `WEB-INF/classes` directory and every JAR file in `WEB-INF/lib` to its `CLASSPATH`. Servlets do not have to be in the `classes` subdirectory. They can be anywhere in the Multiserver's `CLASSPATH` if they are mapped in the default `web.xml` file.

### The images directory

Frequently, a Web application has an `images` or `media` subdirectory that contains all the image files for the application. During development you want the images to be part of your WAR, but at deployment time you want the Web server to serve them. Although images and static files can be served from a WAR, the Web server will serve them more efficiently because Web servers are written in native code and are optimized to serve static content.

The following steps describe how to serve static content from the WAR during development and from the Web server's `htdocs` directory after deployment.

1. During development, put your images in an `images` subdirectory of the top-level Web application directory.
2. Make links in your HTML relative to `/` (Web server root). For example, the HTML your servlets return contains the following:

   ```
   <img src="/images/Enhydra.gif">
   ```

3. During development, use a URL prefix of `/` (Web server root) so all requests go to the Web application.
4. When you deploy, move all static content from your WAR to the Web server's document directory.
5. Configure the Web server and the Multiserver to use an URL prefix like `myPrefix` for your WAR. Images are then served by the Web server, and URLs like `/myPrefix/Hello` are forwarded to the application server.

### Enhydra issues

The previous information applies to any servlet or JSP container that is Servlet 2.2-compliant. This section describes issues that are specific to the Enhydra Multiserver servlet runner.

Super-servlet applications are easy to add to a WAR. You can use a super-servlet application, JSPs, and servlets in the same application. Enhydra applications deploy as a single servlet.

# Chapter 4

# Using Enhydra Kelp

This chapter describes how to use Kelp to develop Enhydra applications with XMLC. It assumes a basic understanding of either JBuilder or JDeveloper and the Enhydra Application Framework.

## Introduction

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

*Kelp for JBuilder* is a set of Enhydra tools for Borland JBuilder. For information on JBuilder, visit the Borland Web site at: http://www.borland.com/jbuilder/.

*Kelp for JDeveloper* is a set of Enhydra tools for Oracle JDeveloper. For information on JDeveloper, visit the Oracle JDeveloper Web site at: http://www.oracle.com/tools/jdeveloper/.

### Kelp features

Kelp provides the following Enhydra tools:

- **Enhydra Application wizards**
  The Application wizards generate Web or wireless applications using either the Servlet API or the Enhydra super-servlet programming model.
- **XMLC Compiler wizard**
  The Compiler wizard lets you set XMLC options, select HTML files to compile, and call XMLC from within the IDE.
- **Enhydra Deployment wizard**
  This wizard allows you to copy static content to the document root, set up your project properties to start the Enhydra server when your run the project, process templates, and create deployable archives.
- **Enhydra Import wizard**
  The Import wizard allow you to import an Enhydra project that uses GNU Makefiles in to your IDE.

- **Enhydra XMLC property pages**
  The XMLC property pages give you full control over how XMLC builds document object model (DOM) classes from your HTML files. You can open the property page directly from the project navigator's right-click menu.
- **Enhydra Template property pages**
  The Enhydra 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 Enhydra configuration processes whenever you make or rebuild your JBuilder project. This feature lets you quickly ensure your files are updated without having to run a wizard.
- **Kelp sample projects**
  These projects demonstrate techniques for creating dynamic Web pages with XMLC for both wireless and Web applications.

# Using the wizards

Kelp provides four wizards that help you develop Enhydra applications from within your IDE:

- Enhydra Application wizard
- XMLC Compiler wizard
- Enhydra Deployment wizard
- Enhydra Import wizard

Use the Enhydra Application wizard to create new Enhydra projects. To create a new Enhydra project, select File | New to open the Object Gallery. Click the Enhydra tab. Select the appropriate wizard for your application. Once you create a project, you can use the XMLC Compiler and Enhydra Deployment wizards to work with it.

The XMLC Compiler, Enhydra Deployment, and Enyhdra Import wizards are located in the Wizards menu. These wizards let you invoke Enhydra-specific parts of the `make` system.

**Note** You can invoke these same processes directly within the Project Builder without opening either wizard when you make or rebuild a project.

In both the XMLC Compiler and Enhydra Deployment wizards, you use a tabbed dialog box to select files, set options, and view the build process. The settings on the Options tab are the same as those found on the associated project property pages.

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

## Using the Enhydra Application wizard

The Enhydra Application wizard generates a set of skeleton files for Enhydra applications. You can generate two kinds of applications: Web applications, which use the Java Servlet API, or Enhydra super-servlet applications, which use the

Enhydra PO/BO/DO model. For more information on the Enhydra Application wizard, see Chapter 2, "Using the Application Wizard to create Enhydra applications."

To create a new application using the Application wizard:

1. Select File | New.
2. Click the Enhydra tab.
3. Select either Web Application or Enhydra SuperServlet application.
4. Click OK.

This will start the Application wizard for the type of Enhydra application you selected.

1. Select your application's client type in the Client Type drop-down menu. The possible choices are:
   - HTML
   - WML
   - cHTML
   - XHTML
2. Enter the project directory name.
3. Enter the project package name.
4. Enter the root path for your applications. Click Set to navigate to your root directory.
5. Click Next.
6. If your application requires copyright text, select either Choose File if you have a text file that contains your copyright text, or Enter Copyright Text to enter the copyright text in the edit box below the radio buttons. If your application does not require copyright text, select No Copyright.
7. Click Next.
8. On the Supplemental Files dialog are two check boxes: Create Makefiles and Create Shell Scripts.

   Create Makefiles creates GNU Makefiles so your project can be built from the command line using `make`. Create Shell Scripts will create Bourne shell scripts to launch your application from the command line.
9. Click Finish to complete the Application wizard.

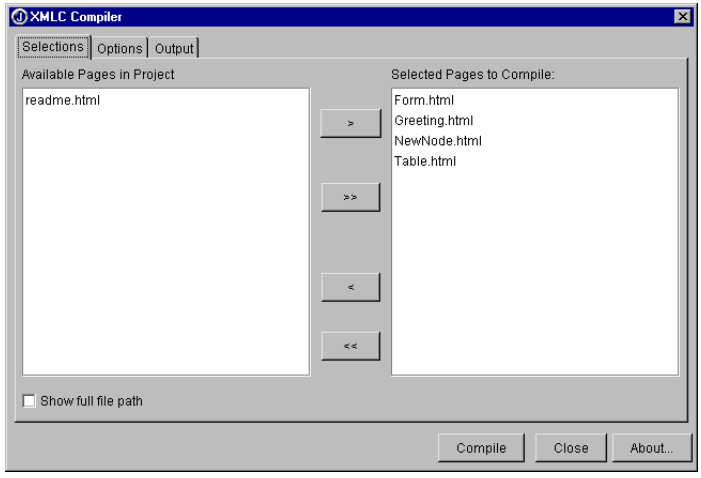
## Using the XMLC Compiler wizard

The XMLC Compiler wizard compiles HTML, WML, cHTML, and XHTML files into DOM classes. To open the wizard, select Wizards | XMLC Compiler. The wizard is only available when a project is open.

**Note** If you do not see the XMLC Compiler option in the Wizard menu, make sure `kelp2.0.jar`, `toolbox.jar`, and `enhydra.jar` are listed in the IDE `CLASSPATH`.

The wizard is organized into dialog box with three-tabs: Selections, Options, and Output. The Selections tab, shown in Figure 4.1, displays all HTML files in the currently selected project. 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 and remove all files from the selection list. Check the Show Full File Path checkbox to display the full path of the files.

**Figure 4.1** Selections tab of the XMLC Compiler wizard

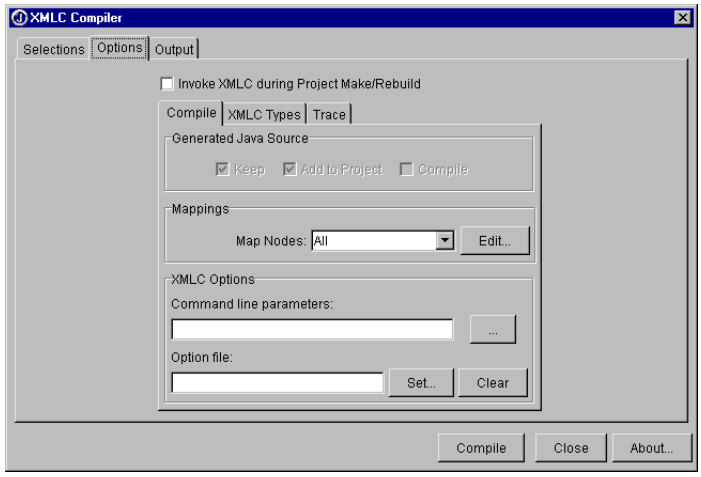



The Options tab displays additional tabs for compile, XMLC types, and trace options.

- The Compile options let you customize the generated DOM classes.
- The XMLC Types tab allows you to add new file extensions to be associated with XMLC, and display on the Selections tab.
- The Trace options let you display detailed information during the compile process without affecting the generated classes.

The Options tab also has an Invoke XMLC during Project Make/Rebuild check box. Check this box to call XMLC without opening the XMLC Compiler wizard. This calls XMLC when you build or make a project node that contains a selected HTML file.

**Figure 4.2** Options tab of the XMLC Compiler wizard




The Output tab shows the results of your XMLC compile based on the files you selected on the Selections tab. For information on the Output tab, see "Setting output options" on page 56.

### Using mapping tables for generated class names

One common make option is to use a mapping table to customize generated class names. By default the XMLC Compiler wizard 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 HTML directory to the presentation package name.

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

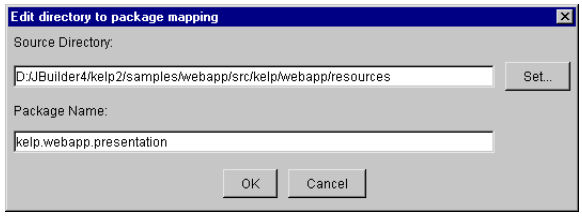
To create mappings:

1. Click Edit in the Mappings section of the Make tab to open the Project Map editor that lets you associate source directories with package names.
2. 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 4.3 shows how to set up the compiler to use the `org.enhydra.kelp.sample.presentation` package name when compiling HTML pages stored in `/usr/local/jbuilder/kelp2/sample/org/enhydra/kelp/sample/resources`.

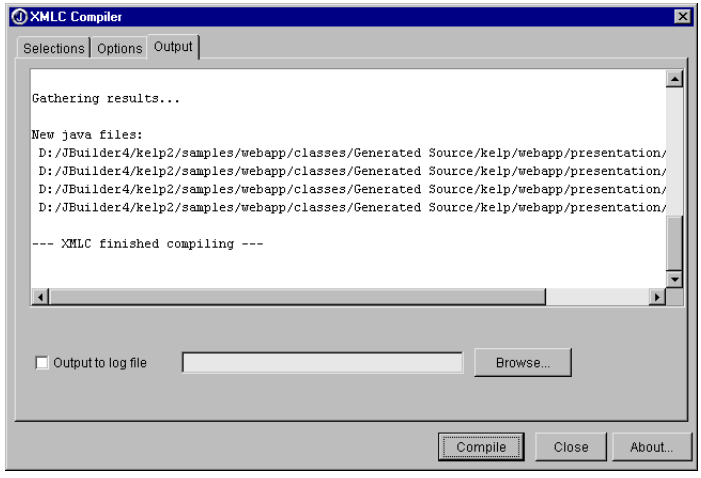
**Figure 4.3** Mapping a source directory to a package




### Setting output options

The Output tab (Figure 4.4) is automatically selected when you click Compile in the XMLC Compiler wizard. 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 checking Output To Log File and entering a file name.

**Figure 4.4** Output tab of the XMLC Compiler wizard




The Output page displays the files that are created during the compilation process. At the start of the compilation process, the wizard erases any files it created during a previous compile. If a HTML file contains a new error, the associated Java and class files will be erased and not regenerated.

## Using the Deployment wizard

The Deployment wizard lets you quickly configure projects for your current environment and directory structure. The wizard can do four tasks. The first task is to generate configuration files and deployment descriptors from templates. The second is to setup your project so you can launch Enhydra with your application. The

third is to copy static content to your document root. The fourth is to create a deployable archive, based on the type of application.

### Selections tab

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.

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

### Options tab

The Options tab has four subtabs: Paths, Content Types, General, and Template.

Check Deploy During Build to deploy your application on each project rebuild. The Show Messages box is enabled if Check Deploy During Build is checked. If Show Messages is checked, you'll any deployment messages in the Output tab.

The Paths tab lets you change the paths for your project.

Deploy Root controls setting for configuration your project's path, document root, and archive file. Click the ellipses button to navigate to a folder.

Bootstrap File is used for launching Enhydra from your IDE. Click the ellipses button to navigate to the project's bootstrap file.

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.

In the Content section, Resource Path is a folder that contains any static content files for your project. These static files will be copied to your document root using relative paths. The Content Types sub-tab controls what static files the wizard recognizes. Click the ellipses button to navigate to the folder containing the static resources for your project.

The Content Types subtab shows the extensions of static resource files that will be copied from the Resource Path 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.

The General subtab has two sections: Presentation API, where you select what type of project archive file should be created, and Project Run Configuration, where you adjust the settings for starting Enhydra with the project.

If your project uses the Java servlet API, select Servlet in Presentation API. If you are using the Enhydra super-servlet API, select Presentation Object. Enhydra super-servlet archives contain the classes in your output directory. Servlet archives are Web Application Resource (WAR) files that contain the output classes, static content files, and a deployment descriptor.

In Project Run Configuration, check Configure Project For Running Enhydra if you want to launch the Multiserver with your project. If this box is checked, the two sub-options are enabled. Check Project Run Parameters to have the wizard automatically set the parameters of the Multiserver. Create StartEnhydra.java should only be checked when you use JDeveloper with Kelp. `StartEnhydra.java` is a helper class that starts Enhydra.

The Template tab lets you adjust the template settings for your project. Configuration templates have the extension `.conf.in`. Deployment descriptor templates have the extension `xml.in`. Templates are text files with placeholders that are replaced with system specific information by the wizard. One of the default placeholders is `@CLASSES@` and it 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 wizard. 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 wizard will substitute 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 find values of the form `@*_PATH@` are handled differently than typical 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`.

The Default button doesn't set the table back to static values. It searches through your `.conf.in` templates and imports any replacements it finds.

## Using the Enhydra Import wizard

The Enhydra import wizard lets you quickly import source files from an existing Enhydra GNU Makefile project into a JBuilder or JDeveloper project file. As an

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

**Note** The JBuilder project file that comes with Enhydra 3.0.x is not compatible with Kelp 2. We recommend that you follow these steps to create a new project for use with Kelp 2.

Here are the steps for running the DiscRack example using Kelp with JBuilder 4. They are broken down into three sections:

- Importing DiscRack
- Building DiscRack
- Running DiscRack

If you have the application running outside of JBuilder, you should also be able to skip"Running DiscRack."

### Importing DiscRack

Follow these steps to import DiscRack in to JBuilder.

1. First build DiscRack as described in the README file to generate the data package source code.
2. Create a new JBuilder project file.
3. Select Wizards | Enhydra Import.
4. Set the project directory to `<enhydra_root>/examples/DiscRack`.
5. Click Next to navigate through the wizard, accepting all default settings.
6. Click Finish to import the project.

### Building DiscRack

Follow these steps to build DiscRack.

1. Open the XMLC Compiler wizard.
2. Click Compile to generate the DOM Java source files using XMLC.
3. When XMLC is finished compiling, click Close.
4. Select Project | Make to compile the Java source files.
5. Select Wizards | Enhydra Deployment.
6. Click Deploy to copy static content files, process configuration templates, and create a deployable archive.

### Running DiscRack

Follow these steps to run DiscRack.

1. Select Project | Project Properties.
2. Add the InstantDB library to the required library list:
   1. Select the Paths tab.
   2. Select the Required Libraries subtab.

3. Click Add.
4. Click New.
5. Name the library InstantDB.
6. Click Add.
7. Navigate to the location of `idb.jar`, select it, and click OK.
8. Click OK until the Project Properties dialog is closed.

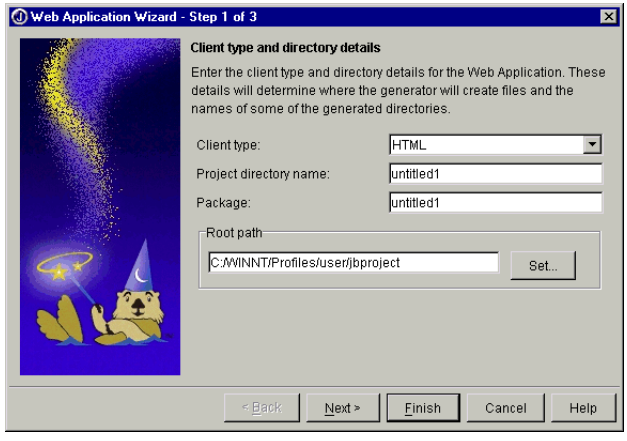
3. Select Run | Run Project.
4. Open a browser to `http://localhost:5555` to view the application.

## Using the Enhydra Application wizard

The Enhydra Application wizard speeds up the development of Enhydra applications by creating skeleton project files. Figure 4.5 shows the Enhydra Application wizard.

**Important** Running the Enhydra Application wizard while you are using a project that already contains Enhydra files may cause unexpected results. Before you run the Enhydra Application wizard, you should create a new project for the generated files.

**Figure 4.5** Enhydra Application wizard




Once you have a opened a new project, select File | New to open the Object Gallery. The gallery provides options for quickly generating many common classes. Click the Enhydra tab, select the Enhydra Application icon, and click OK to open the wizard.

For more information on the Enhydra Application wizard, see Chapter 2, "Using the Application Wizard to create Enhydra applications."

After you have set the options you want, click OK to generate your Enhydra application. After generation is complete, you should 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. The steps will vary, depending on what version of JBuilder you are using.

# Using the Property pages

*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.

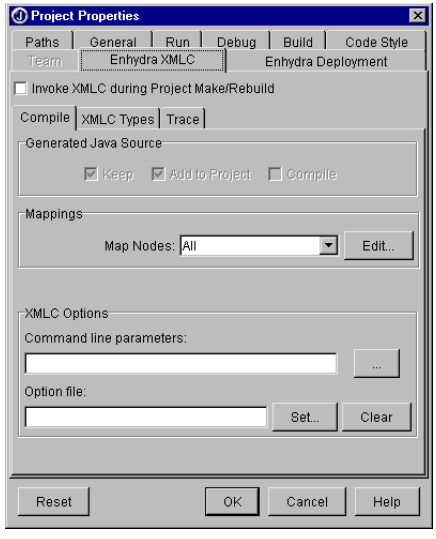
- To open the Project Properties dialog box, select Project | Project Properties.
- To open a Node property page, right-click a file in your project and select Properties from the right-click menu.

JBuilder opens a property page specific to the file you've 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 Enhydra Template node property page, right-click a 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. In JDeveloper, right-click the HTML file and select XMLC Properties.

## XMLC project property page

You can view the XMLC project properties page by opening the Project | Project Properties dialog box and selecting the Enhydra XMLC tab (Figure 4.6). This page shows the same make and output options that you can set on the Options tab of the XMLC Compiler wizard.

**Figure 4.6** 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."

Click Edit to open the XMLC Parameter Editor. The XMLC Parameter Editor is used to better organize complex XMLC command line options.

Within the Mappings section, you can define a mapping table and set which nodes 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 Kelp sample project uses a mapping table to map all the HTML files in a resources directory into the `org.enhydra.kelp.sample.presentation` package.

To view or edit the package name mapping:

1. Open the XMLC Compiler wizard.
2. Select the Options tab.
3. 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
```

Click 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.

## Enhydra Deployment property page

Select Project | Project Properties and click the Enhydra Deployment tab to view the Enhydra Deployment properties.

The Enhydra Deployment property page contains the same settings as the Options tab of the Enhydra Deployment wizard. For more information, see "Using the Deployment wizard" on page 56.

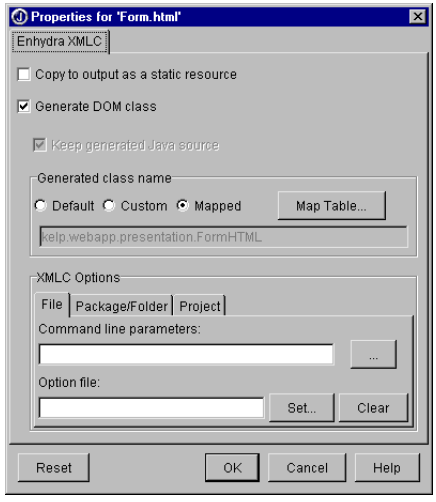
## XMLC node property page

You can open the XMLC node property page by right-clicking an HTML, cHTML, WML, or XHTML file in the Project pane and selecting Properties. This opens the

Properties page for the selected file. This page has four options, as shown in Figure 4.7:

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

**Figure 4.7** 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 Enhydra 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 Custom option to enter your own class name. When using custom class naming, be 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 property page" on page 61 and "Using the XMLC Compiler wizard" on page 53.

### 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 4.7 on page 63).

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."

**Note** The following XMLC options are not supported as command-line parameters when using Kelp:

- `-class`
- `-dump`
- `-info`
- `-javac`
- `-javacflag`
- `-javacopt`
- `-keep`
- `-methods`
- `-sourceout`
- `-verbose`
- `-version`

### Enhydra Template node property page

The Enhydra Template node property page contains the same settings that appear on the Template tab of the Enhydra Deployment project properties page. Right-click the `.in` file and select Properties to open the Enhydra Template node property page for a `.in` file in your project.

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

## Setting project properties

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

## Paths page

Configuring your project to use the correct paths for file output, project files, and libraries will simplify the development of your Enhydra applications. 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. This is also where Multiserver looks for images that have relative references to the presentation objects. For example, the Enhydra 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 display properly, the output directory must be set to a `classes` subdirectory under the source directory.

The Enhydra Deployment wizard also uses this setting when creating deployable archives

### Source subtab

The *source path* is where the compiler looks for Java files and packages to compile. When compiling 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 select multiple source paths. This can be useful when you are working on several modules from the Enhydra source code. 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 4, be sure that your output path is not a subdirectory of any of your source paths.

**Note** You should remove any source paths that your project does not require.

### Required libraries subtab

Enhydra applications require that `enhydra.jar` be in their `CLASSPATH`. 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 button 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 `enhydra.jar` for the library matches the `enhydra.jar` referenced in the IDE class path. Use the About button in the XMLC Compiler wizard to determine the version of Enhydra that is in the IDE `CLASSPATH`.

## Build page

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 check 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 page

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 the 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 Multiserver. Use the Set button to select the Multiserver startup class, `com.lutris.multiServer.Multiserver`.

### Application parameters option

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

`<source directory>/output/conf/servlet/servlet.conf`

# Kelp sample projects

Kelp includes two sample projects that demonstrates 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 `/usr/local/lutris-enhydra3.5/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 5, "DiscRack sample application," of *Getting Started with Lutris Enhydra*.

## Deploying the Web application

Kelp includes the Enhydra Deployment wizard that you can use to automatically configure the Web Application project. Before running the Enhydra Deployment wizard or the sample, be sure to review the steps provided in the readme.html file that is included with the project.

If you are running under Linux or Solaris and you do not have root privileges, you may need to copy the Kelp sample projects to your home directory before working with it. To do this copy `<IDE_root>/kelp2/samples` to a directory under your home directory.

If port 9000 is available, you can start up Enhydra and view the sample servlets without modifying the configuration templates. If you need to run on another port, you will need to modify the following line in `servlet.conf.in`. The configuration templates are located in the sample project's `/input/conf/servlet` directory. Change the following line to an available port.

```
Connection.p9000.Port = 9000
```

When you modify files within the IDE, be sure to select File | Save All to commit the file to disk before running the Enhydra Deployment wizard. The wizard may ignore changes made since the last save.

If you are using a port other than 9000, the links within the readme.html will fail. Add the port you are using to the URL. To use port 8080, enter `http://localhost:8080/` as your URL. If localhost fails, try using the IP address 127.0.0.1 in place of `localhost`.

The Enhydra Deployment wizard creates two configuration files from the templates. The first file is `servlet.conf`. Along with the port number, it controls the document root and logging file paths. The `servlet.conf.in` template has placeholders for path settings. The path settings are set through Enhydra Deployment wizard.

The Enhydra Deployment wizard creates the web.xml deployment descriptor using the same placeholder replacement mechanism used to create the `servlet.conf` file. The `web.xml` file contains a `testDataPath` parameter to set where the servlets can find text and property files used within the sample. To change this directory, change the `<param-value>` to your directory.

```
<init-param>
   <param-name>
      testDataPath
   </param-name>
```

```
    <param-value>
      @JAVA_DEPLOY_PATH@/data
    </param-value>
  </init-param>
```

**Note** If you modify any templates, be sure to run the Enhydra Deployment wizard to regenerate the configuration files and deployment descriptors. Enhydra will ignore any changes made to the templates that have not been deployed.

The Enhydra Deployment wizard also sets up the sample project so that you can launch the application with the Enhydra server. It does this by setting the project's main run class, or by generating a `StartEnhydra.java` file. In either case, the wizard sets the `servlet.conf` file as a parameter to 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 will normally contain a 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 Lutris Enhydra*.

**Note** If you have moved a sample project after running Enhydra Deployment wizard, you should run the wizard 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 the XMLC Compiler wizard. 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 `org.enhydra.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 for 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 an JDBC data source. For the sake of simplicity, this example uses 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 may 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 based on input values from retrieved from a Web browser. For simplicity, this example uses a property file to store displayed values. Normally values would be stored in a JDBC data source that would be accessed through the business and data layers of an application.

# Debugging Enhydra applications

The only requirement to debug Enhydra applications from JBuilder is that you start Multiserver from your JBuilder project. Although it defaults to using configuration files specific to the sample project, you can pass it the location of any `multiserver.conf` file.

To run an Enhydra application, you will need two configuration files. One is for your application and the other is for Multiserver. The configuration file for Multiserver specifies the location of any application configuration files. A detailed description of Enhydra configuration files is available in Appendix B, "Multiserver Administration Console" of *Getting Started with Lutris Enhydra*.

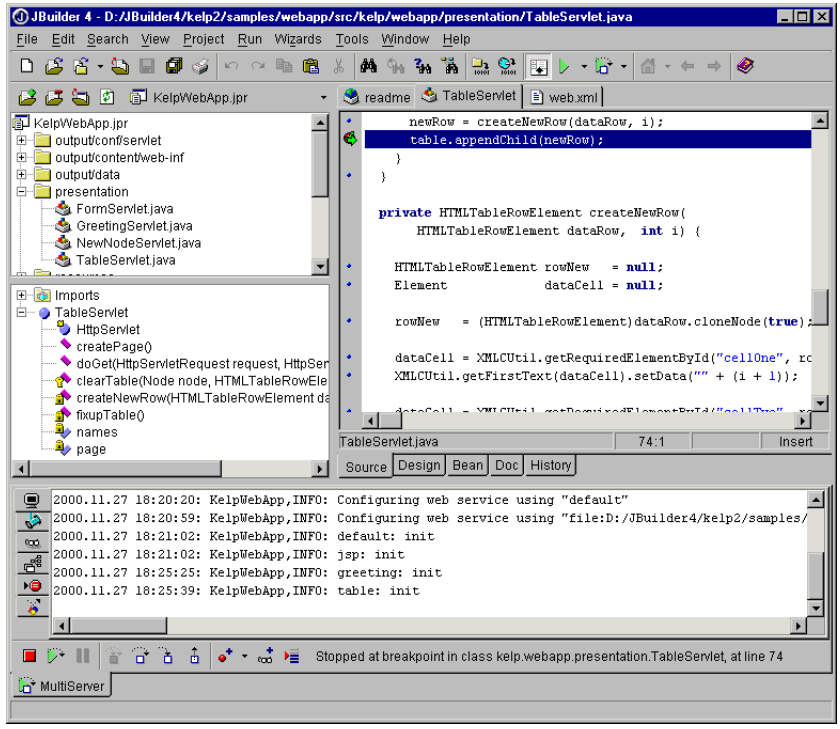
Before you debug an application, make sure you can run the Kelp sample project successfully. This means running the Enhydra XMLC wizard and the Enhydra Deployment wizard. If this is successful, step through the following procedure to see how easy it is to debug an Enhydra application.

1. Make sure the Multiserver is not already running.
2. Open the Kelp sample project.
3. Select `TableServlet.java` in the `kelp.webapp.presentation` package.
4. Locate the `for` loop of the `fixupTable()` 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.
5. If you are using JDeveloper, select `StartEnhydra.java`.
6. From the menu, select Run | Debug.

   This starts the debugger. The debugger opens a command window even if you have selected Execution Log for console I/O.
7. Open the table page in your browser by opening `http://localhost:9000` and clicking Table Page.

This triggers the breakpoint, as shown here:




You can set the value of `i` to `4` to change how many rows are generated in the table. Your browser may time out if you keep the program suspended for too long.

# Working with the Kelp source code

Because Kelp is an open source project, the source code needed to customize or enhance Kelp is available to everyone. To get the source code, download the Enhydra source release.

All the Kelp classes are in `kelp2.0.jar`. Setup installs the JAR in `<JBuilder_root>kelp2/lib` within your JBuilder directory. If you used the Custom setup option to install the open source, you will have three project files you can use to modify the three components that make up the JBuilder wizard.

The `jb-kelp.jpr` file contains all the source files required to build `kelp2.0.jar`. These classes use Swing and JBuilder's Open Tools API to implement the wizard. To use this file, you need to create a library using JBuilder Addin Jars that contains `pthelp3.1.jar`, `jbuilder.jar`, `jbuilder.zip`, and `jdeveloper.zip`. The `pthelp3.1.jar` and `jbuilder.jar` files are available in JBuilder Foundation. `jdeveloper.zip` file is available with Oracle JDeveloper 3.

To work with a single IDE such as JBuilder Foundation, you can leave out the unrelated library files and remove the appropriate packages from the project. The IDE-specific packages are as follows:

| | |
|---|---|
| `org.enhydra.kelp.jbuilder` | Borland JBuilder |
| `org.enhydra.kelp.jdeveloper` | Oracle JDeveloper 3 |

Visit the following newsgroups for information about the developing addins and using the JBuilder OpenTools API:

- `borland.public.jbuilder.opentools`
- `borland.public.jbuilder.thirdpartytools`

# Kelp working group

JBuilder Kelp is part of the Enhydra Kelp family of products. The Kelp working group is dedicated to making Enhydra technologies easier to use through the use of Java IDE add-ons. The working group has a mailing list at: http://www.enhydra.org/project/workingGroups/Kelp.html.

You can use this list to post questions or suggestions for this or other Kelp products. View the mail list archive to get information on recent releases and tips on how to make the most of Kelp. You can also use the Kelp working group to contribute to the Kelp open-source project.

# Chapter 5

# Enhydra XMLC

This chapter describes Enhydra XMLC, the XML compiler. XMLC lets you create Web and XML applications with a clean separation between layout and content, enabling interface designers and programmers to work in parallel on applications without interfering with each other's work.

## Introduction

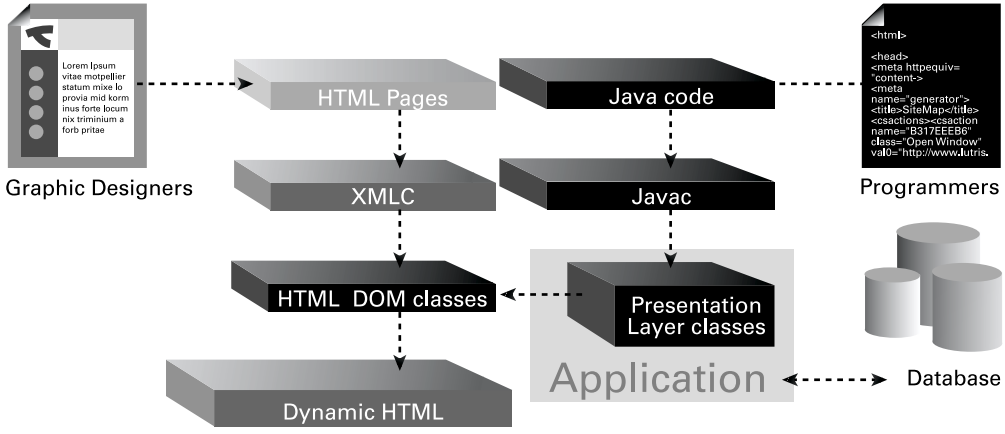
XMLC is a compiler for documents written in eXtensible Markup Language (XML) or Hypertext Markup Language (HTML). XMLC takes as its input an XML or HTML document and creates a Java class that represents that document as a DOM structure. A Java application can use this class to programmatically modify the document and then pass the resulting dynamic content to a Web server, save it to the file system, or use it in some other way.

From a programming perspective, the HTML or XML pages become "resources." Your code can manipulate the document content, without affecting its layout.

One way to think of XMLC is as a *template engine* for XML and HTML documents, because it creates an API from a template document. Unlike other template engines, XMLC is highly extensible. In addition to compiling existing document types, such as Wireless Markup Language (WML), you can extend it to compile other XML document types (for example, MathML).

Figure 5.1 illustrates application development with XMLC:

**Figure 5.1** Application development with XMLC




## Why use XMLC?

You can use XMLC for any application that needs to manipulate XML or HTML files. However, XMLC has a number of advantages when developing Web applications. For example, two of the most popular techniques for generating dynamic Web content are:

- Programs or scripts using the Common Gateway Interface (CGI), which have HTML embedded in the application code.
- Java Server Pages (JSP), which have Java application code embedded in HTML pages.

Both approaches work well for simple, hand-coded HTML pages; however, they do not work as well for complex applications or sophisticated Web pages developed with graphical HTML design tools. This is because they intermingle HTML and application code. In contrast, XMLC enables separation of Java code and HTML pages. This:

- Allows parallel development of HTML layout by graphic designers and programming code by engineers, once you have created the basic storyboard and functional structure of the HTML pages.
- Facilitates application maintenance because changes to page layout do not affect programming logic, and vice-versa. Further, XMLC's auto-recompilation feature lets you change an application's presentation (look and feel) without stopping the application.
- Significantly improves application performance because documents are parsed at compile time rather than at runtime.

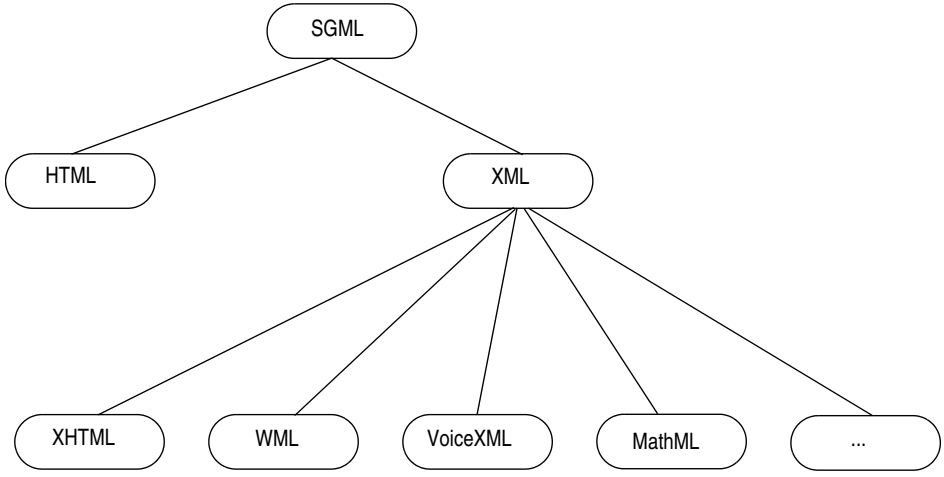
- Helps to ensure application integrity by catching inconsistencies between HTML documents and code. Because XMLC generates objects that reflect document structure, the Java compiler can detect changes that affect the associated code. For example, if the designer removes a tag that the code uses, the compiler catches this problem at compile time, rather than causing a problem at runtime.

## XMLC and markup languages

XMLC ships with support for compiling HTML, XML, cHTML, XHTML, VoiceXML, and WML. For more information on languages for wireless applications, see the *Wireless Application Developer's Guide*.

HTML and XML are both derived from the Standard Generalized Markup Language (SGML), a very general document description language. Figure 5.2 shows the relationships among the markup languages in the SGML family.

**Figure 5.2** SGML family of markup languages




### XML

The eXtensible Markup Language (XML) is designed to describe the properties of data, including its display. XML is actually a meta-language that you use to define your own markup tags. Thus, XML-derived languages such as Wireless Markup Language (WML), VoiceXML, and Math Markup Language (MathML) have their own sets of tags for particular problem domains. By its nature, XML is extensible for a variety of application domains, so there can be any number of XML derivatives.

## Document Object Model

The Document Object Model (DOM) defines the hierarchical object structure of HTML or XML documents. The DOM provides an API for HTML and XML documents.

The DOM is a standard maintained by the World Wide Web Consortium (W3C). You can learn more about the DOM at the W3C Web pages: http://www.w3.org.

Both HTML and XML documents consist of *nodes*, that correspond to tags, delimited by angle brackets (< and >). So, for example, the HTML tag BODY is a node. Each tag, for example, <BODY>, is paired with a closing tag, for example </BODY>. Everything between the opening tag and the closing tag is said to be the tag's contents. In general, a node may have contents and may also contain other tags. For example, in this fragment of HTML,

```
<BODY><P>Some content</P><P>Some more content</P></BODY>
```

The BODY node contains the two P nodes; the first P node has the text content "Some content," the second "Some more content."

Both HTML and XML documents have tree structures, with a single root node with other nodes branching off the root. The DOM data structure for these documents is sometimes referred to as a *DOM tree*. In the example above, the two P nodes are branches of the BODY node.

The following example shows how document elements fit together.

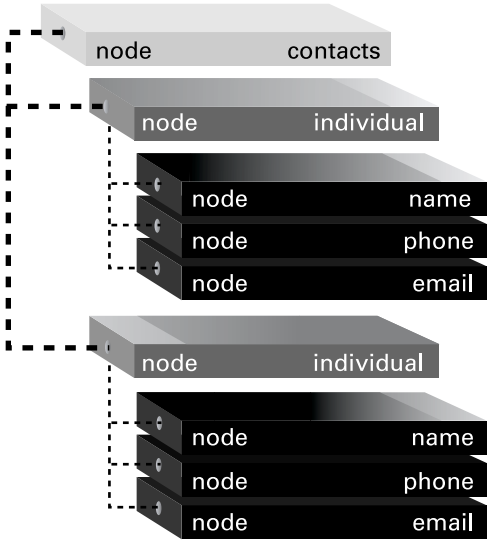
## Example DOM tree

Here is an example of a very simple XML document containing a contact list with names, phone numbers, and email addresses. Figure 5.3 shows the DOM tree for this document.

```
<?xml version="1.0"?>
<CONTACTS>
   <INDIVIDUAL>
      <NAME>Wally Walrus</NAME>
      <PHONE>831-555-1234</PHONE>
      <EMAIL>wally@lutris.com</EMAIL>
   </INDIVIDUAL>
   <INDIVIDUAL>
      <NAME>Ollie Otter</NAME>
      <PHONE>831-555-4321</PHONE>
      <EMAIL>ollie@lutris.com</EMAIL>
   </INDIVIDUAL>
</CONTACTS>
```

As shown in Figure 5.3, the root node for this DOM tree is CONTACTS. It has two children, each called INDIVIDUAL. Each INDIVIDUAL node, in turn, has NAME, PHONE, and EMAIL nodes as children.

**Figure 5.3** Example of a DOM tree for an XML file




XMLC compiles an XML document into an object that:

- Extends the class `com.lutris.xml.xmlc.XMLObject`
- Implements the interface `org.w3.dom.Document`

XMLC compiles an HTML document into an object that:

- Extends the class `org.enhydra.xml.xmlc.HTMLObjectImpl`
- Implements the interface `org.w3.dom.html.HTMLDocument`

When you instantiate the class that represents the page, you can use the methods of these classes and interfaces to manipulate the document and its content. “Using XMLC to generate Web pages” on page 85 provides several examples of using the DOM tree.

## DOM implementations

Although the W3C provides the standard DOM interface, actual implementations of the interface vary. XMLC uses a DOM implementation from the Apache Xerces project (for more information, see http://xml.apache.org/xerces-j/index.html). In addition, XMLC has a special, custom DOM derived from Xerces, called *Lazy DOM*.

Lazy DOM generally improves the runtime performance of XMLC document classes by minimizing the number of DOM nodes instantiated. For documents in which the application does not modify or access the majority of the document, the Lazy DOM will increase runtime performance significantly over Xerces. However, if the application does access a majority of nodes in a document, the Lazy DOM may be slower than Xerces.

If you are using POs and the `writeDOM()` or `writeHTML()` methods, you can set the XMLC_DOM_STATS option in your multiserver config file and it will generate runtime statistics to the log file for each page. This can be useful in determining how much of the DOM is being expanded, and whether using Lazy DOM will yield performance improvements.

**Note** Searching the DOM tree will expand all the document nodes, and applications should thus avoid doing so to achieve acceptable performance.

The Lazy DOM API is the same as the Xerces API, so you never have to modify your code to change the DOM that you use.

XMLC uses Lazy DOM by default for HTML and generic XML documents. You can also build DTD-specific derived DOMs, such as WML, on the Lazy DOM. To use the Xerces DOM instead, use this command-line option:

```
-dom xerces
```

You may also specify the DOM in the metadata `<documentClass>` element. For more information, see "Using XMLC metadata files" on page 84, and Appendix B, "XMLC metadata file schema."

### How Lazy DOM works

When using Lazy DOM, all instances of an XMLC document class share a read-only *template DOM*. Each instance of a document class starts out with just the top-level `Document` object. XMLC does not expand the `Document` object's child nodes. When an application accesses a node of the instance DOM via one of the DOM API methods, XMLC copies nodes from the template DOM into the instance DOM.

XMLC expands all direct children of a node when an application accesses any child of the node. It expands attribute nodes separately from children, and it expands all attributes of an element simultaneously. The formatter methods that convert the documents to text files will traverse the tree without expanding unexpanded nodes. A node can exist without its parent being expanded if accessed via its ID attribute accessor method. XMLC uses this to optimize `getElementXXX()` methods.

## How to use XMLC

The most common use of XMLC is for developing a presentation layer for an Enhydra application; however, you can also use XMLC without Enhydra. For example, you can use XMLC to create a standalone application that generates a semi-dynamic Web site, such as a catalog that is updated whenever you run the application. See "Using XMLC to generate Web pages" on page 85 for an example.

### Simple XMLC example

You can use XMLC to dynamically alter a Web page. A typical scenario is:

- A Web page designer and a Web developer agree that a certain page will contain one dynamic element, and they assign it the ID attribute "TheTime." They must assign a unique identifier (ID attribute) to each HTML element that will contain dynamic content within a page.

**Note** The ID attributes must be unique within a single HTML page. ID attributes can re-used in separate HTML pages.

- The designer creates a first version of the page, which includes a SPAN tag with an ID attribute value "TheTime," shown in the code sample that follows.
- The developer compiles the page with XMLC and then adds Java code to manipulate the named element.

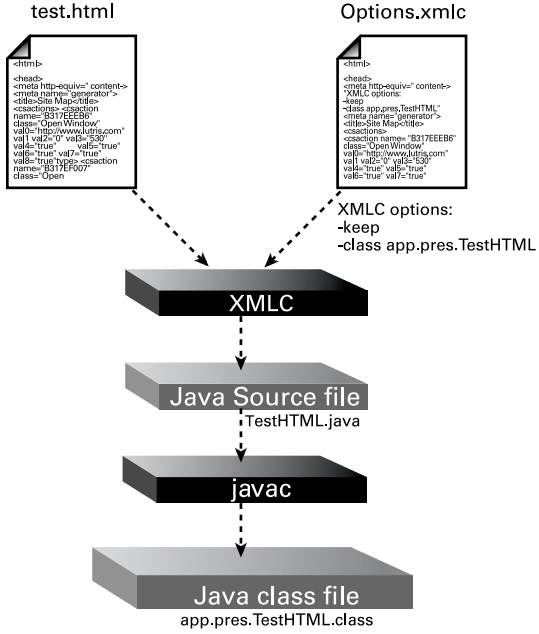
```
<HTML>
<BODY>
The current time is <SPAN ID="TheTime">00:00:00</SPAN>
</BODY>
</HTML>
```

When the developer compiles `Test.html`, XMLC:

- Parses `Test.html`.
- Generates a Java source-code file named `TestHTML.java`. XMLC deletes this file after compiling it, unless you provide the `-keep` command-line option to retain the source file.
- Calls the Java compiler, `javac`, to generate the Java class `TestHTML`.

Figure 5.4 illustrates these operations:

**Figure 5.4** XMLC at work




The `TestHTML` class contains methods that can access elements in the original document. For the Web page in the previous example, `TestHTML` includes a method named `setTextTheTime()` that the application can use to change the text in the corresponding `SPAN` tag.

XMLC generates a `setText`*`xxx()`* method for each `ID` attribute inside of a `SPAN` tag that it encounters in the input document. The *`xxx`* in the method name is replaced by the capitalized spelling of the ID attribute value of the `SPAN` tag.

**Note** To conform to Java conventions, XMLC automatically capitalizes the first character of the ID name. For example, if you specify an ID attribute "foo," XMLC creates the method `setTextFoo()`. If you specify an ID attribute "FOO," XMLC creates the method `setTextFOO()`.

After running `Test.html` through XMLC, you can write a Java program that uses the methods in the `TestHTML` class. For example, this code segment inserts the current time into the `Test.html` page.

```
import java.util.Date;
public String myTimeFunction() {
   String now = new Date().toString();
   TestHTML doc = (TestHTML)comms.xmlcFactory.create(TestHTML.class);
   doc.setTextTheTime(now);
   return doc.toDocument();
}
```

The `doc.toDocument()` method returns a string representation of the document. For more examples using XMLC to generate dynamic Web content, see "Using XMLC to generate Web pages" on page 85.

# Using the XMLC command

This section provides an overview and examples of some of the most commonly used XMLC command-line options. For a detailed description of all the XMLC command-line options, see "XMLC command-line options" on page 98.

## Command syntax

The syntax of the XMLC command is

`xmlc [`*`options`*` | `*`optfile`*`.xmlc ...] `*`docfile`*

where *`options`* is one or more command-line options, *`optfile`*`.xmlc` is one or more options files or metadata files, and *`docfile`* is the document to compile. For information on using options files, see "Using an options file" on page 83. For information on using metadata files, see "Using XMLC metadata files" on page 84.

Each XMLC option begins with a hyphen (`-`) and can be followed by an argument (`arg`):

`-`*`opt`* `[`*`arg`*`]`

For example:

```
-class fooBar
```

**Tip** Instead of specifying options on the command line, you can provide an `.xmlc` file containing the options. See "Using an options file" on page 83.

**Note** The Kelp tools include an XMLC wizard that lets you set XMLC options, select HTML files for compilation, and control how XMLC builds DOM classes from input files. For more information about Kelp, see Chapter 4, "Using Enhydra Kelp."

## Changing the Java class name

By default, XMLC creates a class with the same name as the input file. For example, for an HTML page `Foo.html`, XMLC creates a class named `Foo` in file `FooHTML.class`.

The `-class` option changes the name of the Java class file that XMLC creates. You can also specify a package name in the `-class` argument, which causes the class to be generated in the specified package.

### Example

This command compiles `Simple.HTML` and generates a class named `MyDocClass`:

```
xmlc -class MyDocClass Simple.HTML
```

This command creates a class named `SimpleHTML` in a package named `simpleApp.presentation`:

```
xmlc -class simpleApp.presentation.SimpleHTML Simple.HTML
```

## Saving the Java source-code file

By default, XMLC deletes the Java source file it generates and saves only the class file. If, however, you are debugging your project, you may need to keep the Java files. The `-keep` option saves the Java source file that XMLC generates as input to the Java compiler.

When you specify the `-keep` option, XMLC saves the source file using the same name it uses for the generated class file. If you use the `-keep` option without the `-class` option, XMLC creates a Java source file with the same name as the input file, but with a `.java` suffix.

### Example

This command compiles a file named `Simple.HTML` and saves the Java source-code file, `Simple.java`:

```
xmlc -keep Simple.html
```

## Modifying URLs

XMLC can modify the URLs in an HTML page it compiles. This can be useful, for example, with an Enhydra application in which you want to maintain a working storyboard during development. You can create template HTML pages with links among the pages that end in "`.html`"—then, when you compile the pages, you need to replace these URLs with links that end in "`.po`" to link to presentation objects at runtime.

XMLC provides three options for modifying URLs:

- `-urlmapping` option converts all occurrences of a specified URL to a different URL.
- `-urlregexpmapping` option converts URLs that match the first specified regular expression to new URLs matching the second specified regular expression.
- `-urlsetting` option converts the URL for a specified ID to a new URL.

**Note** You can use these options with the `-docout` option to write a new HTML file with updated URLs instead of producing a class file.

### Example

This command compiles `Simple.HTML`, producing a class in which all instances of the URL `Edit.html` are replaced with the URL `Edit.po`:

```
xmlc -urlmapping 'Edit.html' 'Edit.po' Simple.html
```

## Specifying the HTML parser

XMLC uses an HTML parser when it compiles an HTML input file, and an XML parser when it compiles an XML input file. You can change the parser that XMLC uses with the `-parser` option. The default parser for HTML is HTML Tidy. The only XML parser currently available is the Xerces parser.

**Note** Earlier versions of XMLC used the Swing HTML parser by default. The Swing parser generates different DOM trees for nonconforming HTML than does the HTML Tidy parser. Thus, if you previously compiled nonconforming documents with the Swing parser and do not want to make changes to your code, you might need to specify that XMLC use the Swing parser.

### Example

This command compiles `Simple.HTML` using the Swing parser instead of the HTML Tidy parser:

```
xmlc -parser swing Simple.HTML
```

## Deleting mock-up data

The `-delete-class` option deletes any elements in the input file that have the specified `CLASS` attribute value. This lets you maintain a Web page with mock-up data that is not included in the generated document.

### Example

The following table row is tagged with the class name `discardMe`:

```
<tr class="discardMe">
   <td>Van Halen</td>
   <td>5150</td>
   <td>Blah</td>
</tr>
```

The following option causes XMLC to discard any HTML tags whose `CLASS` attribute is `discardMe`:

```
-delete-class discardMe
```

## Diagnosing problems

You can use several of the XMLC options to help understand the results of a compilation:

- `-verbose` traces the overall execution of a compilation.
- `-parseinfo` traces the execution of the parser. It can be useful for debugging page problems.
- `-parser tidy` uses the Tidy parser to validate the HTML. The Tidy parser will output messages to help you debug your files.
- `-info` produces a dump of the compiled page, including a list of all IDs and all URLs in the page.
- `-methods` generates a list of all the methods generated for each class that XMLC creates.

**Note** You can use the `-nocompile` option with any of the above options to produce a listing of information without generating a class file.

### Example

This command compiles `Simple.HTML` and displays all available information about the compilation process:

```
xmlc -verbose -parseinfo -info -methods Simple.html
```

## Using an options file

Instead of entering XMLC options directly, it is often convenient to use an options file to specify options. An options file is a text file that contains a line for each XMLC option to use. You can use more than one options file at a time, and you can specify options on the command line in addition to naming options file(s).

- If you specify more than one options file in a command line, XMLC processes them one at a time, from left to right, starting with the first (leftmost) file named on the command line.
- If you specify individual options in addition to one or more options files, XMLC processes the individual options after the options files.

**Note** Some options, such as `-implements`, can have multiple values. Other options, such as `-parser`, can have only one value. If you include multiple uses of an option that can have only one value, XMLC applies the last value indicated.

### Options file format

An XMLC options file has the following format:

- Each line containing an option must start with the hyphen (`-`) character, followed by the option name, and then any arguments.
- XMLC uses whitespace to separate items.

  If you need to use an item that itself contains whitespaces, you can enclose that item with quotation marks. You can use either single (`'`) or double (`"`) quotes, as long as you use matching pairs to surround the text.
- Comment lines begin with the number sign (`#`) character.
- XMLC ignores blank lines.
- You can use UNIX-style escape sequences such as newline (`\n`) and tab (`\t`). XMLC converts each such sequence into a single character when it parses the string.

### Example

The following command compiles `Simple.html` using the options file, `options.xmlc`:

```
xmlc options.xmlc Simple.html
```

The file `options.xmlc` contains these lines:

```
-urlmapping 'Edit.html' 'Edit.po'
-urlmapping 'DiscCatalog.html' 'DiscCatalog.po'
-urlmapping '../personMgmt/Exit.html' '../personMgmt/Login.po?event=logout'
-delete-class discardMe
```

## Using XMLC metadata files

Instead of using an options file, you can use an XMLC metadata file to specify how a XMLC will process a document. You can combine the use of metadata files, command-line options, and options files for a single document. However, metadata files are more structured and allow more flexibility in document processing directives than command line options. In general, it will be simpler to use one method rather than a combination.

The XMLC metadata schema is described in detail in Appendix B, “XMLC metadata file schema.”

XMLC metadata files are XML files that can specify directives for the entire document, as well as directives for specific elements within the document. In addition to specifying how a document is to be compiled, an XMLC metadata file can also specify requirements on the structure of portions of the document. This allows you to detect changes in the document that may affect the application code if the application does not compile after a change to the document.

XMLC metadata files also enable you to have a common set of directives for multiple versions of a document. You can use a single metadata file for multiple localized versions of a document (for example, English and French), or multiple markup language versions (for example, HTML and WML) of a document.

You use an XMLC metadata file the same way that you use an options file:

```
xmlc [options | optfile.xmlc ...] docfile
```

where *optfile*.xmlc is one or more metadata files. An XMLC metadata file must have a filename extension of .xmlc. XMLC can differentiate between a metadata file and an options file, because the metadata file contains an XML header.

# Using XMLC to generate Web pages

Although the most popular use of XMLC is for developing the presentation layer of an Enhydra application, you can also use it in other ways. For example, you can also use XMLC to:

- Generate and manipulate XML data to be used in business-to-business (B2B) transactions.
- Compile HTML templates that are used by a standalone application to generate HTML files and save them to the file system.

This section describes some of the most important XMLC programming techniques in the context of a standalone application, DynaCat. You can also apply these techniques to Enhydra applications.

## DynaCat sample application

DynaCat is a standalone Java application that illustrates some XMLC programming techniques, including populating a table and filling in a form. It is an example of a *semi-dynamic* Web application, which generates HTML files each time it is run, in contrast to a standard Web application, which generates HTML files "on the fly," in response to client browser requests.

To the Web server, the HTML files generated by a semi-dynamic application appear just like static HTML files; however, their content can actually come from a database or some other data source. This makes a semi-dynamic application ideal for applications such as catalogs, which do not need to be updated every time a user access them, but rather on an occasional basis.

The source code for DynaCat is in:

```
<enhydra_root>/examples/semidynamic
```

DynaCat has these files:

- DynaCat.java, the source code for the application
- Disc.java, a simple class representing a compact disc
- Catalog.html, the template file for a disc catalog (HTML table)
- Form.html, the template file for an input form
- build, a shell script for building the application

For simplicity, DynaCat by default uses hard-coded string data. However, it also has a method that gets data from the InstantDB DiscRack database instead. To use this

method, you must install and configure InstantDB, and create and populate the DiscRack database as described in Chapter 7, "Using InstantDB."

DynaCat's `main()` method controls the program. It calls methods to get data, manipulate the template document objects, and then save the output into HTML files.

## Building DynaCat

To build DynaCat, use the provided build script.

1. Make `<enhydra_root>/examples/semidynamic` the working directory.
2. Make sure your CLASSPATH contains `<enhydra_root>/examples/semidynamic`.
3. In a shell window (an Enhydra shell on Windows), enter the following command:

   ```
   ./build
   ```

   You will see a few messages in the shell window as XMLC compiles the template pages. The build script creates the following files:

   - `CatalogHTML.java` and `FormHTML.java`, the source files that XMLC creates from the template files.
   - `CatalogHTML.class` and `FormHTML.class`, the class files that XMLC creates by compiling the source files.
   - `Disc.class`, the class file for a Disc object.

**Note** If you have any problems building DynaCat, make sure your `CLASSPATH` environment variable includes a reference to `enhydra.jar`.

**Note** The `build` script needs a fully qualified path to `xmlc` if `<enhydra_root>/bin` is not in your `PATH`.

## Running DynaCat

To run DynaCat, enter the following command in the shell window:

```
java DynaCat
```

You will see the following messages displayed in the shell window:

```
WRITING HTML TO catalog_dynamic.html
WRITING HTML TO form_dynamic.html
```

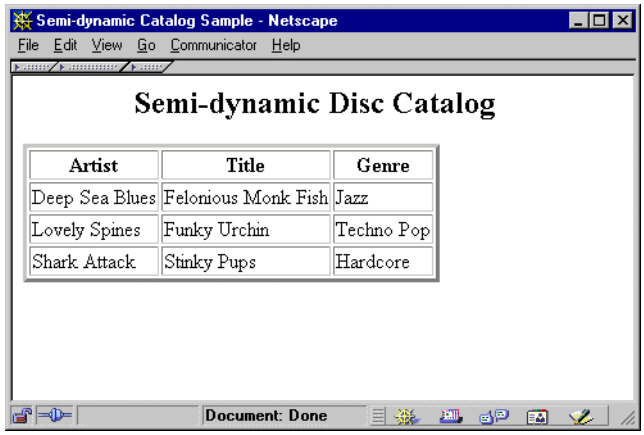
Load these pages in your browser to see the final results, as shown in Figure 5.5 and Figure 5.6.

## Writing the generated HTML output files

After building the pages based on the document templates and the data, DynaCat writes output HTML pages using the `DOMFormatter` object:

```
File outFile = new File("catalog_dynamic.html");
DOMFormatter formatter = new DOMFormatter();
formatter.write(catalogPage, outFile);
```

**Figure 5.5** DynaCat catalog page




## Populating a table

Populating an HTML table with data from a database or some other data source is a common application task.

**Note** Using XMLC to work with tables is also discussed in *Getting Started with Lutris Enhydra*.

### About the catalog page

`Catalog.html` is the template HTML page containing the table that DynaCat populates. When you compile this file with XMLC, it creates a DOM representation of the page that DynaCat uses.

This code shows a portion of `catalog.html`:

```
...
<tr>
   <th>Artist</th>
   <th>Title</th>
   <th>Genre</th>
</tr>

<tr id="TemplateRow">
   <td><SPAN id="Artist">Van Halen</SPAN></td>
   <td><SPAN id="Title">Fair Warning</SPAN></td>
   <td><SPAN id="Genre">Good Stuff</SPAN></td>
</tr>

<tr class="discardMe">
   <td>Van Halen</td>
   <td>5150</td>
   <td>Blah</td>
</tr>
...
```

The table has a heading row followed by four template (placeholder) data rows with a `CLASS` attribute of “discardMe.” These template rows let the designer see how the

page will look after the program runs. When XMLC compiles the page, it uses the `-delete-class` option to remove these rows.

This is the XMLC command line used to compile the page:

```
xmlc -class cataloghtml -delete-class discardMe catalog.html
```

The template row left in the table has an ID attribute with value "TemplateRow," which DynaCat uses to access the row object in the DOM tree.

To add a disc description to the table, DynaCat accesses the template row object, modifies the data in the row, clones it to create a new row object, and adds the cloned row to the table in the DOM tree. After it has added all data to the table, DynaCat removes the template row and then writes the document out to a HTML file.

## Populating the table

To populate the table, DynaCat first instantiates a `CatalogHTML` object based on the template HTML file:

```
CatalogHTML catalogPage = new CatalogHTML();
prepHTML( catalogPage );
```

The `prepHTML()` method prepares the HTML page for modification: it removes the `ID` attributes from the template row to prevent duplicate ID values, which are not allowed by the DOM.

```
private static void prepHTML(CatalogHTML page) {
   try {
      HTMLTableRowElement templateRow = page.getElementTemplateRow();

      // Remove ids to prevent duplicates
      // (browsers don't care, but the DOM does)
      templateRow.removeAttribute("id");
      HTMLElement artistCellTemplate = page.getElementArtist();
      HTMLElement titleCellTemplate = page.getElementTitle();
      HTMLElement genreCellTemplate = page.getElementGenre();

      artistCellTemplate.removeAttribute("id");
      titleCellTemplate.removeAttribute("id");
      genreCellTemplate.removeAttribute("id");
...
} //prepHTML
```

The `getDiscs()` method returns a group of discs that can be manipulated later.

```
private static Vector getDiscs() {
   Vector v = new Vector();
   v.addElement(new Disc( 1, "Felonious Monk Fish", "Deep Sea Blues", "Jazz", true ));
   v.addElement(new Disc( 2, "Funky Urchin", "Lovely Spines", "Techno Pop", true ));
   v.addElement(new Disc( 3, "Stinky Pups", "Shark Attack", "Hardcore", true ));
   return v;
}
```

DynaCat iterates through the retrieved records and calls a method that adds the contents of each record to the page:

```
vRecords = getDiscs();
Enumeration eRecords = vRecords.elements();
while ( eRecords.hasMoreElements() ) {
   Disc disc = (Disc)eRecords.nextElement();
   addDiscToPage(catalogPage, disc); // Add disc as table row
}
```

The `addDiscToPage()` method, which is shown below, performs several operations to add the disc information to the HTML table. It:

- Accesses the DOM tree to retrieve a placeholder row in the disc table
- Fills in the title, artist, and genre information in the placeholder object
- Clones (copies) the object to produce a new row object
- Appends the row to the table object

```
public static void addDiscToPage(CatalogHTML page, Disc disc) {
   HTMLTableRowElement templateRow = page.getElementTemplateRow();
   Node discTable = templateRow.getParentNode();
   try {
      page.setTextArtist(disc._artist);
      page.setTextTitle(disc._title);
      page.setTextGenre(disc._genre);
      Node clonedNode = templateRow.cloneNode(true);
      discTable.appendChild(clonedNode);

   } catch( Exception ex ) {
      System.out.println("Error adding disc to HTML " + ex);
   }
} //addDiscToPage
```

## Populating forms

Another common application task is populating a form with default values. Although DynaCat does not process user input, it does illustrate how to populate a form with default values, which is useful with Enhydra or other Web applications. The `populateForm()` method in DynaCat populates the form elements with default values.

**Note** Using XMLC to work with forms is also discussed in Chapter 5, "DiscRack sample application," of *Getting Started with Lutris Enhydra*.

### About the form page

`Form.html` contains the form that DynaCat populates with data from the database. When you compile this file with XMLC, it produces a DOM tree representation of the page, which DynaCat can then manipulate.

This code snippet shows a portion of the `Form.html` file:

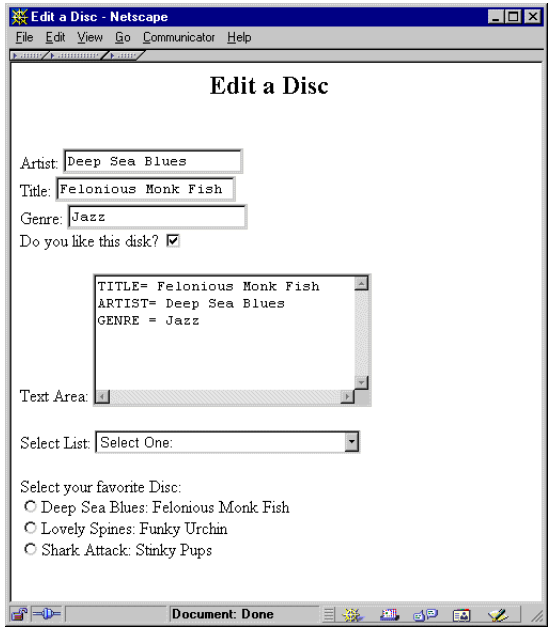
```
<FORM ACTION="DiscCatalog.html?Edit.po" NAME="EditForm" METHOD="GET">

<INPUT TYPE="hidden" NAME="discID" VALUE="invalidID" ID="DiscID">
Artist: <INPUT TYPE="TEXT" NAME="artist" id="Artist" ><BR>
Title: <INPUT TYPE="TEXT" NAME="title" id="Title" ><BR>
Genre: <INPUT  TYPE="TEXT" NAME="genre" id="Genre" ><BR>
Do you like this disk? <INPUT TYPE="checkbox" NAME="like" CHECKED ID="LikeBox">
<P>
Text Area: <TEXTAREA NAME="summary" ID="summary" ROWS=6 COLS=30></TEXTAREA>
<P>Select List:
<SELECT id="DiscList" Name="discID">
<OPTION ID="invalidID">Select One:</OPTION>
<OPTION id="templateOption">This is dummy text</OPTION>
</SELECT>
<P>
Select your favorite Disc:<BR>
```

```
<DIV ID="radioButtonGroup">
<INPUT TYPE=RADIO NAME="favorite" ID="templateRadio" VALUE="0">
</DIV>
</FORM>
```

The form page contains examples of these common HTML form elements, also shown in Figure 5.6:

- Text fields
- Text area
- Pull-down list boxes (select lists)
- Radio buttons
- Check boxes

**Figure 5.6** DynaCat form page




## Text fields

The following snippet shows the text fields in `Form.html`:

```
<INPUT TYPE="hidden" NAME="discID" VALUE="invalidID" ID="DiscID">
Artist: <INPUT TYPE="TEXT" NAME="artist" id="Artist" ><BR>
Title: <INPUT TYPE="TEXT" NAME="title" id="Title" ><BR>
Genre: <INPUT  TYPE="TEXT" NAME="genre" id="Genre" ><BR>
```

The `populateForm()` method uses the following statements to populate the text fields:

```
page.getElementDiscID().setValue(id);
page.getElementTitle().setValue(title);
page.getElementArtist().setValue(artist);
page.getElementGenre().setValue(genre);
```

The call to `getElement`*`xxx`*`()` returns the DOM object representing the text field with ID of *xxx*. For example, `getElementTitle()` returns the object with ID of "Title."

Notice that the text fields also have a NAME attribute, used when the form is submitted. Although NAME and ID do not have to be the same, it is usually most convenient to make them the same.

**Note** You can set the values of HIDDEN and PASSWORD form fields the same way you set them for text fields.

## Check boxes

The following snippet shows the checkbox in Form.html:

```
Do you like this disk? <INPUT TYPE="checkbox" NAME="like" CHECKED ID="LikeBox">
```

The populateForm() method also sets the checked state of the check box with this statement:

```
page.getElementLikeBox().setChecked(isLiked);
```

The setChecked() method takes a boolean argument; if true, the check box is checked by default, if false, the check box is unchecked by default.

## Radio buttons

The following snippet shows the radio button in Form.html:

```
<DIV ID="radioButtonGroup">
<INPUT TYPE=RADIO NAME="favorite" ID="templateRadio" VALUE="0">
</DIV>
```

The populateRadioGroup() method creates a radio button group with this code:

```
public static void populateRadioGroup(FormHTML page,
      Enumeration eRecords) {
   Disc disc;
   HTMLInputElement radioButtonTemplate = page.getElementTemplateRadio();
   radioButtonTemplate.removeAttribute("id");
   HTMLDivElement div = page.getElementRadioButtonGroup();
   while ( eRecords.hasMoreElements() ) {
      disc = (Disc)eRecords.nextElement();
      HTMLInputElement clonedRadioButton =
         (HTMLInputElement) radioButtonTemplate.cloneNode(true);
      clonedRadioButton.setValue(disc._title);
      Node radioText = page.createTextNode(disc._artist + ": " +
            disc._title);
      clonedRadioButton.appendChild(radioText);
      div.appendChild(clonedRadioButton);
      HTMLBRElement br = (HTMLBRElement)page.createElement("BR");
      div.appendChild(br);
      }
   div.removeChild(radioButtonTemplate);
}
```

The method clones the radio button template tag and sets its value to the title of each disc. Then, it creates a text node and appends it as a child of the new cloned radio button. Finally, it appends the cloned button to the DIV tag, which acts as a container for the whole radio button group, and appends a break (BR tag) to the DIV to make each radio button appear on a new line.

### Text areas

The following snippet shows the text area in `Forms.html`:

```
Text Area: <TEXTAREA NAME="summary" ID="summary" ROWS=6 COLS=30></TEXTAREA>
```

The following statement sets the default value of the multiline text field (`TEXTAREA`) to the value of the `discSummary` string.

```
String discSummary = "TITLE= " + disc._title + "\nARTIST= " + disc._artist + "\nGENRE = " +
     disc._genre;
page.getElementSummary().appendChild(page.createTextNode(discSummary));
```

Notice that `discSummary` contains newline characters (`\n`), to force newlines (line feeds) within the field.

### List boxes

The following snippet shows the list box in `Form.html`:

```
<P>Select List:
<SELECT id="DiscList" Name="discID">
<OPTION ID="invalidID">Select One:</OPTION>
<OPTION id="templateOption">This is dummy text</OPTION>
</SELECT>
```

The `populateSelectList()` method populates the multiple-option list box `SELECT` list. Because it requires all disc records, it performs its own "query," calling `getDiscs()`. Then, for each disc in the result set, it performs the following:

```
public static void populateSelectList(FormHTML page, Enumeration eRecords) {
...
   HTMLOptionElement clonedOption =
   (HTMLOptionElement) templateOption.cloneNode(true);
   clonedOption.setValue( Integer.toString(disc._ID) );
   Node optionTextNode =
   page.createTextNode(disc._artist + ": " + disc._title);
   clonedOption.appendChild(optionTextNode);
   selectList.appendChild(clonedOption);
...
```

The call to `cloneNode()` creates a clone (copy) of the template option, then the call to `setValue()` sets its value to the disc's ID.

**Note** To be meaningful, each `OPTION` must have a unique `VALUE` attribute. In this example, a disc's `ID` property is assumed to be unique.

The DOM method `createTextNode()` creates a new text node, then the call to `clonedOption.appendChild()` makes it the child (content) of the `OPTION` tag. Finally, `selectList.appendChild()` adds the new cloned option to the `SELECT` list.

## Manipulating JavaScript

Sometimes you need to set JavaScript content from your application. For example, you may need to do this if you want to set JavaScript variable values from a database, or if you want to maintain a working storyboard and replace the content of a `SCRIPT` tag at XMLC compile time.

Because SCRIPT elements are treated specially in the DOM, this requires a special technique. This technique works equally well for VBScript, or any other client scripting language. First, create a template script element like this:

```
<SCRIPT ID="myScript" TYPE="text/javascript">
var xyz = 0; // Dummy content
</SCRIPT>
```

The SCRIPT tag can go anywhere in the document's HEAD or BODY. It can have any "dummy" content desired, to make the application storyboard work. Then, as illustrated in the DynaCat application, use a method like setScript() to add content to the SCRIPT tag:

```
static void setScript(HTMLScriptElement element, String script) {
   Node child;
   while ((child = element.getFirstChild()) != null) {
      element.removeChild(child);  // Delete any children of SCRIPT
   }
   child = element.getOwnerDocument().createCDATASection(script);
   element.appendChild(child);  // Create new node to hold data.
}
```

You call this method with the HTMLScriptElement, whose content you want to replace, and with a String that represents the script content you want to insert. For example:

```
setScript(script, JS_STRING);
```

where JS_STRING has a value set from within the application, for example:

```
JS_STRING = "function popAlert() { alert('New Script Substituted in Page!!!') }";
```

To see this in action, remove the comment preceding this line in DynaCat.java, comment out the previous line that sets JS_STRING to an empty string, then rebuild and run the application. When you load the page, your browser will display the message "New Script Substituted in Page!!!"

An alternative technique is to use HIDDEN form elements and set their values as described in "Populating forms" on page 37.

## Compile-time includes

An *include* is a directive to incorporate the entire contents of one document within another document. You can use includes for common headers, footers, or other blocks of content that are duplicated in multiple pages.

XMLC supports compile-time includes, which occur when you run the XMLC command, in contrast to runtime includes, which occur when an application runs. The individual included files can take any form, but the resulting document must be a valid HTML file. XMLC converts the resulting document to a DOM and generates a class from it.

You must use the XMLC -ssi option when compiling your files in order to use includes. XMLC recognizes the syntax of server-side includes (SSI), as used in most web servers.

One approach to using includes with XMLC is to define an interface for each file to be included. Then, the generated DOM class must implement that interface. This way,

common code that operates on the included file can operate on generated objects based on the file.

### Syntax

The syntax for an XMLC include is:

```
<!--#include file="filename" -->
```

where *filename* is the name of the file to be included. When compiling the files with XMLC, you must use the `-ssi` option. If you don't, XMLC will ignore the directives. The directive must be used exactly as shown above. In particular:

- Do not put a space before the word `include`
- A space must precede the ending comment ( `-->`)
- The file name must be enclosed in quotes.

The filename can include a path, either an absolute path or a relative path. Relative paths are in relation to the location of the file containing the include directive. An included file may itself have an include directive (this is known as a "nested include"). You may have up to 64 levels of nested includes.

After XMLC processes an include, the resulting document is just as if the HTML from the included document were inserted into the including document, at the point of the include directive.

**Note** The resultant HTML file must be a legal, well-formed HTML document.

# Using XMLC with Enhydra

This section describes how to use XMLC with the Enhydra make system, and how to use Enhydra's automatic recompilation feature for pages generated by XMLC.

## Using the Enhydra make system

The Enhydra `make` command helps you build applications with XMLC. The `stdrules.mk` make file included with Enhydra in `<enhydra_root>/lib` defines a number of `make` variables that XMLC uses, as summarized in Table 5.1.

By convention, the Enhydra `make` rules cause XMLC to generate class files with the same base name as the source HTML files with "HTML" appended. For example,

when you compile `Simple.HTML` using the standard Enhydra `make` rules, XMLC generates the class file `SimpleHTML`.

**Table 5.1** Variables used by XMLC when compiling HTML

| Variable | Description |
|---|---|
| <*Language*>_CLASSES | The list of classes to be generated from <*Language*> using XMLC. Each class must be<br>• Named in the form *xxxx*<*Language*><br>• Generated from a <*Language*> file named *xxxx*.<*Language* extension> in <*Language*>_DIR. For example, the classes generated from an HTML file come from `xxxx.html` in `HTML_DIR`.<br>• Named in this list without the `.class` extension |
| <*Language*>_DIR | The name of the directory that contains the <*Language*> files for XMLC to compile. For example use `HTML_DIR` for HTML files. This directory name is relative to either the current directory or the root (`$ROOT`). |
| XMLC_<*Language*>_OPTS | Options to pass to XMLC when it compiles <*Language*> files. For example use `XMLC_HTML_OPTS` for HTML files. You can leave this variable unspecified or empty.<br>**Note:** You cannot specify both the `XMLC_`<*Language*>`_OPTS` and `XMLC_`<*Language*>`_OPTS_FILE` variables. |
| XMLC_<*Language*>_OPTS_FILE | The name of an XMLC options file with the extension `.xmlc`. This file contains options to pass to XMLC when it compiles <*Language*> files. For example use `XMLC_HTML_OPTS_FILE` for HTML files. You can leave this variable unspecified or empty.<br>**Note:** You cannot specify both `XMLC_`<*Language*>`_OPTS` and `XMLC_`<*Language*>`_OPTS_FILE` variables. |

## Example

This example shows a Makefile for compiling four HTML objects with XMLC:

```
ROOT = ../../../..

PACKAGEDIR = golfShop/presentation/xmlc/login

HTML_DIR = ../../html/login

HTML_XMLC_OPTS_FILE = login.xmlc

HTML_CLASSES = LoginHTML \
               LogoutHTML \
               CheckVersionHTML \
               NewAccountHTML

include $(Root)/config.mk
```

## Automatically recompiling with XMLC

You have seen how XMLC lets you keep an application's content and layout separate. Normally, though, if you change an HTML page, you have to recompile the application for the change to take effect. However, XMLC also has the capability to automatically recompile HTML pages as they change, without requiring that the application be stopped.

Automatic recompilation adds some additional overhead to your application. When a user visits a page, Enhydra checks the timestamp of the HTML page, and if it has changed, it automatically recompiles the page with XMLC. To reduce this overhead, you can compile the HTML page with XMLC yourself and configure the application to reload the class when a user first visits the page.

To enable automatic recompilation in your application:

1. Use `xmlcFactory` to instantiate your document objects as described in "Instantiating pages with xmlcFactory."
2. Set the `XMLC_AUTO_COMP` option in your `config.mk` file as described in "Setting up the application class directory structure."
3. Set `Server.ClassPath[]` in your `app.conf` file as described in "Setting up the application class directory structure."
4. Add options in your `config.mk` and `app.conf` files to update your class files or to recompile your HTML files at runtime as described in "Specifying how document classes are updated."
5. Optionally, add XMLC logging as described in "Adding XMLC logging to track the recompilation."

### Instantiating pages with xmlcFactory

Use `xmlcFactory` to instantiate your pages in your Java code. For example:

```
WelcomeHTML welcome = (WelcomeHTML)comms.xmlcFactory.create(WelcomeHTML.class);
```

**Caution** Do not use the standard `new` constructor to instantiate document objects if you want to enable automatic recompilation.

You can always use `xmlcFactory` to instantiate your pages. If you do not have automatic recompilation enabled, the `xmlcFactory.create` method simply calls `new`. If you set your code up to call `xmlcFactory.create`, you can add automatic recompilation later and need not make any changes.

### Setting up the application class directory structure

Most Enhydra applications deploy in a single Java archive (`.jar`) file that contains all the application classes. However, to use automatic recompilation, you must run your application directly from the class files, and you must store the HTML files in the same directories as the corresponding Java classes.

You need to set two values to make this happen:

- In the application's `config.mk` file, set `XMLC_AUTO_COMP=YES`.

  This tells XMLC to set up the directory structure and store the files in the appropriate locations. After you run XMLC, your files are found in `<appRoot>/output/lib/classes`, where *appRoot* is the application root directory.

- In your application configuration file, *appName*`.conf.in`, set `Server.ClassPath[]=<class_dir>` where *class_dir* is the relative path to the directory containing the application's class files. For example, `Server.ClassPath[] = <appRoot>/classes`.

## Specifying how document classes are updated

Once you have modified the HTML file as desired, you can:

- **Compile it yourself:** Compile the document with XMLC into a class file, and Enhydra loads the class as needed.
- **Let Enhydra recompile it:** Enhydra recompiles the HTML file into document class and reloads it as needed.

You must configure your application for one of the options shown in Table 5.2.

**Table 5.2** Types of automatic recompilation

| To have Enhydra | XMLC option in config.mk | Setting in app.conf file |
|---|---|---|
| Recompile the HTML file and reload resulting class | `-for-recomp` | `Server.XMLC.Autorecompilation = true` |
| Reload class only | `-generate both` | `Server.AutoReload = true` |

When you specify the `-generate both` option, XMLC generates both an interface and an implementation class.

When you specify the `-for-recomp` option, XMLC generates an interface, an implementation class, and an `.xmlc` file that contains the options used to compile the page. This ensures that the same options are applied when the page is dynamically recompiled.

**Caution** Using the `-for-recomp` option and the `new` constructor to instantiate a document object causes an error; always use the `xmlcFactory` interface to instantiate your document objects, as described in "Instantiating pages with xmlcFactory."

## Adding XMLC logging to track the recompilation

You can optionally add the XMLC option for logging in your `app.conf` file. For example:

```
Server.LogToFile[] = EMERGENCY, ALERT, CRITICAL, ERROR, XMLC
```

This adds information in the log file about XMLC automatic recompilation and reloading.

# XMLC reference

This section is a reference for the XMLC command-line options, XMLCUtil methods, and DOM classes and methods.

For complete reference information, see the online Javadoc, located in the `<enhydra_root>/doc` subdirectory.

## XMLC command-line options

To invoke XMLC from the command line, use this syntax:

```
xmlc [options] [optfile.xmlc ...] docfile
```

Here's a typical example of a command line is used to compile a HTML input file and write an output class file with a specified name:

```
xmlc -d ../../classes -class app.presentation.user.UserTable ../html/usertable.html
```

You can specify any number of the XMLC options shown in Table 5.3. See "Using the XMLC command" on page 29 for more information and examples of using several of the more common options.

**Table 5.3** XMLC command-line options

| Option | Description |
|---|---|
| `-class <class>` | Sets the fully qualified class name for the generated class or interface. See "Changing the Java class name" on page 81 for an example. |
| `-classpath <path>` | Passed on to `javac`. |
| `-d <dir>` | Specifies the destination directory for the class file. This option is passed on to javac. |
| `-delete-class <classname>` | Deletes all elements that have `CLASS` attribute with value `<classname>`; useful for removing mock-up data.<br>**Note**: This class name has nothing to do with a Java class.<br>You can include multiple instances of this option. |
| `-docout <outfile>` | Writes a static document to *outfile* instead of generating and compiling Java code. You can use this option for pages that have URLs that need mapping, but no dynamic content. |
| `-dom <DOMname>` | Specifies the DOM to use. Default is Lazy DOM Valid values are: xerces, lazydom |
| `-domfactory <classname>` | Specifies the Java class for creating DTD-specific documents. This option is not supported for HTML input documents. The DOM factory must have a constructor that does not take any arguments.<br>This class must:<br>• Implement the interface `org.enhydra.xml.xmlc.dom.XMLCDomFactory`<br>• Be on the `CLASSPATH` |

**Table 5.3** XMLC command-line options (continued)

| Option | Description |
|---|---|
| `-dump` | Displays the DOM tree for the input document. See the section "Diagnosing problems" on page 83 for an example. |
| `-extends <classname>` | Specifies the class that the generated document extends. This class must:<br>• Extend `XMLObjectImpl` for XML documents<br>• Extend `HTMLObjectImpl` for HTML documents<br>• Be available on the `CLASSPATH`<br>The class is normally an abstract class. |
| `-for-recomp` | Generates support for automatic class recompilation; the information is stored in a file with an `.xmlc` suffix appended to the class name, as specified with the `-class` option.<br>Implies the `-generate both` option. |
| `-g` | Passed on to `javac`. |
| `-generate <type>` | Specifies what XMLC generates:<br>• `class`: XMLC generates a class that does not depend on an interface (default).<br>• `interface`: XMLC generates only an interface.<br>• `both`: XMLC generates both an interface and an implementation class. The implementation has the suffix `Impl` appended to the class name, and uses the class name specified with the `-class` option.<br>• `implementation`: XMLC generates the class that implements the interface, but not the interface. |
| `-html:addattr <attr>` | Adds the specified attribute *attr* to the list of valid HTML attributes. The parser then allows the attribute for all tags.<br>**Note:** Used only by the HTML Tidy parser. |
| `-html:addtag <tag> <flags>` | Adds the specified *tag* to the list of valid HTML tags. The parser then allows the tag. The `tagname` is case-insensitive.<br>*flags* is a comma-separated list that contains the content model and other options that describe the tag. You can specify the following values:<br>• `inline`: Tag applies to character-level elements.<br>• `block`: Tag applies to block-like elements such as paragraphs and lists.<br>• `empty`: Tag does not have a closing tag.<br>• `opt`: Closing tag is optional for this tag.<br>You must specify at least one of the following flags: `inline`, `block`, or `empty`.<br>**Note:** Used only by the HTML Tidy parser. |
| `-html:addtagset <tagsetname>` | Adds a predefined set of tags to the list of valid HTML tags. You can specify:<br>• `cyberstudio`: Tags added by Adobe Cyberstudio, which are ignored by most browsers.<br>**Note:** Used only by the HTML Tidy parser. |
| `-html:frameset` | Deprecated and is ignored. |

**Table 5.3** XMLC command-line options (continued)

| Option | Description |
|---|---|
| `-html:old-class-constants` | Generates old-style, all uppercase class names. Available for compatibility with applications generated by older versions of XMLC, which generated HTML class attribute constant names in all uppercase. |
| `-implements <interface>` | Specifies the interface that the generated class will implement. You can include multiple instances of this option. |
| `-info` | Prints information about the document object, including IDs and URLs. |
| `-javac <prog>` | Specifies the name of the Java compiler to use. |
| `-javacflag <flag>` | Passes the specified flag to the javac program, including any leading hyphen (`-`) or plus (`+`) characters. You can include multiple instances of this option.<br>**Note:** Use the `-javacopt` option for compiler options that require values. |
| `-javacopt <opt> <value>` | Passes the specified option and value to pass to javac, including any leading hyphen (`-`) or plus (`+`) characters. |
| `-keep` | Saves the generated Java source file. See "Changing the Java class name" on page 81 for an example. |
| `-methods` | Prints the signature of each generated access method, and lists any methods or access constants that were not generated because they were not valid Java identifiers. |
| `-nocompile` | Do not compile the generated Java source file. |
| `-O` | Passed on to `javac`. |
| `-parseinfo` | Prints detailed information about the parsing of the page. |
| `-parser <parser>` | Specifies the parser that XMLC uses:<br>• `tidy`: Enables the HTML Tidy parser. This is the default HTML parser and always performs validation.<br>• `swing`: Enables the Swing parser for HTML. This parser always performs validation.<br>• `xerces`: Enables the Xerces parser for XML. This is the default XML parser and performs XML validation by default. |
| `-sourceout <sourceout>` | Specifies the root directory for source files generated by XMLC.<br>If you specify the `-keep` option, the generated source files are stored in this directory. |
| `-ssi` | Enable processing of server-side includes in the input document. |
| `-urlmapping <origURL> <newURL>` | Maps all occurrences of `<orgiURL>` to `<newURL>`. You can include multiple instances of this option. |
| `-urlregexpmapping <regexp> <replace>` | Maps all occurrences of the URL that matches regular expression `<regexp>` to the URL specified by `<replace>`. You can include multiple instances of this option.<br>This option uses the `gnu.regexp` package and recognizes regular expressions with POSIX extensions. |

**Table 5.3** XMLC command-line options (continued)

| Option | Description |
|---|---|
| `-urlsetting <id> <newURL>` | Changes the URL for the specified *<id>* to the specified *<newURL>*. You can include multiple instances of this option. |
| `-validate yes\|no` | Changes the default document validation mode of the parser. If you specify an option value that the parser does not support, XMLC generates an error. |
| `-verbose` | Generates useful output about the compilation process. |
| `-version` | Prints XMLC version number. If you do not specify any other options, XMLC quits after printing the version number.<br>**Note:** You do not need to specify a `docfile` with this option. |
| `-xcatalog <catalog>` | Specifies the catalog file to use for resolving external entities. You can use this option to specify local DTDs. |

## XMLCUtil class

The `XMLCUtil` class contains a number of utility methods for working with DOM representations of documents. This class extends `java.lang.Object`.

**Table 5.4** XMLCUtil methods

| Method | Description |
|---|---|
| `findFirstText` | Returns the first text descendent node of a specified element. |
| `getAttributeByName` | Returns the attribute object for a named attribute. |
| `getElementById` | Recursively searches for the element with the specified ID value, starting at a specified node. |
| `getFirstText` | Returns the first text descendent node of a specified element. |
| `getRequiredElementById` | Recursively searches for the required element with the specified ID value, starting at a specified node. |
| `printNode` | Prints a node and its children. |
| `replaceNode` | Replaces a node with a specified node from another document. |

## DOM classes and methods

This section summarizes some of the primary DOM classes and methods.

### DOM objects

The DOM is a set of general objects for XML documents. To use XML for a particular application, you can define a set of Element classes for that application. The DOM specification includes a set of classes for HTML documents.

The DOM actually supports two different views of a document:

- A flat collection of nodes, each of which has a different type
- An object-oriented hierarchy with inheritance

The node view provides a consistent, low-level interface to each tree node, which can be used in any environment, but might be more important for performance-critical applications. Table 5.5 shows the core node interfaces in the DOM.

**Table 5.5** Standard node interfaces in org.w3c.dom

| Object type | Description |
|---|---|
| `Node` | Represents a single node in the document tree. This is the primary interface for the DOM. |
| `NodeList` | Presents an interface for working with an ordered lists of nodes, such as the list of children of a node. |
| `NamedNodeMap` | Represents a collection of nodes that can be accessed by name. |

Table 5.6 shows the core XML interface objects in the DOM.

**Table 5.6** Core XML interface in the DOM

| Object type | Description |
|---|---|
| `Document` | Represents the entire HTML or XML document and is the root of the document tree. |
| `DocumentFragment` | Represents a portion of a document's tree. `DocumentFragment` objects are considered lighter weight versions of the `Document` object type. |
| `DocumentType` | Provides an interface to the list of entities that are defined for the document. |
| | **Note:** This object type does not have any children. |
| `Entity` | Represents an entity in an XML document. |
| `Entity Reference` | Refers to an entity node. |
| `Element` | Represents an element of a document. Each element can have attributes associated with it. |
| `Attr` | Represents an attribute of an `Element` object. The values allowed for an `Attr` are specified in the DTD for the language you are using. |
| `ProcessingInstruction` | Processor-specific instruction. |
| | **Note:** This object type does not have any children. |
| `Comment` | Comment in the source file, which starts with the "`<!--`" character sequence and ends with the "`-->`" character sequence. |
| | **Note:** This object type does not have any children. |
| `Text` | Textual element. |
| | **Note:** This object type does not have any children. |
| `CDataSection` | Used to escape blocks of text that would otherwise be regarded as markup. You can use `CDATA` sections to include material without having to escape every delimiter in the material. |
| | **Note:** This object type does not have any children. |
| `Notation` | Represents a notation that is declared in the DTD of the document. Used to declare the format of an unparsed entity or to declare processing instruction targets. |
| | **Note:** This object type does not have a parent or any children. |

## DOM Java interfaces

Although W3C does not provide implementations of the DOM, it does provide a set of Java interfaces and support objects for XML and HTML. You can find the Javadoc for the HTML interfaces in the online documentation provided with Enhydra.

Table 5.7 summarizes some of the commonly used HTML interfaces. You can see that these objects are named to match the familiar HTML tags:

**Table 5.7** Commonly used HTML interfaces in the DOM

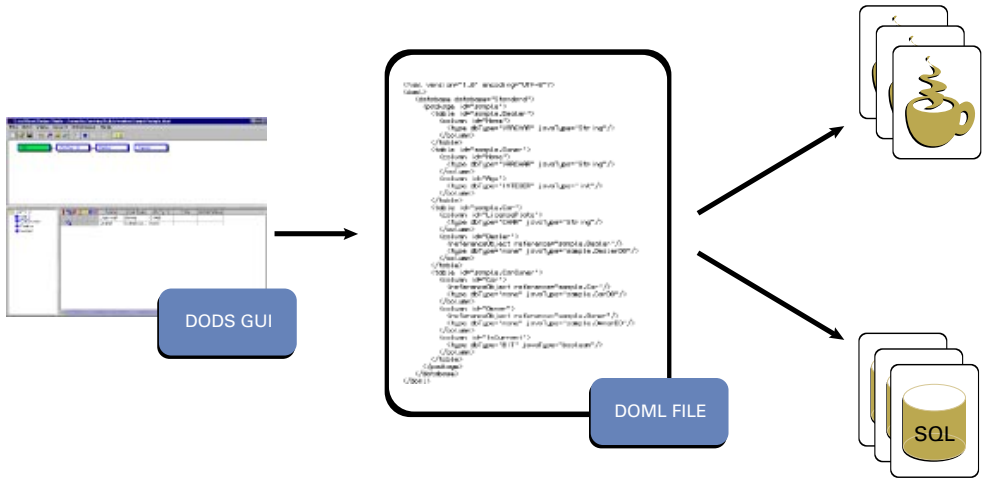
| Object type | Represents in document |
|---|---|
| `HTMLAnchorElement` | Link anchor |
| `HTMLBodyElement` | Body of the HTML document |
| `HTMLBRElement` | Line break element |
| `HTMLButtonElement` | Button element |
| `HTMLDocument` | Root of the HTML hierarchy, which contains the document's contents |
| `HTMLFontElement` | Font definition element |
| `HTMLFrameElement` | Frame definition |
| `HTMLHeadElement` | Head information of the HTML page |
| `HTMLHeadingElement` | Heading |
| `HTMLHRElement` | Horizontal rule |
| `HTMLLinkElement` | Link to an external resource |
| `HTMLOListElement` | Ordered list |
| `HTMLParagraphElement` | Paragraph |
| `HTMLPreElement` | Preformatted text |
| `HTMLTableElement` | Table |
| `HTMLTitleElement` | Title of the page |
| `HTMLUListElement` | Unordered list |

Chapter 6

# Using the Data Object Design Studio

The Data Object Design Studio (DODS) consists of three parts: an object-oriented GUI design tool that reads and writes DOML files, a set of code generators that reads DOML files to generate data-access code, and a set of underlying Enhydra classes to facilitate data access in Enhydra applications.

DODS lets you design the data-layer classes of an application that uses the Enhydra application framework. DODS then generates the Java source code for the data-layer classes and compiles them for you.




## Using the DODS graphical user interface

There are three different panels in the DODS GUI: There is a Graphical View, and below that, there is a Package/Object Tree and an Attribute table.

- The Graphical View panel displays the hierarchy of the data object you have created.
- The Tree is a directory structure of the packages that hold the data objects which will be created by DODS.

When the project is finished and built, this directory structure will be created with the appropriate `.java` files in each directory corresponding to the data objects in those packages.

- The Attribute table displays the attributes of a data object.

When a data object is selected in either the Tree or the Graphical View, its attributes are listed in the table. For example, if a customer has the attributes of name, phone number, and address, each of these attributes is listed in the Attribute table when the Customer object is selected.

## Running with parameters

The script that starts DODS, `<enhydra_root>/bin/dods`, can be given optional parameters.

```
dods [ <project file> [ <output directory> [ regen ] ] ]
```

If you specify a project file, it will be opened automatically when DODS starts.

If you specify an output directory, it will be preselected when you click the Build button.

If you provide `regen` as the third parameter, the DODS GUI will not load, but the named project file will be loaded, and the code generators will write source code to the named output directory. The `regen` parameter is useful for running DODS from a Makefile or script to completely rebuild an application. The project file can be thought of as the actual source code for the data layer of the application.

## Data Object editor

The Database tab in the Data Object editor dialog box has two main sections. Db Table Name specifies the table name. When you choose a name, keep in mind the naming restrictions of your database vendor. For example, there may be a maximum length for the name, or perhaps the name cannot be an SQL reserved word, or there may be names that are reserved for use by the database itself.

The following table describes DataBase Mapping Type box's options.

| Option name | Description |
|---|---|
| Every Class | Not supported in the current version of DODS. |
| Every Concrete Class | Not supported in the current version of DODS. |
| Entire Hierarchy | Not supported in the current version of DODS. |

| Option name | Description |
|---|---|
| Lazy Loading | This flag affects the behavior of the DO class (defined in the `DO.java` file for the data object). If it is checked, when you supply a known ObjectId to create a DO instance, the DO instance is created but the corresponding row in the table is not retrieved until the first `get()` or `set()` method call is made. This is useful when you are reconstituting a hierarchy of objects from rows in the table, but you will not always need the data for this particular object in that hierarchy.<br>Checking Lazy Loading delays the hit on the database until the moment the data is actually needed. Depending on how your application accesses its database, this can result in extra database hits, and thus hurt performance. If this flag is not checked, all the data members are retrieved from the table when the DO instance is created. If your application will not always need these data members, this can waste memory. |
| Cached | This flag affects the behavior of the DO class. If it is checked, all DO instances are stored in a cache inside the DO class. Subsequent queries of the table using the Query class will first check the cache before hitting the database. This is appropriate when different parts of your application hold references to the same DO instance. |
| Fully Cached | This flag affects the behavior of the DO class. If it is checked, the entire table is queried and cached when your application starts. This is appropriate for tables of static data that is accessed frequently and that will not change during the execution of your application.<br>Note that there is a `refreshCache()` method available if your application needs to refresh the cache with new values in the table. |

## Attribute editor

The Database tab in the Attribute editor dialog has the following elements. The Db Type pull-down menu holds a list of database column types that are valid for storing the Java type chosen for this attribute. Some column types require a size setting, which you enter in the Size edit box.

The following table describes the available options on the Database tab.

| Option | Description |
|---|---|
| Is an Index | If this option is checked, a database index is created on this column in the table. |
| Can be Null | If this option is not checked, the NOT NULL restriction is placed on the column. |
| Can be queried | If this option is checked, the `Query` class generated for this data object will include a `set()` method for this column to allow restricting the search. |
| Referenced DO must exist | If this option is checked and this attribute is a reference to another data object (that is, the Java type for this attribute is another DO), then a REFERENCES restriction is placed on the column. This enforces referential integrity between the tables for these two data objects. |
| Value is constant | If this option is checked, the attribute is made a constant data member in the DO class, and no column is created in the table for this data object. |

## Query classes

The `Query` classes generated by DODS allow you to search the database for DOs. attributes of the DO that are marked Can Be Queried will have a method `setQueryAttributeName` in the `Query` class. These `setQuery` methods allow you to prune the search results.

The `Query` classes generated by DODS all use the `QueryBuilder` helper class. The `setQueryAttributeName` methods invoke the `QueryBuilder.addWhereClause` method to construct the SQL `select` command that performs the search. After creating an instance of a `Query` class, you can obtain the associated instance of `QueryBuilder` to add extra clauses to the select command. Some developers elect to extend a `Query` class to implement new methods for such extra functionality. Using extra clauses, your `select` command can perform more complex searches.

**Note** The `QueryBuilder.NOT_EQUALS` qualifier will not work with DB2 databases. Use `<>` instead.

## Querying a view

You can use DO/Query classes to access a view instead of a table. For example, let's say that you have used DODS to create a DO named `Person`, and you want a view named `count_people`:

```
create view count_people (total) as select count(*) from person
```

You would create another DO named `Count`, specify its table name to be `count_people`, then add an integer attribute named `total` and mark that attribute as Can be Queried.

Click Build to let DODS generate the classes `PersonDO`, `PersonQuery`, `PeopleSQL.sql`, `CountDO`, `CountQuery`, and `CountSQL.sql`.

Discard the `CountSQL.sql` file, and remove the `create table count_people` statement from the `create_tables.sql` file that DODS generated.

Use the `create_tables.sql` file to define your tables in your database. Then manually create the `count_people` view using the create view above.

The `CountQuery` class can now be used just as you would use any `Query` class. The `CountDO` object returned by `CountQuery` will contain the `total` you want.

**Note** Only `get()` methods should be called on the `CountDO` object.

# DODS projects

In DODS, you create a data object entity for each data-layer class that your application requires. Each data object contains attributes.

An attribute describes either a piece of data (for example, int, String, or Date) or a reference to another data object. By using reference attributes, you create an interconnected hierarchy of data objects.

A collection of data object specifications is stored as a project file. You can reopen a project file and make changes to your data objects.

## Code generation

After you have designed your data objects, as described in "Using the DODS graphical user interface" on page 105, DODS generates the Java and SQL source code to implement them.

As a simple example, suppose your application stores and retrieves information about car dealers, car owners, and the cars themselves. For this application, you would create three database tables: `Dealer`, `Owner`, and `Car`.

The `Dealer` table has one column, `Name`, that stores that name of the car dealership as a String.

The `Owner` table has two columns. `Name` stores the name of the car owner as a String, and `Age` stores the age of the owner as an int.

The `Car` table has one column, `LicensePlate`, that stores the license plate number of the car as a String.

DODS will create the following files for this application.

**Table 6.1** Files generated by DODS for car dealership application

| Generated file | Description |
|---|---|
| `DealerSQL.sql` | SQL file that contains the `CREATE TABLE` statement for the `Dealer` table. |
| `DealerDO.java` | Java file that represents a new or existing row in the `Dealer` table. |
| `DealerQuery.java` | Java file that retrieves `DealerDO` objects for row in the `Dealer` table. |
| `OwnerSQL.sql` | SQL file that contains the `CREATE TABLE` statement for the `Owner` table. |
| `OwnerDO.java` | Java file that represents a new or existing row in the `Owner` table. |
| `OwnerQuery.java` | Java file that retrieves `OwnerDO` objects for row in the `Owner` table. |
| `CarSQL.sql` | SQL file that contains the `CREATE TABLE` statement for the `Car` table. |
| `CarDO.java` | Java file that represents a new or existing row in the `Car` table. |
| `CarQuery.java` | Java file that retrieves `CarDO` objects for row in the `Car` table. |

### One-to-many relationships

Suppose the car dealerships only sell new cars. The dealership sells many cars, but only sells a given car once. In this case, you have a one-to-many relationship. You'd represent this relationship by adding another column, `Dealer`, to the `Car` table that contains a reference to the foreign key of `Dealer`. The Java type of this attribute is a `DealerDO`, and the database type is a `REFERENCE`.

### Many-to-many relationships

Car owners can own multiple cars at a time, and may also sell cars to each other. In this case, the car can have multiple owners, which is a many-to-many relationship. To represent this relationship, you need to create a new table, `CarOwner`. It has three columns: `Car`, `Owner`, and `IsCurrent`.

Car contains a reference to the foreign key of the Car table. The Java type is CarDO, and the database type is a REFERENCE.

Owner contains a reference to the foreign key of an Owner. The Java type is OwnerDO, and the database type is REFERENCE.

IsCurrent is a boolean value that is true if the owner referenced by the OwnerDO is the current owner of the car. The Java type is boolean, and the database type is BIT.

DODS would generate the following files for this new table:

**Table 6.2** DODS-generated files for CarOwner table

| Generated file | Description |
|---|---|
| CarOwnerSQL.sql | SQL file containing the CREATE TABLE statement for CarOwner table. |
| CarOwnerDO.java | Java file that maps instances of many to many relationships. |
| CarOwnerQuery.java | Java file that retrieves instances of many to many relationships. |

## Creating the tables

Before you use the Java classes generated by DODS, you first need to use the SQL files to create your database tables. To make this easier, DODS creates the file create_tables.sql. create_tables.sql is a concatenation of DealerSQL.sql, OwnerSQL.sql, CarSQL.sql, and CarOwnerSQL.sql. Use this file with the interactive SQL tool for your database to create the tables.

## Using the DO classes to create data

After the tables are created in your database, you can use the Java classes generated by DODS. For example, to create a car dealership, a car owner, a car, and to establish the relationships between them all, you'd do the following:

```
// create a dealer
DealerDO autoWorld = DealerDO.createVirgin();
autoWorld.setName( "Bob's Auto World" );

// create a car
CarDO lemon = CarDO.createVirgin();
lemon.setLicensePlate( "ABC123" );
lemon.setDealer( autoWorld );

// create an owner
OwnerDO joe = OwnerDO.createVirgin();
joe.setName( "Joe Lanechange" );
joe.setAge( 16 );

// joe is the current owner of the car
CarOwnerDO currentOwner = CarOwnerDO.createVirgin();
currentOwner.setCar( lemon );
currentOwner.setOwner( joe );
currentOwner.setIsCurrent( true );
currentOwner.save();
```

The call to currentOwner.save() does the following:

- Writes, or saves, the OwnerDO, Joe Lanechange, to the Owner table in the database.
- Writes the CarDO, lemon, to the Car table in the database.
- Writes the CarOwnerDO object, currentOwner, into the CarOwner table in the database.

Because the `CarOwnerDO` object, `currentOwner`, holds references to other DO objects, `CarDO` and `OwnerDO`, it must first write them to the database before it can write itself to the database.

Because the `CarDO`, `lemon`, holds a reference to a `DealerDO`, `autoWorld`, when `currentOwner.save()` calls `lemon.save()`, `lemon.save()` itself calls `autoWorld.save()`. So, the dealership gets saved as the car.

You could also explicitly call the `save()` methods. The DO objects remember whether they are synchronized with the database, and their `save()` methods do nothing if their data has already been saved to the database. For example, if you explicitly called `save()` on all the DO objects, a call to `currentOwner.save()` would only write itself to the `CarOwner` table in the database.

## Using the Query classes to retrieve data

Now that you've saved data in the tables, you can use the `Query` classes to retrieve objects from the database. The following code retrieves the `DealerDO` created in the previous section.

```
DealerQuery dealerQuery = new DealerQuery();
dealerQuery.setQueryName( "Bob's Auto World" );
DealerDO autoWorld = dealerQuery.getNextDO();
```

This code returns the Cars sold by that Dealer:

```
CarDO[] cars = autoWorld.getCarDOArray();
```

You could do the same thing with the following code.

```
CarQuery carQuery = new CarQuery();
carQuery.setQueryDealer( autoWorld );
CarDO[] cars = carQuery.getDOArray();
```

The `getCarDOArray()` method, however, is much easier to use.

To find the car by its license plate, use the following code.

```
CarQuery carQuery = new CarQuery();
carQuery.setQueryLicensePlate( "ABC123" );
carQuery.requireUniqueInstance();  // plate numbers are unique
CarDO lemon = carQuery.getNextDO();
```

To find out who currently owns the car:

```
OwnerDO[] lemonOwners = lemon.getOwnerDOArray_via_CarOwner();
if ( lemonOwners[0].getName().equals( "Joe Lanechange" ) )
System.err.println( "Joe owns the lemon" );
```

`CarDO.getOwnerDOArray_via_CarOwner()` is another convenience method generated by DODS. The following code does the same thing.

```
CarOwnerQuery carOwnerQuery = new CarOwnerQuery();
carOwnerQuery.setQueryCar( lemon );
OwnerDO[] lemonOwners = carOwnerQuery.getOwnerDOArray();
if ( lemonOwners[0].getName().equals( "Joe Lanechange" ) )
System.err.println( "Joe owns the lemon" );
```

## Using the DO classes to delete data

Suppose you wanted to remove the car owner from the database.

```
OwnerQuery ownerQuery = new OwnerQuery();
ownerQuery.setQueryName( "Joe Lanechange" );
OwnerDO joe = ownerQuery.getNextDO();
joe.delete();
```

The row for Joe Lanechange is removed from the `Owner` table.

Because you created a row in the `CarOwner` table that referenced the row for Joe in the `Owner` table, that row in the `CarOwner` table must also be deleted. If it is not deleted, the database will lose referential integrity. The `CarOwner` table would have a reference to an `Owner` row that no longer exists.

Some databases, like Oracle, have a feature called cascading delete. The database detects a dangling reference like the reference to Joe in the `CarOwner` table, and will automatically delete the row in `CarOwner`. When using DODS to describe your database tables, you specify the database vendor. If that vendor does not have the cascading delete feature, DODS will generate additional code in the `DO.delete()` method to perform the cascading delete to preserve referential integrity in the database.

## Using comparison operators

You can use `Query` objects to find specific data in the database. For example to find all owners who are 16 years old, use this code:

```
OwnerQuery ownerQuery = new OwnerQuery();
ownerQuery.setQueryAge( 16 );
OwnerDO[] troubleMakers = ownerQuery.getDOArray();
```

To find owners younger than 21:

```
OwnerQuery ownerQuery = new OwnerQuery();
ownerQuery.setQueryAge( 21, QueryBuilder.LESS_THAN );
/*
* This produces the SQL query:
*      SELECT Owner.* FROM Owner
*      WHERE  Owner.Age < 21
*/
OwnerDO[] youngDrivers = ownerQuery.getDOArray();
```

Every `Query` class uses a `QueryBuilder` object. Calls such as `ownerQuery.setQueryAge( 21, QueryBuilder.LESS_THAN );` utilize `QueryBuilder` comparison operators.

The following table lists `QueryBuilder`'s comparison operators.

**Table 6.3** QueryBuilder comparison operators

| Operator | Description |
|---|---|
| `EQUAL` | Equal to (==) |
| `NOT_EQUAL` | Not equal (!=) |
| `LESS_THAN` | Less than (<) |
| `LESS_THAN_OR_EQUAL` | Less than or equal to (<=) |
| `GREATER_THAN` | Greater than (>) |
| `GREATER_THAN_OR_EQUAL` | Greater than or equal to (>=) |

**Table 6.3** QueryBuilder comparison operators (continued)

| Operator | Description |
|---|---|
| IS_NULL | Has a null value |
| IS_NOT_NULL | Does not have a null value |
| CASE_INSENSITIVE_EQUAL | Applies only to strings. Equals, or matches, regardless of case. |
| CASE_SENSITIVE_CONTAINS | Applies only to strings. Contains the string, and case-sensitive. |
| CASE_INSENSITIVE_CONTAINS | Applies only to strings. Contains the string, regardless of case. |
| CASE_SENSITIVE_STARTS_WITH | Applies only to strings. Begins with the string, and case-sensitive. |
| CASE_INSENSITIVE_STARTS_WITH | Applies only to strings. Begins with the string, regardless of case. |
| CASE_SENSITIVE_ENDS_WITH | Applies only to strings. Ends with the string, and case-sensitive. |
| CASE_INSENSITIVE_ENDS_WITH | Applies only to strings. Ends with the string, regardless of case. |

## Using QueryBuilder for advanced queries

You can access the `QueryBuilder` object to add SQL `WHERE` clauses to perform more sophisticated searches.

For example, the following code will find Owners who are 16 years or older, and who have purchased cars from a dealer whose name begins with "Bob":

```
OwnerQuery ownerQuery = new OwnerQuery();
ownerQuery.setQueryAge( 16, QueryBuilder.GREATER_THAN_OR_EQUAL );
QueryBuilder qb = ownerQuery.getQueryBuilder();
qb.addWhere( OwnerDO.PrimaryKey, CarOwnerDO.Owner );
qb.addWhere( CarOwnerDO.Car,     CarDO.PrimaryKey );
qb.addWhere( CarDO.Dealer,       DealerDO.PrimaryKey );
qb.addWhere( DealerDO.Name,      "BOB",
QueryBuilder.CASE_INSENSITIVE_STARTS_WITH );
OwnerDO[] bobsCustomers = ownerQuery.getDOArray();
/*
* The code above uses a mix of setQueryXxx() methods
* and QueryBuilder.addWhere() methods.
* It produces this SQL query which JOINs the necessary tables:
*
*      SELECT Owner.* FROM Owner, CarOwner, Car, Dealer
*      WHERE  Owner.Age             >=   16
*      AND    Owner.oid             =    CarOwner.Owner
*      AND    CarOwner.Car          =    Car.oid
*      AND    Car.Dealer            =    Dealer.oid
*      AND    LOWER( Dealer.Name )  LIKE "bob%"
*
* For each column in your table,
* DODS creates a static RDBColumn member in your DO class.
* These members are used in the calls to addWhere().
*
* For each table (see any .sql file generated by DODS),
* DODS adds an 'oid' column as the primary key.
* So, OwnerDO.PrimaryKey is the RDBColumn member
* for the 'oid' column in the Owner table.
*
* QueryBuilder.CASE_INSENSITIVE_STARTS_WITH is a comparison operator
* which produces string comparison clauses using SQL's LIKE operator.
*/
```

Your familiarity with SQL will help you compose advanced queries with `QueryBuilder`.

## Using QueryBuilder without Query classes

In some cases you do not need DOs, you just need to read from the database. For instance, if you won't be saving any changes to data in a database table, you don't need DO objects in your query. Another is when you are generating a report, and only need a few columns. In these cases, you can use `QueryBuilder` without `Query` classes.

For example, to generate a report listing the license plate of each car, the dealer who originally sold the car, and every owner of the car, use the following code.

```
//create a QueryBuilder that will retrieve the desired fields
QueryBuilder qb = new QueryBuilder();
qb.select( CarDO.LicensePlate );
qb.select( DealerDO.Name );
qb.select( OwnerDO.Name );

// compose the necessary joins
qb.addWhere( OwnerDO.PrimaryKey, CarOwnerDO.Owner );
qb.addWhere( CarOwnerDO.Car,     CarDO.PrimaryKey );
qb.addWhere( CarDO.Dealer,       DealerDO.PrimaryKey );

// retrieve the values for the desired fields
RDBRow row;
while ( null != ( row = qb.getNextRow() ) )
   System.err.println(
      " Plate # =" + row.get( CarDO.LicensePlate ).getString() +
      " Dealer =" + row.get(  DealerDO.Name      ).getString() +
      " Owner =" + row.get(   OwnerDO.name       ).getString() );
```

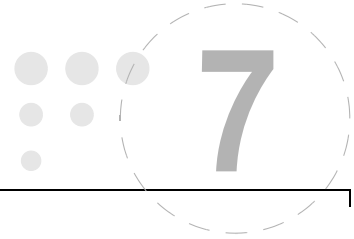
## Debugging queries using QueryBuilder

The `QueryBuilder` class has many `addWhere` methods to help you assemble complex queries. Calling the method `QueryBuilder.debug()` will print the assembled SQL before it is executed.

## Caching tables in memory

If you have a simple table with static data, you can load the entire table in to memory when your application starts to speed up data access. Specify that the DO class is cache-enabled. Your `Query` object will search the cache instead of querying the database.

**Note** Cached tables are not relational databases. You cannot run JOIN operations on cached tables. If you call `getQueryBuilder()`, your `Query` object will assume that you are setting up a complex query that the cache can not support, and will send the query to the database, ignoring the cache.



# Chapter 7

# Using InstantDB

This chapter introduces the InstantDB database and describes how to use it with Enhydra. For information on installing InstantDB, see the installation instructions on the CD.

For information on advanced topics, and complete reference documentation, see the online HTML documentation that was installed to the `doc` subdirectory of the directory in which you installed InstantDB.

## Introduction

InstantDB is a pure Java Relational Database Management System (RDBMS). It has a very small installation footprint: the core classes occupy less than 200 KB of space.

InstantDB supports many aspects of the SQL-92 entry-level standard, including:

- Joins
- Transactions
- Sub-selects
- Table aliasing
- Triggers
- Foreign keys

InstantDB also supports encrypted data storage and comes with a JDBC driver, so any Java application can use it, including Enhydra applications.

### Configuring your system

After you have installed InstantDB, as described in the installation instructions on the CD, be sure to put the InstantDB JAR files `idb.jar`, `jta-spec1_0_1.jar`, and `idbexmpl.jar` in your `CLASSPATH`. For example, if *<idb_root>* is the directory in which you installed InstantDB:

```
CLASSPATH=<idb_root>/Classes/idb.jar:<idb_root>/Classes/jta-spec1_0_1.jar:
      <idb_root>/Classes/idbexmpl.jar
export CLASSPATH
```

**Note** When using the Cygwin shell on Windows, you must use a backslash (\) and a semicolon (;) to separate entries. For example:

```
CLASSPATH=<idb_root>/Classes/idb.jar\;<idb_root>/Classes/jta-spec1_0_1.jar\;...
```

## Creating a new database

The easiest way to create an InstantDB database is to use the application called `ScriptTool`, which is included with InstantDB as a sample application. This application is described in greater detail in "ScriptTool" on page 126.

To create a new database with ScriptTool:

1. Create a directory for your database.

   This is not required, but is recommended to keep all the files together in one place. Make this the active (working) directory.

2. Create a database properties file in the working directory, and give it the extension `.prp`.

   The entries in this file are described in "Using properties files" on page 119. For an example of a database properties file, see `<InstantDB_root>/Examples/sample.prp`.

**Tip** You can usually just copy this file to a file with the name of your database and then alter it as needed.

3. Create a text file in the working directory that contains the SQL CREATE TABLE statements for your database and a reference to the properties file you created in step 2.

   The syntax of this input file is described in "ScriptTool" on page 126. For an example of a SQL input file, see `<InstantDB_root>/Examples/sql1.txt`.

4. Enter the following command in a shell window, where `<SQLFile>` is the name of the SQL file created in step 3:

   ```
   java org.enhydra.instantdb.ScriptTool <SQLFile>
   ```

   This command runs the Java application, `ScriptTool`, using `<SQLFile>` as input. `ScriptTool` creates three subdirectories:

   - `indexes` contains files with index and primary key information.
   - `system` contains files that define all the database metadata.
   - `tables` contains files with table information.

To confirm that the database was created, make sure these directories are there. You can also use the `DBBrowser` utility, as described in the following section.

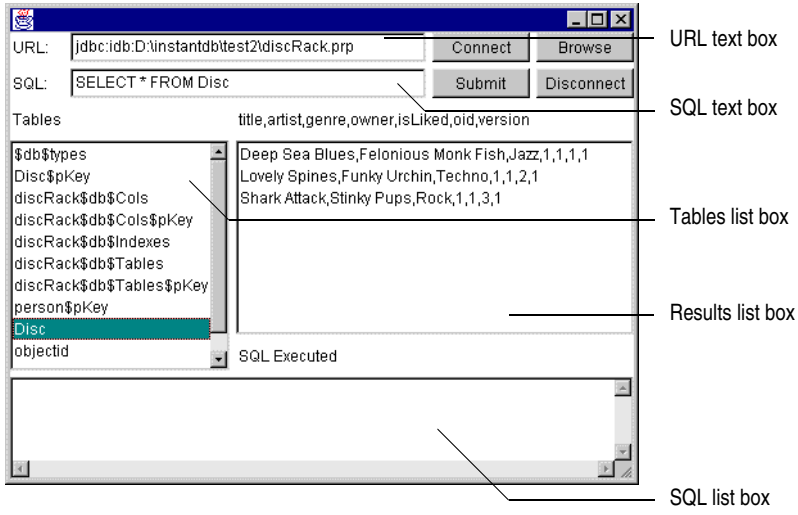
## Viewing a database

You can view the metadata and contents of an InstantDB database with the `DBBrowser` application.

To view the contents of a database, use the command:

```
java -Xms16m -Xmx32m org.enhydra.instantdb.DBBrowser
```

This command runs `DBBrowser`, the Java sample application that serves as the InstantDB database browser.




The Results list box initially displays the system properties such as the Java Virtual Machine (JVM) being used. To browse a database:

1. Click the Browse button.
2. In the File Selection dialog box, select a properties file (for example, `discRack.prp`).
3. Click Open.
4. To connect to the database, click the Connect button.

   After a brief pause, the Tables list box displays all the tables in the database, including the system tables and indexes.

To query a table, select the table in the Tables list box then click the Submit button. This constructs an SQL statement in the SQL text box, executes it, and displays the results in the Results list box. For example, if you choose the Disc table in the DiscRack database, then click Submit, you see "SELECT * FROM DISC" in the SQL text box, and then all the data in the Disc table in the Results list box.

## Using the InstantDB JDBC driver

InstantDB includes a JDBC driver that enables it to connect to Java applications.

### Database URLs

To identify an InstantDB database, use the following URL:

`jdbc:idb:`*`path`*

where *path* is the full path of the database properties file. The properties file tells InstantDB where to find or create the database tables, and specifies all the properties of the database, as described in "Using properties files" on page 119.

**Note** For backward-compatibility with older versions of InstantDB, you can also use URLs of the form:

`jdbc:idb=`*path*

### JDBC result sets

As indicated in the JDBC specification, InstantDB limits the number of open result sets to one per statement. As soon as InstantDB executes a new SQL statement, it closes any existing result set. To access multiple result sets simultaneously, use multiple statements.

It is important to close result sets explicitly, whenever possible. If the garbage collector tries to destroy an open result set, InstantDB's `finalize()` method tries to close the result set. This can lead to transaction access conflicts with the main thread in some virtual machines—which, in turn, can lead to deadlocks.

## Using InstantDB with Enhydra

You can configure an Enhydra application to use an InstantDB database with just a few steps. Because both Enhydra and InstantDB are pure Java applications, the entire application is portable to any system that has a standard JVM.

### General procedure

To use an InstantDB database with an Enhydra application named *appName*:

1. In the application configuration file `<`*appName*`>/<`*appName*`>.conf` (sometimes, by convention, `<`*appName*`>/<`*appName*`>.conf.in`), set the following line:

   ```
   DatabaseManager.DB.<database_id>.Connection.Url = "jdbc:idb:<propFile>.prp"
   ```

   where `<`*database_id*`>` is the database identifier used in the configuration file, and `<`*propFile*`>` is the full path to the database properties file.

2. In the same configuration file, identify the JDBC driver with the line:

   ```
   DatabaseManager.DB.database_id.JdbcDriver = "org.enhydra.instantdb.jdbc.idbDriver"
   ```

3. Add the path to `idb.jar` to the setting for `CLASSPATH` in the application's `start` script in `<`*appName*`>/start`.

**Note** On Windows systems, database files must be on the C drive due to limitations with the Cygnus tools.

After making these changes, rebuild the application, then test it by loading the appropriate URL in your browser.

## Running the DiscRack application with InstantDB

To run the Enhydra sample application, DiscRack, you must first create a DiscRack database in InstantDB and then configure the application to use InstantDB.

### Creating the DiscRack database

Create the DiscRack database as follows:

1. Create a new directory to contain the database; call it, for example, `discrack_idb`. Make this the working directory.
2. Copy the InstantDB properties file, `<enhydra_root>/DiscRack/discRack/data/discRack.prp`, to the new directory you just created.
3. Use the `ScriptTool` utility to create the new DiscRack database as described in "Creating a new database" on page 116.

   Enhydra comes with a file that contains the CREATE TABLE statements for the DiscRack database.
4. Enter the following command in a shell window:

   ```
   java org.enhydra.instantdb.ScriptTool <enhydra_root>/examples/DiscRack/
        discRack/data/create_tables.sql.instantDB
   ```

   This command reads the CREATE TABLE statements for the DiscRack database and creates the corresponding InstantDB database.

### Configuring DiscRack

To run the DiscRack application with InstantDB:

1. Modify the application configuration file and the `start` script as described in the previous section.
2. Start DiscRack as you normally would; for example, by entering this command in the `output` directory:

   ```
   ./start
   ```

   The application starts, and you can then access it from your Web browser. For more information, see *Getting Started with Lutris Enhydra*.

# Using properties files

An InstantDB properties file is a text file that defines an InstantDB database. When creating a database, InstantDB derives the name of the database from the name of the properties file. For example, a properties file named `foo.prp` creates a database called foo.

**Note** Renaming a properties file after a database has been created causes the database to become inaccessible.

The properties file includes this information:

- Database directories
- Tuning properties
- Logging and debugging properties
- Transaction and recovery properties
- Date, time, and currency properties
- String-handling properties

## Format

A properties file consists of a series of name/value pairs, separated by an equal sign (=), on separate lines. Lines beginning with an exclamation point (!) are comments. For example:

```
! Path where database tables are held.
tablePath=./tables
```

Property names are case-sensitive.

**Note** There should not be any spaces before or after the equal sign in a property assignment line.

## Database directories

The properties file contains entries that specify the directories where InstantDB will store database files. These files contain information such as system tables, indexes, and the actual database tables. Table 7.1 summarizes directory location properties.

**Note** On Windows, you can use either forward slashes (/) or double backslashes (\\) as the path delimiter. The highest degree of portability is provided when you use relative paths with forward slashes.

### Using paths relative to the properties file

You can specify paths relative to the properties file itself. To enable this option, include the entry:

```
relativeToProperties=1
```

This is the most portable scheme because if you move the database, you do not need to edit the properties file.

### Using absolute or relative paths

If the properties file does not contain the `relativeToProperties` entry, or if `relativeToProperties` has a value of zero (0), database directory properties can be either absolute or relative paths to the user directory. The user directory is specified by the system property `user.dir`, and is usually the directory in which the JVM is running. Changing `user.dir` does not change the user's current directory, it simply changes the property. Java does not provide a way to programmatically change the user directory at runtime.

You can also specify paths using system properties. To do this, precede the path with a dollar sign ($). For example, this entry sets tablePath to the runtime value of the system property user.dir:

```
tablePath=$user.dir
```

The following table shows the properties for the file directory-location entries.

**Table 7.1** Properties file directory-location entries

| Property | Required | Specifies location of |
|---|---|---|
| indexPath | No | Index tables. Defaults to tablePath if not specified. |
| partitionCount | No | Number of additional table partitions. |
| partition1<br>partition2<br>… | No | Absolute paths to alternative partitions. |
| systemPath | No | System tables, such as the table of all columns. Defaults to tablePath if not specified. |
| tablePath | Yes | Database tables. |
| tmpPath | No | Temporary tables, such as result sets. Defaults to tablePath if not specified. |

## Defining partitions

You can alter the locations for table files by defining partitions. A *partition* is a directory identified with a partition number. To use partitions, specify the total number of partitions with the entry partitionCount, and specify a directory for each partition. For example:

```
partitionCount=2
partition1=c:/users/petes/tables
partition2=d:/users/tom
```

This defines two partitions on different disk drives.

**Note** Partition directories are always absolute paths and must physically exist before InstantDB can create tables in them. InstantDB does not create the directories for you.

To create a table on a particular partition, use the syntax:

```
CREATE TABLE name ON PARTITION n.
```

For example, to create a new table on partition 1, use:

```
CREATE TABLE petesTable ON PARTITION 1 (int1 int).
```

This command creates the table petesTable in the directory c:\users\petes\tables.

## Tuning properties

Tuning properties affect how InstantDB caches data in memory. For very large databases, these properties can have a dramatic effect on performance. For more information, see the online documentation.

**Table 7.2** Tuning properties

| Property | Default value | Description |
|---|---|---|
| cacheAmount | 256 rows or 10% | How many rows in each column to cache.<br>If `cacheCondition` is `CACHE_ROWS`, the value is a number of rows; if `cacheCondition` is `CACHE_PERCENT`, the value is a percentage of the total number of rows in the table. |
| cacheCondition | CACHE_ROWS | If the value is `CACHE_ROWS`, `cacheAmount` is a number of rows. If the value is `CACHE_PERCENT`, `cacheAmount` is a percentage of the total number of rows in the table. |
| controlColCacheSize | 8192 | Specifies the size of the $$control system column. |
| fastUpdate | 0 | By default, InstantDB writes to the database after every INSERT, DELETE, or UPDATE is committed. A nonzero value specifies that InstantDB does not write changes to the database immediately. This can improve performance. |
| flushAfterCacheMisses | 128 | If the main readahead buffer is not used to retrieve data, this allows a thread to use it again. Useful when multiple threads are in READ UNCOMMITTED state.<br>If there are multiple read-only threads, lower this setting; if there is a single thread, leave the default value. |
| indexLoad | 5 | Percentage of free space in an index that must be present before the index reorganizes itself. |
| resultsSetCacheAmount | 100 | How many rows in each result set to cache.<br>If `resultsSetCache` is `CACHE_ROWS`, the value is the number of rows; if `resultsSetCache` is `CACHE_PERCENT`, the value is a percentage of the number of rows in the result set. |
| resultsSetCache | CACHE_ROWS | `cacheCondition` setting to use for result sets. |
| resultsOnDisk | 0 | By default, InstantDB holds result sets entirely in memory. A nonzero value specifies that result sets are saved to disk. |
| rowCacheSize | 20 | Number of rows to read into the disk lookahead buffer. Recommended value is between 128 and 256. |
| searchDeletes | 0 | By default, InstantDB does only a cursory search for deleted rows during UPDATE statements. Setting `searchDeletes=1` causes more detailed searches for deleted rows. This slows down UPDATE executions, but results in more compact tables. |
| singleRowCount | 8 | Allows tables to be read a single row at a time. Useful when multiple readers are doing full-table scans. |
| systemCacheSize | 100% | How many rows in each system table column to cache.<br>If `systemCacheCondition` is `CACHE_ROWS`, the value is a number of rows; if `systemCacheCondition` is `CACHE_PERCENT`, the value is a percentage of the total number of rows in the table. |

**Table 7.2** Tuning properties (continued)

| Property | Default value | Description |
|---|---|---|
| systemCacheCondition | CACHE_PERCENT | If the value is `CACHE_ROWS`, `systemCacheSize` is a number of rows. If the value is `CACHE_PERCENT`, `systemCacheSize` is a percentage of the total number of rows in the table. |
| systemRows | rowCacheSize | Number of rows of system tables to read into the disk lookahead buffer. Recommended value is between 128 and 256. |
| timerCheck | 5000 | Interval in milliseconds between checks for timed-out queries. |

## Logging and debugging properties

These properties affect where InstantDB sends trace output. For more information, see the online documentation.

**Table 7.3** Logging and debugging properties

| Property | Default value | Description |
|---|---|---|
| exportSQL | 0 | Nonzero means include SQL statements in the export file. |
| traceConsole | 0 | Nonzero means trace output also directed to console. |
| traceFile | N/A | Relative or absolute path where exporting and trace information is sent. |
| traceLevel | 0 | Bitmap of various items that can be traced. |

## Transaction and recovery properties

InstantDB supports ANSI standard transactions. It executes all SQL statements within a transaction. By default, auto-commit is enabled so there is no need to perform explicit commits after every statement.

Two properties control transaction processing:

- transLevel controls how the journal file is used.
- transImports determines the number of rows imported in an IMPORT statement before the current transaction is committed.

### Recovery

If InstantDB opens a database and finds that data is missing, it cannot tell whether another JVM is using the database or whether it has failed to shut down cleanly and needs to recover the data. By default, InstantDB asks whether it should perform recovery. You can alter InstantDB's behavior with the recoveryPolicy database property.

- Setting `recoveryPolicy=1` causes InstantDB to perform automatic recovery when it finds a corrupt database.

  This is useful when the database is being used as a server and automated failover recovery is required.

- Setting `recoveryPolicy=2` causes InstantDB to issue a prompt when it encounters a potentially corrupt database.

  This is useful during development when interactive database monitoring might accidentally lead to corruption.

- Setting `recoveryPolicy=0` causes InstantDB to refuse to open the database if data appears to be missing

This table lists additional transaction and recovery properties.

**Table 7.4** Transaction and recovery properties

| Property | Default value | Description |
|---|---|---|
| recoveryPolicy | 2 | 0: Do not perform recovery.<br>1: Perform automatic recovery.<br>2: Prompt the user via standard input (stdin). |
| transImports | 100 | When doing an import, defines the number of rows imported before the transaction is committed. Recommended value 8192. |
| transLevel | 1 | Sets the level of transaction journalling. |

**Note** InstantDB's recovery policy makes the most sense in the context of its use by multiple JVMs, which is beyond the scope of this document. For information on multiple JVMs using InstantDB, see the online documentation.

## Date, time, and currency properties

These properties control how InstantDB handles dates, times, and currency values. For more information, see "DATE data types" on page 135 and "CURRENCY type" on page 137.

**Table 7.5** Date, time, and currency properties

| Property | Default value | Description |
|---|---|---|
| currencyDecimal | 2 | Number of digits after decimal point in currency outputs. |
| currencySymbol | $ | Currency symbol used in currency outputs. |
| dateFormat | "yyyy-mm-dd" | Default format for date columns. |
| milleniumBoundary | 0 | If set, all two-digit dates less than its value are interpreted as 21st century dates. |
| nowMeansTime | 0 | Set to 1 causes the date string "now" to store a full timestamp. Default is to store only the date for fields with no hour in the format string. |

## String-handling properties

These properties control InstantDB's string handling. For more information, see "Strings" on page 139.

**Table 7.6** String handling properties

| Property | Default value | Description |
|---|---|---|
| altStringHashing | 0 | If set to 1, string hashes use the JDK `Object.hashCode()` function. By default, uses InstantDB's string hashing. |
| likeIgnoreCase | 0 | Set to 1 to cause LIKE clauses to always perform case-insensitive comparisons. |
| prepareIgnoresEscapes | 0 | Set this value to 1 (one) if you want `PreparedStatement.setString()` to ignore `\` (backslash) characters when processing string constants. When set, InstantDB does not attempt to interpret `\` (backslash) as the start of an escape sequence. |
| strictLiterals | 0 | Same as SET LITERAL STRICT_ON. Prevents string literals from being interpreted as column names or numbers. |

# Using the InstantDB sample applications

InstantDB comes with some sample Java applications. These are intended to introduce users to JDBC programming with InstantDB. Because InstantDB includes the source code of the sample applications, you can modify them to suit your own purposes (subject to the Enhydra Public License).

Some of these applications are useful utility programs that let you perform tasks such as database creation, browsing, and data manipulation.

**Note** To use the sample applications, your `CLASSPATH` must include `idbexmpl.jar`, the Java archive that contains the example application classes.

The sample applications include:

- `commsql`, a simple command-line utility that processes SQL statements interactively
- `ScriptTool`, a utility that accepts SQL scripts
- `dump`, which displays the contents of InstantDB system files
- `JDBCAppl` and `DBBrowser`, which let you browse and query a database from within an applet (`JDBCAppl`) or a Java application (`DBBrowser`)
- `SQLBuilder`, an application that uses Java reflection to read metadata

## commsql

The `commsql` application accepts SQL statements from the command line and executes them interactively. To run `commsql`, enter this command in a shell window:

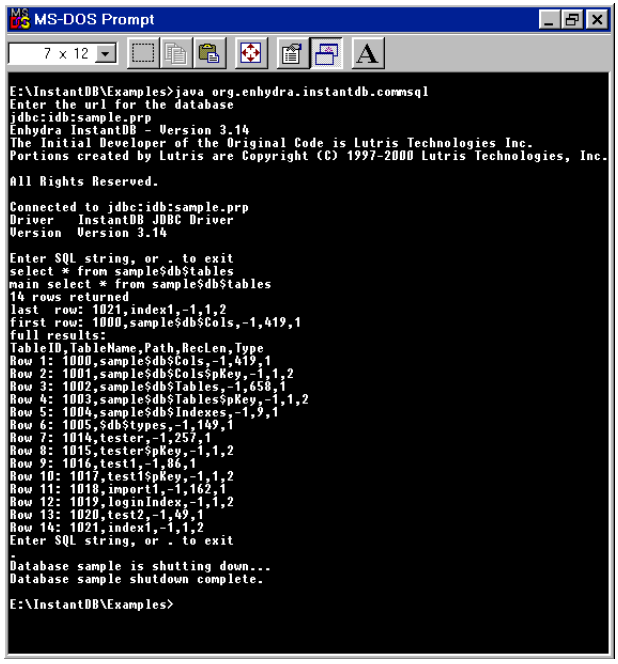
```
java org.enhydra.instantdb.commsql
```

The program initially loads the InstantDB JDBC driver and then prompts you for a URL to connect to. Enter the URL of an InstantDB properties file. For the URL syntax, see "Database URLs" on page 117. InstantDB then opens the database corresponding to this properties file. If the file does not exist, InstantDB creates it. You will then see the prompt:

```
Enter SQL string, or . to exit
```

You can enter any valid InstantDB SQL commands. If the command produces a result set, `commsql` displays the results in the shell window.

When you want to exit the program, type a period (.) on a line by itself.




## ScriptTool

The `ScriptTool` application was the original InstantDB sample program, called `sample`. Over time, however, it has become considerably more complex and is now a useful utility program for running SQL scripts.

`ScriptTool` runs an input file containing commands that it executes. The format of the input file is described in the next section. If you invoke ScriptTool without any

command-line arguments, it opens the file `sql1.txt` by default. This file is provided in the `/Examples` directory and contains sample commands to be executed.

To specify an input file on the command line, use this syntax:

```
java org.enhydra.instantdb.ScriptTool <SQLFile>
```

where *SQLFile* is the name of the input file.

## Format of input file

A `ScriptTool` input file is an ASCII text file containing a series of commands that `ScriptTool` executes in order. Every command must begin with a command letter, followed by either a SQL statement or further command information. Every command must be terminated with a semicolon (;).

Each input file must begin with a `d` command to load at least one JDBC driver. Normally, an `o` command to open a database would follow immediately thereafter. Table 7.7 describes all valid commands for an input file.

**Table 7.7** Commands for ScriptTool utility

| Command | Description |
|---|---|
| c *stmt* | Execute command *stmt*. For example:<br>`c close;`<br>See Table 7.8 on page 128. |
| d *driver* | Load JDBC driver, *driver*. For example:<br>`d org.enhydra.instantdb.jdbc.idbDriver;` |
| e *SQLstmt* | Execute *SQLstmt* with no result set. For example:<br>`e UPDATE table1 SET col1=col1+1;` |
| i *tableName* | Get index information for *tableName*. |
| l *tableName colName* | Display last values inserted into table *tableName* or column *colName*. |
| o *url* | Open URL *url*. For example:<br>`o jdbc:idb=sample.prp;` |
| p *SQLstmt* | Create a prepared statement, *SQLstmt*. For example:<br>`SELECT * FROM ?;` |
| q *SQLquery* | Execute query *SQLquery* that returns a result set. |
| r *count* {<br>*cmds*<br>} | Loop that executes the commands, *cmds*, in brackets *count* times. |
| m [n \| p \| f \| l \| i \| c \| b \| t] | Move within scrollable result set:<br>• n: next row<br>• p: previous row<br>• f: first row<br>• l: last row<br>• c: current row<br>• i: move to the insert row<br>• b: move before the first row<br>• t: move after the last row |
| m a *row* | Move to absolute row number *row* in a scrollable result set. |

**Table 7.7** Commands for ScriptTool utility (continued)

| Command | Description |
|---|---|
| m r *numRows* | Move a number of rows *numRows* relative to current position in a scrollable result set. |
| s *param1, param2,...* | Set parameters on a prepared statement and execute.<br>Optionally precede each parameter with a type specifier enclosed in percent signs (%). The following types are recognized:<br>• `%asciiStream%`<br>• `%binaryStream%`<br>• `%timestamp%`<br>• `%boolean%`<br>For example, this command calls `preparedStatement.setTimestamp()`. If no type specifier is included, a `setString()` is performed.<br>`s %timestamp%2000-01-14 10:52:52.12345678;` |
| t *script* | Start off a new thread with its own input script. |
| u *colName value* | Update a row in the table underlying a result set. |

The `c` command is a "catch all" for performing various miscellaneous tasks. It must be followed by one of the statements listed in Table 7.8.

**Table 7.8** Miscellaneous ScriptTool statements

| Command | Description |
|---|---|
| autocommit on | Turns autocommit mode on. |
| autocommit off | Turns autocommit mode off. All further transactions have to be explicitly committed or rolled back. |
| batch mode | Further statements and prepared statements get batched together. |
| break | A useful "no-op" command. By setting a breakpoint on the line of code in `ScriptTool.java` that handles this, you can get a script to break at any point by simply placing this command wherever you want the break to occur. |
| cancel row updates | Cancel updates to a row. |
| close | Close the current connection. |
| commit | Commit the current transaction using the JDBC `Connection.commit()` method. |
| create test object | Creates a small test object to test saving objects as blobs. |
| delete row | Delete a row in the table underlying a result set. |
| execute batch | Executes batched commands. |
| exit | Exit the program immediately. |
| export all | Exports all tables to CSV text files. |

**Table 7.8** Miscellaneous ScriptTool statements (continued)

| Command | Description |
|---|---|
| get columns *tableNamePattern columnNamePattern* | Retrieves information about the columns with names that match the string pattern *columnNamePattern* in tables with names that match the string pattern *tableNamePattern*. The percent sign (%) can be used for matching zero or more characters, and an underscore (_) can be used for matching a single character. For example, to get information about all columns with three-letter names that start with "co" in tables with names that start with "t" and end with "e," the command would be constructed as follows:<br>`c get columns t%e co_;`<br>Only metadata entries matching the search pattern are returned. If a search pattern argument is set to a null ref, that argument's criteria will be dropped from the search. |
| get imported_keys *tableName* | Retrieves and displays foreign keys referenced by the table *tableName*. |
| get exported_keys *tableName* | Retrieves and displays columns in the table *tableName* that are referenced as foreign keys. |
| get crossReference *primaryTable foreignTable* | Retrieves and displays the foreign keys in the table *foreignTable* that reference keys in the table *primaryTable*. |
| import all | Import all tables from CSV text files. |
| insert row | Insert a row in the table underlying a result set. |
| rollback | Rollback the current transaction using the JDBC `Connection.rollback()` method. |
| set global connection | Make the current `Connection` object available to other threads. |
| set global prepared statement | Make the current prepared statement the global statement that other threads can pick up. |
| set isolation READ_UNCOMMITTED | Set the transaction isolation level to `Connection.TRANSACTION_READ_UNCOMMITTED`. |
| set isolation SERIALIZABLE | Set the transaction isolation level to `Connection.TRANSACTION_SERIALIZABLE`. |
| set timeout *n* | Set the timeout on queries to *n* seconds. (ScriptTool defaults to 300). |
| show metadata | Displays the type info and table list result sets. |
| time | The time in milliseconds that the previous command took to run. |
| toggle result set close | Close a result set after displaying results. |
| toggle RSMD | Toggles display of result set metadata after queries. |
| update row | Update a row in the table underlying a result set. |
| use global prepared statement | Use the current "global" prepared statement rather than the thread's own prepared statement. |
| use global connection | Use the global `Connection` object rather than setting up a separate connection for this thread. |
| wait for children | Wait for all child threads to complete. |

### Example

A sample script is shown below:

```
; First load the JDBC driver and open a database.
d jdbc.idbDriver;
o jdbc:idb=sample.prp;

; start off a couple of threads reading from some other scripts
t sql2.txt;
t sql3.txt;

; Record all results
e SET EXPORT "export0.txt" FIXEDLENGTH COLNAMEHEADER ROWNUMBERS
      CONTROLCOL SUMMARYHEADER;

; Create the table and its index
e DROP TABLE tester;
e CREATE TABLE tester (
      id              int PRIMARY KEY,
      fullName        CHAR(30),
      email           CHAR(60),
      login           CHAR(8),
      password        CHAR(20) );
e CREATE INDEX loginIndex ON tester ( login );

; put some initial data in the table
e INSERT INTO tester VALUES (1,"Alice","Alice@isp","Alice","Alice");
r 5 {;
    e INSERT INTO tester VALUES (2,"Bob","Bob@isp","Bob","Bob");
};
q SELECT * FROM tester;

; modify the table contents
p UPDATE tester SET fullName=?,email=?, login=?, password=? WHERE id=?;
s 'pete@some','pete','pete','pete',2;
q SELECT * FROM tester;

c close;
```

## dump

The `dump` application displays (dumps) the contents of some of InstantDB's system files. You can display the contents of either the system table containing column information, the system table containing table information, or the journal file.

You must run `dump` from the directory that contains the system file being dumped. By default, `dump` displays to the standard output device (standard out), but you can redirect to any file if required.

The command-line syntax is as follows:

```
java org.enhydra.instantdb.dump dbName {t | c tableID | j}
```

where `dbName` is the name of the database.

- The `t` flag, when used alone, displays the content of the tables system file (*dbName*`$db$tables`).

- The `c` flag displays the columns system file (*dbName*`$db$cols`) for the table with that `ID` number.

  If `tableID` is 0, it displays information for all tables. To determine the `ID` number for a table, display the contents of the table *dbName*`$db$tables`.

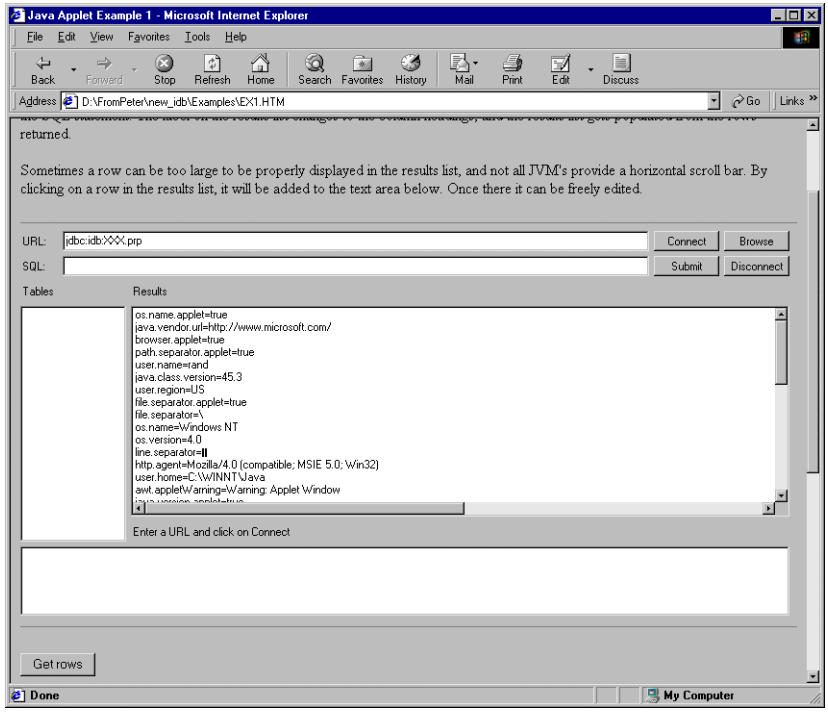
- The `j` flag displays the contents of the journal file.

## JDBCAppl and DBBrowser

The `JDBCAppl` application illustrates how to access InstantDB from a Java applet. Open the HTML file `/Examples/ex1.htm` in your Web browser to run `JDBCAppl`. This file also includes instructions for using `JDBCAppl`.

`JDBCAppl` tries to load the RmiJdbc SQL driver and the JDBC-ODBC bridge, in addition to the InstantDB JDBC driver. It ignores any errors if any of these fail to load.

This figure shows JDBCAppl, as it appears in Microsoft Internet Explorer.




Also included in `ex1.htm` is a piece of JavaScript that illustrates how a Java applet's public methods and properties can be accessed using the JavaScript object model.

`DBBrowser` provides a main method and frame for `JDBCAppl`, so you can run it as a standalone application instead of an applet.

### Applet security and JavaScript compatibility

To get the JDBCAppl applet to work properly, there are two issues you need to consider: applet security and JavaScript compatibility.

InstantDB needs permission to read from and write to the disk. Internet Explorer allows an applet to perform local file access if you add the path for the InstantDB and JDBCAppl classes to the CLASSPATH in the registry:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Java VM\Classpath
```

Netscape Communicator does not normally allow an applet to write to or read from the file system. The proper way to get the applet to work with Communicator is to sign it with a digital certificate that is either trusted, or that has itself been signed by a trusted certification authority. For details, see Netscape's Web site: http://home.netscape.com.

As a workaround, you can include the following import statement in the applet:

```
import netscape.security.PrivilegeManager;
```

Then, add these lines to the method to request the additional privileges:

```
PrivilegeManager.enablePrivilege("UniversalPropertyRead");
PrivilegeManager.enablePrivilege("UniversalPropertyWrite");
PrivilegeManager.enablePrivilege("UniversalFileRead");
PrivilegeManager.enablePrivilege("UniversalFileWrite");
PrivilegeManager.enablePrivilege("UniversalFileDelete");
```

The file JDBCAppl.nets in the Examples directory is a version of JDBAppl.java that already contains the above changes.

Finally, find Communicator's prefs.js file (on Windows, typically in C:\program files\netscape\users\default), and add this line:

```
user_pref("signed.applets.codebase_principal_support", true);
```

When you compile the applet, make sure the following line (or the path to your Netscape Java classes) is included in your CLASSPATH:

```
c:\progra~1 \netscape\communicator\program\java\classes\java40.jar
```

## SQLBuilder

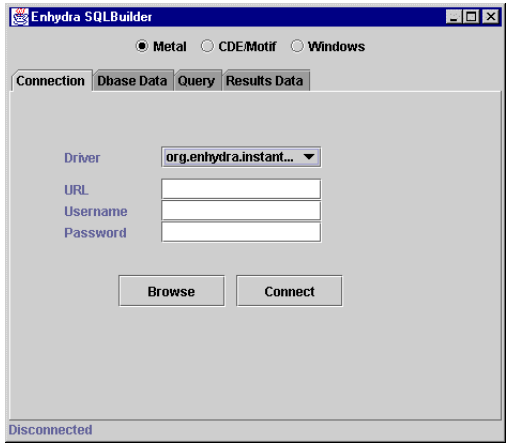
SQLBuilder is a standalone Java application that you can use to view database tables, records, and fields. It's similar to JDBCAppl except:

- It retrieves database metadata and result sets using Java reflection.
- It lets you specify a row limit on queries.
- It lets you build simple queries by selecting columns.

To run SQLBuilder, enter the following command in a shell window:

```
java org.enhydra.instantdb.SQLBuilder.SQLBuilder
```

The SQLBuilder window appears:




`SQLBuilder` has two main user interface areas:

- A group of radio buttons you can use to select the application's "look and feel."
- The main SQLBuilder interface, which consists of four tabbed panes: Connection, Dbase Data, Query, and Results Data.

The Connection pane lets you connect to a database. To connect, choose a JDBC driver from the pull-down list, and enter a URL for the database. To connect to an InstantDB database, click the Browse button and select a properties file. Some databases may also require a user name and password, though InstantDB does not. Next, click the Connect button. The status changes first to "Connecting" and then to "Connected." The Connect button then changes to a Disconnect button.

The Dbase Data pane displays the database metadata for the database.

The Query pane provides a pull-down list of the available tables. For the currently selected table, a second pull-down list shows the available columns.

- Selecting a table causes its columns to be entered in the Columns pull-down list box. It also sets up a default query to select everything from the table.
- Selecting a column causes the query to change to that column only.
- Selecting more columns causes them to be added, one by one, to the query.

The Row Limit box lets you limit the number of rows returned by the query, and the Submit button submits the currently entered query. You can also enter any other query in the Query text box.

When `SQLBuilder` is connected to an InstantDB database, the contents of the results table represent the query results. This takes advantage of InstantDB's result set navigation API calls. For other databases, the table is always 1000 rows long, and you can only move forward through the result set.

The Result Data pane shows result set metadata for the current result set. This is mostly column-based data. To display metadata for a column, choose a column from the pull-down list.

# InstantDB data types

Because InstantDB is a pure Java database, and its only interface is JDBC, its data types are based on Java's data types. The available data types are listed in the following table.

**Table 7.9** InstantDB data types

| Type | Synonyms | Comment |
|---|---|---|
| BYTE | TINYINT, BIT, BOOLEAN | 1 byte signed integer |
| INT | INTEGER, SHORT, SMALLINT | 4 byte signed integer |
| LONG | | 8 byte signed integer |
| DECIMAL | NUMERIC | Varies with specification |
| CURRENCY | | Descended from LONG |
| DOUBLE | | 8 byte floating point |
| FLOAT | | 4 byte floating point |
| DATE | DATETIME, TIMESTAMP, TIME | Descended from LONG |
| CHAR | VARCHAR, VARCHAR2 | Variable-length string |
| SMALLCHAR | | Single byte ASCII variable-length string |
| BINARY | VARBINARY, LONGVARBINARY, OLE, LONGCHAR, TEXT, IMAGE | Binary data or long text |

## Numeric types

InstantDB's numeric data types follow the same rules as their Java equivalents, with one exception: a NULL value is represented by the minimum negative number for BYTE, INT, and LONG types. Any attempt to use the minimum value with these types results in a `SQLException`.

**Note** This does not apply to FLOAT or DOUBLE where "not a number" (NaN) is used to represent NULL.

### Auto-incrementing

There is no separate COUNTER type. Instead, the column condition `AUTO INCREMENT` can be added to a column during table creation. For example:

```
CREATE TABLE table1 (int1 int AUTO INCREMENT)
```

The auto-increment condition can be applied to both INT and LONG column types.

It is often convenient to be able to turn AUTO INCREMENT off (for example, during IMPORT operations). You can do this with the following SQL statement:

```
SET table1 AUTO INCREMENT OFF
```

To turn AUTO INCREMENT back on:

```
SET table1 AUTO INCREMENT ON
```

By default, AUTO INCREMENT integer (int) columns start at 1,000; long columns start at 10,000,000,000; and all values are incremented by one. You can change the default with the command:

```
SET INCREMENT_BASE {<base>|MAX} ON {<table>.<column> | ALL}
```

For example, to set the initial value to 100, use the command:

```
SET INCREMENT_BASE 100 ON mytable.mycolumn
```

You may sometimes need to find a column's maximum value and then allocate auto-increment values above that—which is particularly useful if a table has been imported from an external source. For example:

```
SET INCREMENT_BASE MAX ON mytable.mycolumn
```

If you have imported a large number of tables, it might be convenient to tell the database to set all the auto-increment columns to be one more than their maximum values. You can do this with:

```
SET INCREMENT_BASE MAX ON ALL
```

It is not currently possible to alter the increment step.

## Decimal and numeric types

InstantDB supports fixed-precision arithmetic using the CURRENCY type. However, for compatibility with other SQL implementations, it also accepts the DECIMAL and numeric data types.

DECIMAL and numeric are not fixed-precision data types in InstantDB. Instead they are mapped to INT, LONG, or DOUBLE types, as described in Table 7.10.

**Table 7.10** Numeric data type mapping

| Precision | Scale | Mapped type |
|---|---|---|
| none | none | LONG |
| any | > 0 | DOUBLE |
| 1 to 9 | none or 0 | INT |
| > 9 | none or 0 | LONG |

## DATE data types

As in Java, the DATE type is stored as a `long` value. InstantDB uses its own date conversion methods to translate date and timestamps into `long` values.

## Formatting dates

InstantDB's default format for dates is "yyyy-mm-dd" (for example, "2000-12-24"). InstantDB provides a nonstandard SQL extension to control how dates are formatted for output. Use the following syntax:

```
SET DATE FORMAT format
```

where *format* can include any of the strings listed in Table 7.11, separated by any character. Typically a hyphen (-) or forward slash (/) is used for the date separator.

Once you set the date format, all tables that are created use this date format and queries against these tables use this format to represent dates. The SET DATE FORMAT command is connection-specific; it affects only those tables created on the current database connection. For example, this statement:

```
SET DATE FORMAT "dd/mm/yyyy"
```

sets the date format so dates have the form 16/11/1997.

Date values can also include the time. To include time information, add time format letters as described in Table 7.11. For example, this statement sets the date format so dates have the form 13:12:58.626 16/11/1997:

```
SET DATE FORMAT "hh:nn:ss.lll dd/mm/yyyy"
```

The following table shows date and time format strings.

**Table 7.11** Date and time format strings

| Format string | Description | Example |
|---|---|---|
| yyyy | Four digit year | 2001 |
| yy | Two digit year | 67 |
| mmm | Three letter month | Jan |
| mm | Two digit month | 11 |
| dd | Day | 31 |
| hh | Hour | 12 |
| nn | Minute | 59 |
| ss | Second | 59 |
| lll | Milliseconds | 999 |

## Timestamps

When InstantDB inserts values into a DATE column, it also accepts the string "NOW" or "now" to represent the current date. This is known as a *timestamp*. A timestamp does not include the time unless specified by the column format. To include the full time in addition to the date in a timestamp, use the string "current time."

**Note** InstantDB does not store nano-second values, only milliseconds. Consequently, when you insert a `java.util.Timestamp` value into the database, there may be a rounding error at the millisecond level.

If you insert a literal timestamp into a date column, the full timestamp is inserted, regardless of the date format. Time zone differences can also result in unexpected

values in a date column. To check what is really in a date column, try a command like:

```
SELECT TO_DATE(date_COL, "hh:nn:ss dd:mm:yyyy")
```

### Date functions

Table 7.12 describes date functions available with InstantDB.

**Table 7.12** Date functions

| Name | Parameters | Example |
|---|---|---|
| TO_DATE(*date*, *fmtStr*) | Returns *date* formatted according to format string *fmtStr*. | `TO_DATE(date1, "hh:nn:ss dd:mm:yyyy")` |
| TO_NUMBER(*date*, *field*) | Returns the numeric value of the *date*. The fields are as defined in java.util.Calendar. | `TO_NUMBER(date1, 10)` |

### Interpreting two-digit dates

By default, InstantDB interprets two-digit years as being in the 20th century. To specify a 21st century year, you need to use a four-digit year. For many applications, however, this is insufficient. To address this, InstantDB lets you set the property milleniumBoundary in the properties file. When this property is set, it allows two-digit dates less than milleniumBoundary to be interpreted as 21st century dates.

For example, if milleniumBoundary is set to 40, all two-digit dates in the range 00..39 are interpreted as 2000 through 2039, respectively, and all two-digit dates in the range 40..99 are interpreted as 1940 through 1999, respectively.

## CURRENCY type

For CURRENCY columns, the default currency symbol is the dollar sign ($), and the default number of digits after the decimal point is 2. A nonstandard extension lets you change the format of currency data:

```
SET CURRENCY {SYMBOL symbol|DECIMAL numDigits}
```

where *symbol* is the currency symbol (for example, $), and *numDigits* is the number of digits to display after the decimal.

A CURRENCY column is always created with the current currency format. Once a column is created, it retains those settings, even if new SET CURRENCY statements change the global values.

When inserting currency values, you can optionally include the currency symbol. If you do this, you must quote the value. For example:

```
INSERT INTO curtable VALUES (200.00)
```

and

```
INSERT INTO curtable VALUES ('$200.00')
```

are both equivalent (assuming the default global settings).

Currency values are held as appropriately-scaled Java LONG values. This means, for example, that the value $300.00 is actually held as 30,000 cents. Where InstantDB can obtain a context for evaluating a currency constant, it applies suitable formatting rules. Thus, the following statements all produce identical results:

```
SELECT * FROM mytable WHERE cost < 30000
SELECT * FROM mytable WHERE cost < 300.00
SELECT * FROM mytable WHERE cost < '$300.00'
```

For both DATE and CURRENCY types, the formatting information is held on a per-connection basis. This means that when you perform a SET DATE or SET CURRENCY command, the changes affect only the current connection.

When a CREATE TABLE is subsequently performed, the format settings in effect for the current connection are stored with any date or currency columns that get created. The formatting options do not affect how values are actually stored, only how they are displayed, and in the case of currency columns, how values must be formatted for input.

You cannot have different currency format settings within the same table. For example, you can't have display formats for U.S. dollars and German Marks in the same table. Similarly, you cannot have two date columns in a table if one displays only the date and the other displays only the time.

When a result set is created, the formatting options for any columns in the result set are dependent on the columns from which they were sourced. They are not dependent on the current connection's settings.

## BINARY type

Declaring a column to be of the BINARY type (or one of its equivalents) allows it to hold an arbitrary array of bytes. The JDBC driver returns such objects as `byte[]`.

There are several ways you can insert data into a binary column:

- Use a string that identifies a file. InstantDB reads the contents of the file as the binary data. For example:

  ```
  INSERT INTO blobtable VALUES ("c:\example\binary.dat")
  ```

- Use a string containing a sequence of integers. For example:

  ```
  INSERT INTO blobtable VALUES ("0x10, 0x20, 48, 64")
  ```

- Use an explicit text string. For example:

  ```
  INSERT INTO blobtable VALUES ("hello world")
  ```

  If you use the TEXT or LONGCHAR pseudonyms for binary columns, InstantDB assumes that you want only literal text in the column; it does not attempt to interpret the data as numbers or file names.

- Use a single hexadecimal number. For example:

  ```
  INSERT INTO blobtable VALUES (0001021B1C1D1E1F)
  ```

- Use the `PreparedStatement.setBytes()` and `setObject()` methods.

**Note** Be careful when you use the sequence of integers technique. The resulting values must fall into the normal Java byte range (–128..127). If the resulting value is outside this range, it is interpreted as a "short" and saved as two bytes. If the value is outside the short range, it is interpreted as an "int"; if it is outside the int range, it is interpreted as a "long."

The table containing the binary column only holds a pointer to the actual binary data. The data is held in a separate "blob" file. InstantDB recognizes two types of binary data as special—strings and Java objects. Strings are always returned as Java string objects. Objects saved using the `PreparedStatement.setObject()` method are returned in their native object format. Such objects must implement the `java.io.Serializable` interface in order to be saved as binary data.

Blobs are cached just like any other data, which means you should be careful when handling large blobs. You might want to explicitly declare blob columns as CACHE 0 ROWS.

## Strings

InstantDB handles strings with the CHAR(n) data type. Strings can be delimited by either a single quotation mark (') or a double quotation mark ("). Thus, `'a string'` and `"a string"` are equivalent. Within a string, the other quotation character can be used freely (for example, `"Here's a string with a quote"`). The delimiter can be included in a string by using a backslash (\) as an escape. For example:

```
"He said \"How's That\"".
```

Java recognizes the backslash (\) as an escape character, so within a Java program you need to escape the backslash, and you also need to escape the double quotes. Therefore, you would write the previous example as:

```
"He said \\\"How's That\\\"".
```

You can also include horizontal tabs and newlines in strings by using the standard tab (`\t`) and line feed (`\n`) characters.

### Case-insensitive comparisons

By default, string comparison in SQL is case-sensitive. InstantDB supports the nonstandard SQL extension IGNORE CASE in LIKE clauses to make string comparison case-insensitive. For example:

```
SELECT * FROM mytable WHERE name LIKE "A%" IGNORE CASE
```

This query matches all names beginning with uppercase `A` or lowercase `a`.

You can make all string comparisons that use LIKE case-insensitive by including the following line in the database properties file:

```
likeIgnoreCase=1
```

### Using SMALLCHAR

InstantDB uses the Java standard char and `String` types for string data. As with all Java character-based data, each character is represented as a two-byte Unicode

character. Java I/O and internationalization methods automatically handle translation to and from local single-byte character sets. Thus, InstantDB can handle a large number of character sets easily. However, all string data, whether in memory or on disk, requires two bytes to represent each character. This is typical of Java applications.

Despite the very high capacities now available in terms of both disk size and memory capacities, there are some environments (such as hand-held computers, personal organizers, and mobile phones) where compactness is more important than internationalization. In such environments, use InstantDB's SMALLCHAR data type.

The SMALLCHAR data type holds only the least significant byte of the corresponding Unicode character. This corresponds to the normal ASCII character set and can therefore represent the normal range of Latin character sets. You can use SMALLCHAR columns instead of CHAR or VARCHAR columns, if you do not need to represent non-Latin characters.

## String literals

Some development environments automatically generate SQL statements in which column names (as well as string literals) are quoted. For example:

```
SELECT * FROM mytable WHERE "mycol" = "abcd"
```

Notice both the column name and the literal value are quoted. By default, InstantDB accepts this and searches the column `mycol` for the value "abcd." However, this can be ambiguous. For example, should

```
SELECT * FROM mytable WHERE "myothercol" = "mycol"
```

be interpreted as a join on the two columns, or a search for the text "mycol" in "myothercol?"

To dispense with such ambiguities, you can modify InstantDB's default behavior with the property strictLiterals in the database properties file. To make InstantDB interpret all quoted strings as string literals, use this setting:

```
strictLiterals=1
```

**Note** The only exceptions to this are date strings, which can usually be interpreted from context.

You can also use the following SQL statement to change the interpretation of string literals:

```
SET LITERALS [STRICT_ON|STRICT_OFF]
```

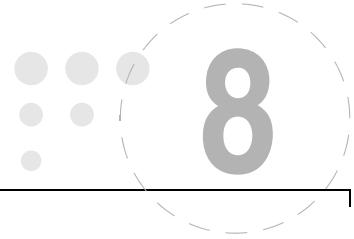
Any such change applies to all active connections, not just the issuing connection. In the previous statement, STRICT_OFF is the default.

## String functions

Table 7.13 describes InstantDB's string functions.

**Table 7.13** String functions

| Name | Description | Example |
|---|---|---|
| UPPER(*string*) | Returns *string* converted to uppercase. | `UPPER("abcd")` |
| LOWER(*string*) | Returns *string* converted to lowercase. | `LOWER("ABCD")` |
| SUBSTR(*string*, *startPos* [, *len*]) | Returns substring of *string*, starting at *startPos*, optionally of length *len*.<br>If *len* is not specified, substring extends to end of *string*. | `SUBSTR("ABCD",1,3)` |
| LENGTH(*string*) | Returns the length of *string*. | `LENGTH("ABCD")` |



# Chapter 8

# Using PostgreSQL

This chapter introduces the PostgreSQL database and describes how to use it with Enhydra. For information on installing PostgreSQL, see the top-level index.html file on your product CD.

## Introduction

PostgreSQL is an open-source database server that supports the SQL-92/SQL-3 language standards, transaction integrity, and type extensibility. It is a descendant of Postgres, a pioneering object-relational database developed at UC Berkeley.

### Features

PostgreSQL, in addition to including standard relational database structures that have primitive data types—such as floating-point numbers, integers, character strings, money, and dates—also incorporates some core object-oriented concepts:

- Classes
- Inheritance
- Types
- Functions

Because it contains both relational and object-oriented features, PostgreSQL is referred to as an *object-relational database*. The object-oriented features let users extend the system to suit their application needs. For increased flexibility, PostgreSQL also supports constraints, triggers, rules, and transactional integrity.

### Where to find PostgreSQL documentation

The RPM installation program installs documentation files in:

`/usr/doc/postgresql-<version number>`

The documentation includes README files, a user guide, and a programmer's guide in HTML format.

You can find additional information on the PostgreSQL Web site at: http://www.postgresql.org.

For information on the RPM installation process, see the top-level index.html file on your product CD.

# Using PostgreSQL with Enhydra

To enable the Enhydra database manager to access PostgreSQL, you must add the PostgreSQL JDBC driver file to your `CLASSPATH`. The best way to do this is to edit your application's `start` script and add the line:

```
CLASSPATH=/usr/lib/pgsql/jdbc<version number>.jar
```

By default, the PostgreSQL RPM installation puts the JDBC JAR file, `jdbc<version number>.jar`, in the directory `/usr/lib/pgsql`. If you move the JAR file, remember to change the reference in the `start` script.

**Important** Be sure to edit the `start` script file in `<appName>/<appName>`, not `<appName>/output`, which is overwritten every time you build the application.

## Using DODS

As described in *Getting Started with Lutris Enhydra*, the Data Object Design Studio (DODS) requires special columns named OID and VERSION in each table. However, OID is a reserved word in PostgreSQL. Fortunately, the column names used for OID and VERSION are configurable.

To configure these names, add the following lines to your application configuration file, making sure they come before any other DatabaseManager lines:

```
DatabaseManager.ObjectIdColumnName = "<ColName_for_ObjectId>"
DatabaseManager.VersionColumnName = "<ColName_for_Version>"
```

where `<ColName_for_ObjectId>` and `<ColName_for_Version>` are the column names to use instead of OID and VERSION; for example:

```
DatabaseManager.ObjectIdColumnName = "ObjectId"
DatabaseManager.VersionColumnName = "ObjectVersion"
```

**Note** In the DODS Database menu, select PostgreSQL to generate appropriate SQL. The CREATE TABLE statements that DODS generates don't always have the correct data types for PostgreSQL, so you may need to edit them. You can modify the data types that DODS generates by editing the `dods.conf` file.

## Running DiscRack

This section describes how to create the DiscRack PostgreSQL database and configure DiscRack to use PostgreSQL.

## Creating the DiscRack database

Before running the DiscRack application, you need to create the DiscRack database. Use PostgreSQL's `psql` tool to load the SQL script containing the necessary CREATE TABLE statements:

```
psql> \i <DiscRack_root>/data/create_tables.sql.PostgreSQL
```

where *DiscRack_root* is the root directory of the DiscRack application, normally <*enhydra_root*>/examples/DiscRack/discRack.

## Configuring DiscRack to run with PostgreSQL

Follow these steps to configure DiscRack to use PostgreSQL.

1. Add the following lines to the appropriate group in the `discRack.conf.in` file located in `DiscRack/discRack`.

   ```
   DatabaseManager.DB.db_id.JdbcDriver = "postgresql.Driver"
   DatabaseManager.DB.db_id.Connection.Url = "jdbc:postgresql:db_id"
   ```

   where *db_id* is the name of your database.

   ```
   DatabaseManager.DB.db_id.Connection.User = "your_username"
   DatabaseManager.DB.db_id.Connection.Password = "your_pwd
   ```

   where *your_username* is the user name and *your_pwd* is the password.

2. Add these lines to `discRack.conf.in`:

   ```
   DatabaseManager.ObjectIdColumnName = "ObjectId"
   DatabaseManager.VersionColumnName = "ObjectVersion"
   ```

   This is explained in "Using DODS" on page 144.

3. Modify the `CLASSPATH` in the `start` script as explained in "Using PostgreSQL with Enhydra" on page 144.

4. Rebuild the DiscRack application by typing `make` in the directory `DiscRack/discRack`.

5. In a browser, access DiscRack at `http://localhost:5555`.

# Chapter 9

# Using Enhydra Director

This chapter describes Enhydra Director, a load balancing and sharing server for Enhydra applications.

## Overview of Director

Director works in conjunction with a Web server to provide load balancing and failover capabilities to Enhydra applications.

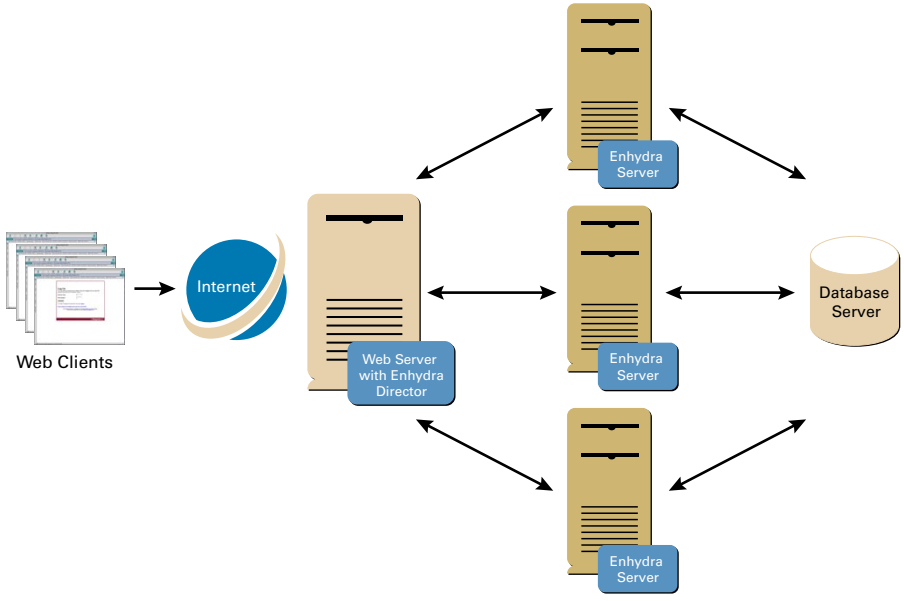
Here is a typical Director set-up, illustrated in "A typical Director installation" on page 148:

DemoCompany has a website, www.democompany.com, and uses an Enhydra application, demoApp, to process orders from this site. They also use an Apache Web Server, configured to use Director on a network interface outside their firewall. Behind the firewall are three machines running instances of demoApp, with private IP addresses. demoApp uses data located on a database server also located within the firewall.

When the Web server receives requests for http://www.democompany.com/demoApp, Director routes those requests to one of the three Enhydra servers with demoApp. The Enhydra server never receives requests directly from the user. All requests are received by Director and forwarded on to the appropriate server. If one of the Enhydra servers fails, Director sends requests to another Enhydra server.

As the number of connections increases, Director sends requests to the Enhydra servers based on the amount of load on the servers. This ensures that the load is distributed evenly among all available servers, and no one server is overloaded by connections unless the entire cluster is overloaded.

**Figure 9.1** A typical Director installation




# Installing and configuring Director

This section describes how to install Director and configure it to work with a Web server.

## System requirements

Director runs on Windows NT and 2000, Solaris, and Linux. Additionally, if you build Enhydra from the open-source distribution, the Director module will compile and run on most UNIX systems supported by the iPlanet Web Server.

**Note** Enhydra Director requires TCP/IP networking support. If you have not already done so, configure TCP/IP on at least one of your system's network interfaces.

### Windows NT and 2000

On Microsoft Windows NT and 2000, Director requires the following:

- Windows NT 4.0 (Service Pack 6 or later) or Windows 2000, running on an Intel 80386 or higher CPU (Pentium 200 MHz or faster recommended)
- 64 MB memory
- One of the following Web servers:
  - iPlanet Enterprise Server, version 4.0 or higher
  - Microsoft Internet Information Server, version 4.0 or higher

### Solaris

On Sun Solaris, Director requires the following:

- Sun Solaris version 2.6 running on a Sparc-compatible system with 64 MB RAM (128 MB recommended)
- One of the following Web servers:
  - Apache Web Server, version 1.3.9 or later
  - iPlanet Web Server, version 4.0 or higher

### Linux

On Linux, Director requires the following:

- Version 2.2 of the kernel or later, with support for recent versions of the `glibc` C library. Red Hat 6.1 or higher recommended, running on:
  - Intel 80386 or later CPU (Pentium 200 MHz or faster recommended)
  - 64 MB RAM (128 MB recommended)
- One of the following Web servers:
  - Apache Web Server, version 1.3.9 or higher
  - iPlanet Web Server, version 4.0 or higher

**Note** The minimum amount of memory required on your servers will vary greatly depending on how you use Director. Large-scale enterprise clusters serving complex applications under heavy load will require considerably more memory than a small cluster of servers used for development purposes.

## Preparation

Before installing Director, determine the following:

- Machines and applications on which you will be running the Enhydra Multiserver
- For each application:
  - Name of the machine and the port number on which each instance of the application will run. The port number is the port used when configuring a connection of type `EnhydraDirector` using the Multiserver Administration Console. For information on configuring your application in the Multiserver Administration Console, see "Configuring your application" on page 167.
  - URL prefix used to identify the application (for example, "`/demoApp`").
  - An application requires session affinity if requests for the same session to an application should always be routed to the application server instance that created the session. Unless your application supports distribution of its sessions among other instances, you should assume that session affinity is needed. If your application does not use session information at all, you do not need session affinity.
- Name of the Web server instance on which you want to configure Director

- TCP port of the Web server instance on which you intend to configure Director (by default, port 80)
- For Apache, determine:
  - Location of the configuration files for Apache, typically `/usr/local/apache/conf`
  - Location of the executables directory for your Apache server, typically `/usr/local/apache/bin`
- For iPlanet/Netscape, determine:
  - Location of the `obj.conf` file for your Web server instances

    On Windows, this is typically `C:\Netscape\Server4\<instance>\config\obj.conf`. On UNIX, this is often `/usr/netscape/server4/<instance>/config/obj.conf`.
  - Where to install the Director files

    On Windows, we recommend `C:\usr\local\enhydra\director\nsapi`. On UNIX, we recommend `/usr/local/enhydra/director/nsapi`.

## Using Director with Apache

This section describes how to configure Director to work with the Apache Web Server.

### Files

This section describes the files you use with Apache.

#### Apache extension module

The Apache extension module, `mod_enhydra_director.so`, is in the Apache `libexec` directory and is loaded with the AddModule and LoadModule directives in `httpd.conf`. This module forwards application requests to the Enhydra server and forwards the server response to the client.

#### Director configuration file

This file, `enhydra_director.conf`, contains configuration information for Director. By default, the Apache extension module (`mod_enhydra_director.so`) looks for this file in the same directory as the `httpd.conf` file.

#### Director daemon

This program, `edir_daemon`, runs in the background and monitors the state of the shared-memory region that is used by child instances of the Director module to coordinate load balancing.

`edir_daemon` is necessary because Apache uses separate processes to allow multithreading of simultaneous requests. The daemon is automatically launched when Apache starts, and automatically cleans up the shared memory and exits when Apache is stopped.

### Director status utility

This utility, `edir_status`, directly reads the shared-memory region used by the Director module. It reports the current state of each configured application and Enhydra server.

### Director scoreboard

The scoreboard, `enhydra_director.ipc`, contains the shared-memory data used by all the Apache child processes when they are handling Director requests. Normally, the contents of this file are kept in memory as an `mmap()`-based, shared-memory region.

This file also serves as the scoreboard lock file, using `fcntl()`-based locking. Its normal home is in the `edir` subdirectory of the Apache `logs` directory. It is normally only accessible to the Apache child user ID.

### Apache error log

The standard error log for the Apache server is typically called `error_log`. The Director handler logs all errors encountered while handling requests to this file. Certain startup messages are also logged to `error_log`; however, the Director daemon logs most startup errors to the system log.

### System log

The system log is typically in `/var/log/messages` or `/var/adm/messages`. If the daemon encounters errors while it is initializing, it has no access to the Apache error log, so it logs the error to the system log using the daemon logging channel.

## Procedure

The following steps assume that you are installing the Director files into your Apache installation directories. To build Director, you must first obtain, compile, and install Apache 1.3.9 or later. You must configure Apache so that Dynamic Shared-Object Support (`mod_dso`) is enabled. To do this, configure Apache with the following options:

```
./configure --enable-rule=SHARED_CORE --enable-module=SO
```

The build scripts use the Apache `apxs` utility to automatically obtain the correct installation directories for the Director components. You should take note of where this utility is installed in case you need to explicitly tell the `configure` script its location. `apxs` is a script that looks for the Apache configuration directories. Be sure Perl 5 is installed, because `apxs` is a Perl script.

### Step 1: Get Director source code

There are two source packages for creating the Director module for Apache. A smaller distribution containing only the source code of Director is included with the prebuilt distributions of Enhydra. The Director source code is also included with the full Enhydra source distribution, available under the Enhydra Public License (EPL). Obtain the Director source code from either of these sources and extract it into a working directory.

If you are using the Enhydra prebuilt distribution, extract <*enhydra_root*>/director/enhydra-director1.0.tar. The extracted source code will be in this directory:

```
./enhydra-director1.0
```

Change directories to enhydra-director1.0. This directory contains the subdirectories common, apache, nsapi, and so on. From this point on, this directory is referred to as $SRCROOT.

If you are using the Enhydra source distribution, extract enhydra-src<*version*>.tar.gz into the current directory. You will find the Director source in Enhydra/modules/EnhydraDirector/src/. From this point on, this directory is referred to as $SRCROOT.

### Step 2: Build the Director Apache module

If you installed Apache in the default location on either Solaris or Linux, run the ./configure script with no arguments, make the project, then install the project:

```
cd $SRCROOT/apache
./configure
make
make install
```

This builds the Director Apache module and installs the mod_enhydra_director.so file into the libexec directory for your Apache distribution.

There are several options for configure. For example:

```
--with-apxs=<Path to location of apxs>apxs
```

allows the location of the Apache apxs script to be specified if the default configuration script cannot find it.

If you use the --with-apxs option and configure still cannot find apxs, Apache may not have been configured to support shared libraries. If this is the case, you will have to reconfigure and rebuild Apache. See “Procedure” on page 151 for the options you need to configure Apache with shared-library support.

**Note** If you are using Apache with the mod_ssl module, use the --enable-eapi option of the configure script when configuring Director.

```
./configure --enable-eapi
```

This option is needed for extended APIs added by mod_ssl.

--enable-debugging adds debugging messages to the build. This allows the EnhydraDirectorDebug option to be specified in httpd.conf or <Options debug="0xNNNNNN"/> to be specified in enhydra_director.conf. Enabling debugging can be useful if you experience problems with setting up the module.

After running make install, you will find mod_enhydra_director.so in your Apache installation’s libexec directory. You will also find the file enhydra_director.conf.default in the conf directory.

**Note** If you installed Apache and Enhydra in nondefault locations, you need to specify the location of apxs and the location of the libexec directory when you configure Apache and Director.

Use the following `httpd.conf` options to specify the locations of Director configuration and data files:

- `EnhydraDirectorConfigFile` specifies the full path to the file, `enhydra_director.conf`. By default this is the configuration file in the `conf` subdirectory of your Apache installation.
- `EnhydraDirectorLockFile` specifies the location of the file used for locking the scoreboard during shared access. If you set this parameter, you should not need to set the `EnhydraDirectorDataFile` option.

  If you installed Enhydra on an NFS file system, you may need to set this parameter. Due to locking bugs in many, if not most, NFS implementations, Director does not work if the lock file is located on an NFS file system.

  By default, the lock file is located under the `logs` directory. To allow the load-balancing code to run as the Apache child user ID, the actual lock file is located under `logs/edir`, where `edir` has permissions set to allow the child processes to read and write. The permissions of the `logs` directory are unaffected.
- `EnhydraDirectorDataFile` is an expert setting for developers and advanced users only. You should not need to set this parameter unless you are a developer or are experimenting with alternate shared-memory modules (see `edir_shmem.h`). This option allows the scoreboard shared-data file to be specified as a different file than the lock file, or even as a System V IPC region.

  By default, the `mmap` shared-memory module uses the same file as the `filesys` mutex module (see `edir_mutex.h`). This means locking and data sharing use the same file on most modern UNIX implementations, including Solaris, Linux, and FreeBSD.
- `EnhydraDirectorDaemonPath` is an advanced option. If you install the Director daemon (`edir_daemon`) in a different location than the Apache `bin` directory, this option can be used to specify the full path to the daemon.
- `EnhydraDirectorDebug` is an advanced option. If you used `--enable-debugging` when building Director, this option can be used to set the debugging mask in the module. The debugging mask is also propagated to the daemon when it is started. For more information on possible values, see `edir_debug.h` or `enhydra_director.conf.default`.

  To enable full, highly voluminous debugging, including hex dumps of all communication between the module and Enhydra, specify the value `0xffffffff`. Be forewarned that doing this will severely limit Director's performance, and will dump hundreds of megabytes of logging data to the Apache error log.

  This option is identical to `<option debug="0xNNNNNNNN"/>` in `enhydra_director.conf`, except that it gets set much earlier and allows debugging of the code that loads the configuration file and scoreboard, as well as the daemon launching code.

## Configuring Director

Use the following steps to configure Director:

### Step 1: Locate httpd.conf

Find out where `httpd.conf` is located for your Apache installation. Usually this is something like `/usr/local/apache/conf/httpd.conf`. However, Apache is very flexible in how it can be installed, so this may differ. The build scripts use `apxs` to automatically install the compiled module and programs into the recommended locations.

### Step 2: Locate DSO modules

Determine where Apache stores its DSO modules. This is usually `libexec` under the Apache installation root. For example, on many Apache installations, the DSO module directory is `/usr/local/apache/libexec`. Some versions of Apache use `/usr/local/apache/sbin` or `/usr/local/apache/modules` for the DSO module directory. Note what directory your version of Apache uses, as you will need this when you configure the LoadModule line in `httpd.conf`.

### Step 3: Add the Director module to the Apache configuration

1. Open `httpd.conf` in a text editor.
2. Search for the last line in the file starting with `LoadModule`. The default `httpd.conf` should have a commented sample line for the LoadModule directive, such as the following:

   ```
   # LoadModule foo_module libexec/mod_foo.so
   ```

   If it doesn't contain a commented sample LoadModule line, go to the end of the file.
3. Using the last component of the `libexec` directory located in step 2, add a line to the list of LoadModules directives (if any), that looks like:

   ```
   LoadModule enhydra_director_module libexec/mod_enhydra_director.so
   ```

   If the last component of your Apache installation's `libexec` path ends in, for example, `/modules`, then you would instead add the line:

   ```
   LoadModule enhydra_director_module modules/mod_enhydra_director.so
   ```

   If there are other LoadModule lines, they usually will contain the correct path to the `libexec` path, so you can use the other directives as a guide.

**Note** If you are using `mod_rewrite`, make sure the `LoadModule rewrite_module` directive comes before the `LoadModule enhydra_director_module` directive if you intend to use `mod_rewrite` to redirect URLs to Director. Also, you should use the `mod_rewrite [PT]` tag to force pass-through of redirected URLs.

4. Search for the last line starting with `AddModule`. If such a line does not exist as a comment, go to the end of the file and add the line:

   ```
   AddModule mod_enhydra_director.c
   ```
5. Add any desired Enhydra-specific directives such as `EnhydraDirectorDebug` to `httpd.conf`. These directives are optional in almost all cases.

6 Save `httpd.conf`.

### Step 4: Create the Director configuration file

The Director configuration file for Apache is normally located in the same directory as `httpd.conf`. However, you can use the `EnhydraDirectorConfigFile` directive in `httpd.conf` to specify a different file:

```
EnhydraDirectorConfigFile /usr/<enhydra_root>/director/enhydra_director.conf
```

For a description of the syntax and the directives available in `enhydra_director.conf`, see "Editing enhydra_director.conf" on page 168.

The file `enhydra_director.conf.default` contains several useful examples that can be easily copied to match your site setup. The comments within the default file also describe most of the available options.

It's important to note that the configuration file is based on a XML DTD (`EnhydraDirectorConfig.dtd`) and is strictly validated. This means the file must contain valid XML, and it must follow the format defined in the DTD. Case is important in all section and property names, and order is important among different sections.

### Step 5: Restart the Apache server

Use the `apachectl` script, or your own script, to restart the Apache server. The usual command is `apachectl restart`.

### Step 6: Verify correct startup

1 Use `ps -aef` or `ps -ax`, depending on your system, to make sure `edir_daemon` is running.

2 In the Apache logs directory, there should be a subdirectory called `edir` that contains the file `enhydra_director.ipc`. This file is the lock and scoreboard file.

3 In the Apache `bin` directory, where `httpd` is located, run `edir_status`. This should dump current information from the scoreboard and configuration files. If this succeeds, Director is running.

4 If errors occur within the `edir_daemon` process, the errors are logged to the system log using `syslog()`. The daemon channel is used. Normally the messages go to the messages file, which is `/var/adm/messages` on Solaris and `/var/log/messages` on Linux.

5 If initialization errors occur within the module, the errors are reported in the Apache error log.

### Step 7: Verify connectivity

1 Start a Web browser.

2 If you have set a `<Status>` section in the configuration, you can connect to its URL prefix to ensure that the handler is functioning. You can use `enhydra_director.conf.default` as your configuration file and connect to `http://localhost/status` to verify Director is working correctly. The application information will not be valid, however.

3 Connect to one of your configured application prefixes.

4 Errors encountered while handling requests are reported in the Apache error log.

## Troubleshooting

**Problem** The Director Apache extension fails to load when Apache is started.

- Check `httpd.conf` for correct configuration of the module, as described in the previous sections.
- Check the system log file for errors reported by the `edir_daemon` program.
- Check the Apache error log for errors reported by the Director module.
- Ensure that the `enhydra_director.conf` file exists and contains valid and syntactically correct configuration data. If the syntax is not valid, Director reports the specific errors and line numbers to the Apache error log.

**Problem** Director starts with no errors, but I can't connect to my application.

- If you configured the `<Status>` section, check the status prefix for connection failures to your back-end Enhydra servers.
- Make sure that the prefixes configured in `enhydra_director.conf` match the prefixes configured for your application on the Enhydra server. The prefixes must match exactly, including case. Trailing slashes in the prefix are not significant and are ignored.
- Make sure that the application servers and ports in the `enhydra_director.conf` file refer to valid and running instances of your application. Make sure your application is running on the Enhydra server.
- You application must use the Director connection method on ports to which the Apache Director extension will dispatch requests. The connection method is set in the `MultiServer.conf` file for your application. The preferred method for setting the connection method is using the Admin Console.
- We highly recommend that you configure the `<Status>` section in the configuration file so you can examine the runtime state and configuration of the handler.

**Problem** Apache child processes dump core (crash).

- This is evidence of an error in the Director module that needs the attention of Lutris engineers.
- There's also the possibility that there is a bug elsewhere in Apache, or in some other module. If possible, remove all unneeded modules from the `httpd.conf` file to try to narrow the problem down to the Enhydra module.
- If you're an engineer and can give us stack traces from the Gnu Debugger (`gdb`), we would be very appreciative, especially stack traces from `--enable-debugging` compilations.
- Please report any bugs of this nature to Enhydra.org, at http://www.enhydra.org. The more details you can provide about what led to the failure, the better we can

track down and fix any problems. Machine state information, configuration information, and any steps to reproduce the bug would be very helpful.

## Using Director with iPlanet Web Server

### Installation

Installation of this module follows the same general procedure as with any other NSAPI extension to a Netscape Web Server, so you may want to refer to the Netscape online documentation at http://www.iplanet.com/ for additional information.

#### Files

The Director module for iPlanet is either the NSAPI Extension `libedir.so` for UNIX or the NSAPI Extension `EnhydraDirector.dll` for Windows.

This file is an NSAPI-extension DLL that does the actual work of forwarding an Enhydra request to the correct Enhydra application server. It must be installed on each iPlanet Web Server instance that will serve your application's URL prefix.

For example, if you have set up iPlanet to enable the Web server www.myhost.com, and you want your application to be located at http://www.myhost.com/myApp, you must enable the `EnhydraDirector.dll` extension in the `obj.conf` configuration file for the www.myhost.com Web server instance. See "Configuring Director for iPlanet" on page 158 for details on configuring this module.

The Director configuration file is `enhydra_director.conf`. One of the required settings in `obj.conf` is the location of this file.

#### Procedure for UNIX systems

These instructions assume that you will install the Director files into `/usr/local/enhydra4.0/director/enhydra-director1.0`. If you want to install the files in some other location, replace this location with your own installation directory in all the following steps.

1. Change directories to `/usr/local/enhydra4.0/director/`.
2. Extract `enhydra-director1.0.tar.gz` with the following command:

   ```
   gunzip enhydra-director1.0.tar.gz ; tar xvf enhydra-director1.0.tar
   ```

   This creates a subdirectory, `enhydra-director1.0`, in the current directory.
3. Change directories to `enhydra-director1.0`.
4. Build the extension-shared object `.so` file.

   The build uses a GNU automake environment, so you begin building by running the `./configure` script. Type `./configure --help` to get a list of supported options.

   If you have installed iPlanet in a nonstandard directory, you need to use this option to specify where the plugins directory is located for your server:

   ```
   --with-netscape-plugins-dir
   ```

Also, the `--enable-debugging` option may be used to add full verbose debugging macros to the build. When you are finished building, you should have these files: `./.libs/libedir.so`, `./enhydra_director.conf.default`, and `./obj.conf.example`.

5 Install the Director files.

Assuming you have chosen the recommended directory, install the files you built above as:

```
/opt/netscape/EnhydraDirector/libedir.so
/opt/netscape/EnhydraDirector/enhydra_director.conf.default
```

The `obj.conf.example` file is just an example `obj.conf` where the necessary additions are highlighted in comments. You must edit your existing `obj.conf` file to configure your particular server.

### Procedure for Windows

These instructions assume that you will install the Director files into `C:\usr\local\<enhydra_root>\director\nsapi\`. If you want to install the files to a different location, replace `C:\usr\local\<enhydra_root>\director\nsapi\` with your own installation directory in all the following steps. If you want to build the DLL from source, see "Building Director for iPlanet" on page 166.

1 Use the precompiled binary (`<enhydra_root>\director\nsapi\EnhydraDirector.dll`) or build the Director files from the source. For instructions on building the iPlanet DLL from source, see "Building Director DLLs from source code" on page 166.

2 Install the Director files. You need the following files:

```
C:\usr\local\<enhydra_root>\director\nsapi\EnhydraDirector.dll
C:\usr\local\<enhydra_root>\director\nsapi\enhydra_director.conf.default
```

## Configuring Director for iPlanet

To configure Director for iPlanet, three steps are required. First, stop the Netscape Web Server by running `C:\Netscape\Server4\<instance>stopsvr` on Windows and running `/usr/netscape/server4/<instance>/stop` on UNIX. Second, install the NSAPI extension. Repeat these two steps as needed for any other virtual Web servers that need to be configured with Director. Finally, edit `/<netscape root>/https-<machine name>.<domain name>/obj.conf` for each Web server instance to specify the location of the Director module.

**Note** The `<enhydra_root>/director/nsapi/obj.conf.example` file is a sample `obj.conf` file that you can edit with site-specific information to simplify configuration.

1 Edit `obj.conf` and go to the lines where the `<Init...>` sections are located. This should be near the beginning of the file.

- For UNIX, add the following init lines:

```
Init fn="load-modules"
funcs="edir_init,edir_nsapi_handler,edir_nsapi_name_trans"
shlib="/usr/netscape/EnhydraDirector/libedir.so"

Init fn="edir_init"
conffile="/usr/netscape/server4/<Instance>/config/enhydra_director.conf"
```

- For Windows, add the following init lines:

```
Init fn="load-modules" funcs="edir_init,edir_nsapi_handler,edir_nsapi_name_trans"
shlib="C:/Netscape/EnhydraDirector/EnhydraDirector.dll"
Init fn="edir_init"
conffile="C:/Netscape/Server4/<Instance>/config/enhydra_director.conf"
```

**Note** The paths in `obj.conf` for Windows must use a forward slash (`/`) character to separate directories.

**Note** The init lines above must consist of only two actual lines in the `obj.conf` file.

If your Netscape server is not installed in `/usr/netscape`, or if you did not install the `libedir.so` DLL in the recommended location, edit the init lines accordingly. Replace `<Instance>` with the Web server instance for which you are configuring.

**2** In the `<Objectname=default>` section, add the line:

```
NameTrans fn=edir_nsapi_name_trans
```

The previous line must go immediately before:

```
NameTrans fn=document-root root=...
```

**3** Just after the closing `</Object>` for `<Objectname= default>`, add the section:

```
<Object name=enhydra_director>
Service fn="edir_nsapi_handler" method=(GET|HEAD|POST)
</Object>
```

The `obj.conf.example` file gives an example of the `obj.conf` file for a fictitious site.

**4** Create the configuration file.

The Director configuration file for NSAPI is the file indicated by the `conffile=...` directive in the `init` lines described previously. There can be multiple instances of Director on a machine, with one instance per configured Web server instance. Each Director instance should be assigned a different configuration file.

The full description of the syntax and of all the directives available in the configuration file is beyond the scope of this document. However, the file `enhydra_director.conf.default` contains several useful examples that can be easily copied to match your site setup. The comments within the default file also describe most of the available options.

The configuration file is based on an XML DTD (`EnhydraDirectorConfig.dtd`) and is strictly validated. This means that the file must contain valid XML, and it must follow the format defined in the DTD. Case is important in all sections and property names, and order is important among different sections.

**5** Start the Netscape server.

Run the `C:\Netscape\Server4\<instance>/startsvr` or `/usr/netscape/server4/<instance>/start` script, depending on your platform.

**6** Make sure the Netscape server started correctly and is running.

Check the Netscape server error log for errors that might have occurred while iPlanet loaded the Director DLL. Also check for Director errors.

7 Verify that you can connect to the server.

- Start a Web browser.

  If you have set a `<Status>` section in the configuration, you can connect to its URL prefix to ensure that the handler is functioning. You can use `enhydra_director.conf.default` as your configuration file and connect to http://localhost/status to verify Director is working correctly. The application information will not be valid, however.

- Connect to one of your configured application prefixes.

  If serious errors occur, they are logged to the server error log.

## Troubleshooting

**Problem** The NSAPI extension fails to load when the Netscape server is started.

- Ensure that the `obj.conf` configuration has the right path for the `.so` or `.DLL` file. This is the `Init ... shlib= ...` directive.
- Check the Netscape server error log for errors.
- Ensure that the `enhydra_director.conf` file exists and contains valid and syntactically correct configuration data. If the syntax is not valid, Director reports the specific errors and line numbers to the Netscape server error log.

**Problem** Director starts up without errors, but I can't connect to my application.

- If you configured the `<Status>` section, check the status prefix for connection failures to your back-end Enhydra servers.
- Make sure that the prefixes configured in `enhydra_director.conf` match the prefixes configured for your application on the Enhydra server. The prefixes must match exactly, including case. Trailing slashes in the prefix are not significant and are ignored.
- Make sure that the application servers and ports in the `enhydra_director.conf` file refer to valid and running instances of your application. Make sure your application is running on the Enhydra server.
- Your application must use the `EnhydraDirector` connection method on ports that the NSAPI Director extension will dispatch requests to.
- We highly recommended that you configure the `<Status>` section in the configuration file so you can examine the runtime state and configuration of the handler.

**Problem** The Netscape server crashes, hangs, or dies unexpectedly.

- This is evidence of an error in the handler or filter DLL that needs the attention of Lutris engineers.
- Please report any bugs of this nature to Enhydra.org, at http://www.enhydra.org. The more details you can provide about what led to the failure, the better we can track down and fix any problems. Machine state information, configuration information, and any steps to reproduce the bug would be very helpful.

# Using Director with Internet Information Server (IIS)

## Installation

Three files are necessary to use Director with IIS: `EnhydraFilter.dll`, `EnhydraHandler.dll`, and `enhydra_director.conf`. These files are described in the following sections.

### ISAPI filter—EnhydraFilter.dll

This file is an ISAPI header preprocessing filter that reroutes Enhydra application URLs to the Director ISAPI Handler application. It can be configured either as a global ISAPI filter or as a local filter on a particular Web server instance.

- If the filter is applied globally, it reroutes Enhydra URLs for all IIS Web Servers on the system.
- If the filter is configured as a local Web server instance, it routes only Enhydra URLs for the configured Web server.

See the following section for details on configuring the filter.

### ISAPI handler extension—EnhydraHandler.dll

This file is an ISAPI extension DLL that does the actual work of forwarding an Enhydra request to the correct Enhydra application server. It must be installed on each IIS Web Server instance that will serve your application's URL prefix.

For example, if you have configured IIS to enable the Web server www.myhost.com, and you want your application to be located at http://www.myhost.com/myApp, you must enable the `EnhydraHandler.dll` extension on the www.myhost.com Web server instance.

### Director configuration file—enhydra_director.conf

This file contains the configuration for Director.

- If you configured `EnhydraFilter.dll` as a global filter, there will be only one instance of this file, and it controls the configuration for all IIS Web Server instances on the system.
- If the filter was configured for specific servers, a different instance of this file will exist for each server.

The configuration file must be located in the same directory as `EnhydraFilter.dll`.

## Procedure

These instructions assume that you installed Enhydra in `c:\usr\local\<enhydra_root>`. If you installed Enhydra in a different location, replace this path with your own installation directory in the following steps.

The ISAPI DLLs, `EnhydraFilter.dll` and `EnhydraHandler.dll` for Director are located at `c:\usr\local\<enhydra_root>\director\isapi`. If you are compiling the DLLs from source, see "Building Director DLLs from source code" on page 166.

**Note** If you are installing IIS, choose Custom install, and install all components except the Personal Web Manager.

## Configuring Director for IIS

To configure Director with IIS, follow these steps:

### Step 1: Stop IIS

To stop IIS:

1. In the Control Panel, double-click Services on NT. On Windows 2000, open Control Panel | Administrative Tools | Services.
2. Select World Wide Web Publishing Service.
3. Click Stop to stop the World Wide Web Publishing Service.
4. Stop the FTP Publishing Service, if you are using the FTP service included with IIS.

### Step 2: Install the filter (global)

If you are installing the filter globally, follow this procedure. If you are installing for a specific server, see "Step 3: Install the filter (for a specific server)."

1. Start the IIS Internet Service Manager program by choosing Start | Programs | Windows NT 4.0 Option Pack | Microsoft Internet Information Server | Internet Service Manager on NT.

   On Windows 2000, open Control Panel | Administrative Tools | Internet Service Manager.

2. On NT, in the tree view that contains the Console Root folder, expand the Internet Information Server subfolder if it has not already been expanded. You should see a small computer icon next to the name of your machine. On Windows 2000, there is no Console Root. The Internet Information Server folder is the root.
3. Right-click the name of your machine (not Default Web Site) and click Properties.
4. In the Master Properties frame, select WWW Service in the selection box, then click Edit to edit the master IIS properties for your machine.
5. Select the ISAPI Filters tab.
6. If the EnhydraFilter filter is already installed, remove it.
7. Click Add.
8. For the Filter Name property, enter `EnhydraFilter`.
9. For the Executable property, type in or browse to the file `C:\usr\local\<enhydra_root>\director\isapi\EnhydraFilter.dll`.
10. Click OK to accept the filter name and executable.
11. Back in the WWW Service Master Properties dialog box, click Apply to add the filter, then click OK to close the dialog box. The filter does not become active until IIS is restarted.

The filter is now installed and will become active when IIS is restarted.

### Step 3: Install the filter (for a specific server)

1. Start the IIS Internet Service Manager program by choosing Start | Programs | Windows NT 4.0 Option Pack | Microsoft Internet Information Server | Internet Service Manager on NT.

   On Windows 2000, select Start | Settings | Control Panel | Administrative Tools | Internet Service Manager.

2. On NT, in the tree view that contains the Console Root folder, expand the Internet Information Server subfolder if it has not already been expanded. You should see a small computer icon next to the name of your machine. On Windows 2000, there is no Console Root. The Internet Information Services folder is the root.
3. Right-click the entry for your website instance or the Default Web Site for the IIS default website instance, and click Properties.
4. Select the ISAPI Filters tab in this dialog box.
5. If the EnhydraFilter filter is already installed, remove it.
6. Click Add.
7. For the Filter Name property, enter `EnhydraFilter`.
8. For the Executable property, type in or browse for the file `C:\usr\local\<enhydra_root>\director\isapi\EnhydraFilter.dll`.
9. Click OK to accept the filter name and executable.
10. In the Default Web Site Properties dialog box, click Apply to add the filter, then click OK to close the dialog box.
11. Restart the WWW Publishing Service in the Services Control Panel.

### Step 4: Install the handler

Repeat as needed for multiple IIS websites.

1. Start the IIS Internet Service Manager program by choosing Start | Programs | Windows NT 4.0 Option Pack | Microsoft Internet Information Server | Internet Service Manager on NT.

   On Windows 2000, select Start | Settings | Control Panel | Administrative Tools | Internet Service Manager.

2. On NT, in the tree view that contains the Console Root folder, expand the Internet Information Server subfolder if it has not already been expanded. You should see a small computer icon next to the name of your machine. On Windows 2000, there is no Console Root. The Internet Information Services folder is the root.

   Underneath this, you should see entries for your website(s). If you have only one website on this system, it is usually set up as Default Web Site.

3. Expand the subtree for the website.
4. If a virtual directory called `EnhydraDirector` already exists, remove it.
5. Right-click Default Web Site (or the name of an alternate website) and select New | Virtual Directory from the popup menu.

6 The alias for the virtual directory must be `EnhydraDirector`. This entry is case-sensitive.

7 Click Next.

8 For the physical path, type in (or browse to) `C:\usr\local\<enhydra_root>\director\isapi`.

9 Click Next.

10 For the permissions, check Allow Execute Access on NT, Execute on Windows 2000, and uncheck all other permissions. This permission must be checked in order for the handler to work properly.

11 Click Finish to add the new Virtual Directory.

The handler is now installed for the selected WWW server instance. Repeat "Step 3: Install the filter (for a specific server)" for each WWW server on which you intend to use Director.

### Step 5: Create the configuration file

Create the configuration file `C:\usr\local\<enhydra_root>\director\isapi\enhydra_director.conf`. The easiest way to do this is to copy `enhydra_director.conf.default` to `enhydra_director.conf`.

The full description of the syntax and of all the directives available in the configuration file is beyond the scope of this document. However, the default file contains several useful examples that can be easily copied to match your site setup. The comments within the default file also describe most of the available options.

An important thing to note is that the configuration file is based on an XML DTD (`EnhydraDirectorConfig.dtd`) and is strictly validated. This means that the file must contain valid XML, and it must follow the format defined in the DTD. Case is important in all sections and property names, and order is important among different sections.

### Step 6: Start IIS

1 On NT, open the Services dialog box from the Control Panel and double-click Services. On Windows 2000, select Start | Settings | Control Panel | Administrative Tools | Internet Service Manager.
2 Find the entry for World Wide Web Publishing Service and select it.
3 Click Start to start the World Wide Web Publishing Service.

IIS should now be running.

### Step 7: Verify correct startup

1 On NT, run Start | Programs | Administrative Tools (Common) | Event Viewer. On Windows 2000, open Start | Settings | Control Panel | Administrative Tools | Event Viewer.
2 Check the Application log for EnhydraFilter events.

If Director started correctly, the filter logs and events indicates the configuration file were processed successfully.

### Step 8: Verify connectivity

1 Start a Web browser.

2 If you have specified a `<Status>` section in the configuration, you can connect to its URL prefix to ensure that the handler is functioning. You can use `enhydra_director.conf.default` as your configuration file and connect to http://localhost/status to verify Director is working correctly. The application information will not be valid, however.

3 Connect to one of your configured application prefixes.

If connection errors occur, they are logged to the Application event log and can be viewed with the Event Viewer.

## Troubleshooting

**Problem** `EnhydraFilter.dll` fails to load when IIS is started.

- Ensure that the IIS configuration has the right path for the filter DLL.
- Check the Application event log for errors reported by the filter or by IIS.
- Ensure that `enhydra_director.conf` exists and contains valid and syntactically correct configuration data. If the syntax is not valid, the filter should report the specific errors and line numbers to the Application event log.

**Problem** The Filter starts up but I can't connect to my application.

- Check the application event log for EnhydraHandler error reports.
- If you configured the `<Status>` section, check the status prefix for connection failures to your back-end Enhydra servers.
- Make sure that the prefixes configured in `enhydra_director.conf` match the prefixes configured for your application on the Enhydra server. The prefixes must match exactly, including case. Trailing slashes in the prefix are not significant and are ignored.
- Make sure that the application servers and ports in the `enhydra_director.conf` file refer to valid and running instances of your application.
- Make sure your application is running on the Enhydra server.
- Your application must use the `EnhydraDirector` connection method on ports that the IIS handler will dispatch requests to.
- We highly recommend that you configure the `<Status>` section in the configuration file so you can examine the runtime state and configuration of the handler.

**Problem** The Event Log messages contain no new lines.

This is a Windows problem that happens immediately after you install an application that registers itself with the event logger. If you reboot Windows, this problem goes away.

**Problem** MSVCRTD.DLL not found

You are trying to run a debug compiled version of the filter or handler, and do not have the debug runtime libraries installed. These libraries come with Microsoft Visual C++.

**Problem** IIS crashes with an Unhandled Exception, or the Web Application Manager reports that the handler is dying.

- This is evidence of an error in the handler or filter DLL that needs the attention of Lutris engineers.
- Please report any bugs of this nature to Enhydra.org, at http://www.enhydra.org. The more details you can provide about what led to the failure, the better we can track down and fix any problems. Machine state information, configuration information, and any steps to reproduce the bug would be very helpful.

## Building Director DLLs from source code

Director includes precompiled binaries of the DLLs needed to configure Director with iPlanet and IIS. However, if you want to make modifications or customizations to these Director DLLs, you can compile the DLLs from the source code.

### Building Director for iPlanet

If you don't have Microsoft Visual C++ 6.0 or later, you should use the precompiled NSAPI extension DLL from your Enhydra install or from the Enhydra.org website.

To build `EnhydraDirector.dll` from source:

1. Go to the `<enhydra_root>\director\` directory (by default, `C:\usr\local\<enhydra_root>\director\`).
2. Extract `enhydra-director1.0.zip` to a temporary directory using an unzip utility like WinZip.
3. Start Visual Studio and open the Director NSAPI workspace, the `/nsapi` directory in the directory where you extracted `enhydra-director1.0.zip`.
4. Set `EnhydraDirector` as the active project.
5. Set the active configuration to Debug or Release, according to your needs.
6. If you installed Netscape server in a nondefault location, you also need to edit the build settings (C Preprocessor) to set the correct include directories for the Netscape header files.
7. Run Build All to build the DLL.

In the same directory as the workspace, you will find a copy of both `enhydra_director.conf.default` and `obj.conf.example`. The newly built DLL will be found in the `Debug` or `Release` subdirectory, according to the usual Visual Studio convention.

### Building Director for IIS

To build `EnhydraFilter.dll` and `EnhydraHandler.dll` for IIS:

1. Go to the `<enhydra_root>\director` directory (by default, `C:\usr\local\<enhydra_root>\director`).
2. Extract `enhydra-director1.0.zip` to a temporary directory using an unzip utility like WinZip.
3. Start Visual Studio and open the Director ISAPI workspace, the `/isapi` directory in the directory where you extracted the zip file.
4. Set `EnhydraDirector` as the active project.
5. Set the active subproject to All.
6. Set the active configuration to Debug or Release, according to your needs.
7. Select Run | Build All to build the `EnhydraFilter.dll` and `EnhydraHandler.dll`.

The DLLs will be in `EnhydraHandler\<Debug or Release>\EnhydraHandler.dll` and `EnhydraFilter\<Debug or Release>\EnhydraFilter.dll`.

## Configuring your application

There are two ways to configure your applications to work with Director. You can modify the setting of the application through the Multiserver Administration Console, or you can edit the configuration files for your application by hand.

### Configuring your application with the Multiserver Administration Console

Create a new connection method for you application, and select `EnhydraDirector` as the connection method, making sure the URL prefix and port number are correct for your application. For more information on using the Admin Console, see Chapter 3, "Using the Multiserver Administration Console."

### Configuring your application by editing multiserver.conf

`multiserver.conf` contains all the settings for applications run by the Enhydra Multiserver. Edit this file to configure your application to work with Director. For more information on editing this file, see "Modifying configuration files by hand" on page 30.

The Connection Methods section of `multiserver.conf` defines channels for Enhydra applications. Add a new channel for your application by adding two lines:

```
Connection.HttpConn<port number>.Type = EnhydraDirector
Connection.HttpConn<port number>.Port = <port number>
```

For example, if your application is on port 8005, add the following:

```
Connection.HttpConn8005.Type = EnhydraDirector
Connection.HttpConn8005.Port = 8005
```

In the Channel section, tell the Multiserver the application name, URL, and whether the application is enabled:

```
Connection.HttpConn<port number>.<Application name>.Servlet = <Application name>
Connection.HttpConn<port number>.Url = /<HTTP server path>
Connection.HttpConn<port number>.Enabled = yes
```

If your application is called `simpleApp`, on port 8005, and its URL is `/simpleApp`, add the following:

```
Connection.HttpConn8005.simpleApp.Servlet = simpleApp
Connection.HttpConn8005.Url = /simpleApp
Connection.HttpConn8005.Enabled = yes
```

## Load balancing with Director

Director uses the following algorithm to load balance requests to Enhydra applications.

If the traffic is low, Director uses a round-robin load balancing scheme to choose which Enhydra server to route requests to. Round-robin load balancing involves directing requests to the next server, regardless of the load on that particular server.

Should the number of requests increase to a point where all servers are busy, Director uses a scoreboard load-balancing scheme. In scoreboard load balancing, Director calculates a score for each server based on the number of concurrent sessions and a user-defined value, `weight`, that specifies how much load the application instance should take.

The score for the server is calculated according to the following formula:

```
score = (weight * scale) / num_conn
```

`weight` is the value assigned in `enhydra_director.conf`, and `num_conn` is the number of current active sections assigned to the server. `scale` is determined by another formula:

```
scale = (maxint / maxweight)
```

`maxint` is the maximum value for an unsigned long integer. `maxweight` is the maximum weight of all the entries in `enhydra_director.conf` for that particular application instance.

In the scoreboard scheme, the server with the highest score that is not failing is assigned the next connection. If all the servers are failing, Director returns to round-robin load balancing. Similarly, if the number of connections goes back to zero, round-robin load balancing is used.

## Editing enhydra_director.conf

This section lists the options for `enhydra_director.conf`, the configuration file for Director. `enhydra_director.conf` is an XML document that must be strictly validated. The general options for Director are specified with an `<Options...>` tag in `enhydra_director.conf`.

The following table lists the global settings for Director.

**Table 9.1** General options for Director

| Option | Example | Description |
|---|---|---|
| daemon | daemon="/usr/local/<*enhydra_root*>/" | Location of the Director daemon. This field is optional and is ignored on multithreaded platforms where the daemon is not used. |
| daemondebug | daemondebug="0x00000001" | Debugging mask for the Director daemon. This field is optional. |
| daemoninterval | daemoninterval="1" | Number of seconds between each check of the scoreboard activity status by the daemon. This field is optional. |
| daemontimeout | daemontimeout="30" | Number of seconds of inactivity after which the daemon exits. This field is optional and is ignored on multithreaded platforms where the daemon is not used. |
| debug | debug="0x00000001" | Debugging mask for the Director module. This field is optional. |
| retrytime | retrytime="30" | Number of seconds Director will wait before attempting to open another connection to an application server that has stopped responding to requests from Director.<br>If all servers for an application are failing, Director reverts to a round-robin load-balancing scheme. |

The following example is an Options tag from a sample `enhydra_director.conf`:

```
<Options retrytime="30" />
```

For each application that Director will handle requests for, you need an application tag (`<Application...>`) in `enhydra_director.conf`. The following table lists the option for the application tag.

**Table 9.2** Per-application options

| Option | Example | Description |
|---|---|---|
| prefix | prefix="/demoApp" | URL prefix on the Enhydra server for a particular application. This string must match the actual URL exactly, including case. |

For each configured application, you should add one or more application server definitions to `enhydra_director.conf`. These options will be within an `<Application...>` block. The following table lists the options available within an application server tag.

**Table 9.3** Options available in an Application tag

| Option | Example | Description |
|---|---|---|
| `host` | `host="host1"` | Host name of the machine on which the application server is running. |
| `port` | `port="7000"` | Host's port number on which the application server is listening. |
| `weight` | `weight="1"` | Integer describing how much load the server instance takes on when the load is heavy. A higher weight indicates that the server instance will receive more connections than a server instance with a lower value. Thus, a server instance given a weight of 3 will take on three times as much load as a server instance with a weight of 1. |
| `auth` | `auth="myPass"` | `auth key` setting of the Enhydra server. This string must match the authorization key of the server exactly. |

The following example is an application tag from a sample `enhydra_director.conf`:

```
<Application prefix="/demoApp">
   <AppServer host="host1" port="7000" weight="1" auth="myPass" />
   <AppServer host="host1" port="7001" weight="1" auth="myPass" />
   <AppServer host="host1" port="7002" weight="1" auth="myPass" />
   <AppServer host="host2" port="7000" weight="3" auth="myPass" />
</Application>
```

The status tag lets you configure a URL to check the status of Director. This tag is normally used in conjunction with restriction settings to limit the clients to authorized administrators. The following table lists the option for the status tag.

**Table 9.4** Status tag option

| Option | Example | Description |
|---|---|---|
| `prefix` | `prefix="/status"` | URL prefix on the Enhydra server for the status application. This string must match the actual URL exactly, including case. |

The restriction settings must be within an application tag or a status tag. The following table describes the options for restricting applications.

**Table 9.5** Restriction settings for applications

| Option | Example(s) | Description |
|---|---|---|
| `client` | `client="127.0.0.1"`<br>`client="10.20.20.0/24"` | Restricts application clients to a particular machine or subnet. This is useful for creating administration applications, or for applications that should be accessed only by authorized clients. |

**Table 9.5** Restriction settings for applications (continued)

| Option | Example(s) | Description |
|---|---|---|
| server | server="www.democompany.com"<br>server="www.democompany.com" port="800" | Restricts an application to requests from a particular Web server. This value can be either an IP address or a host name.<br>This option can be further modified by defining port, so that only requests from a particular port on a particular Web server will be processed.<br>Note that this value is different from the application's host value. host specifies the Enhydra application server, while server specifies the Web server. |
| virtualserver | virtualserver="vweb.democompany.com"<br>virtualserver="vweb.democompany.com" port="80" | Restricts an application to requests coming from a particular virtual server on the Web server.<br>This option can be further modified by defining a port, so that only requests from a particular port on a particular virtual server will be processed. |

The following example is a status tag with restriction settings from a sample enhydra_director.conf:

```
<Status prefix="/status">
   <Restrict server="127.0.0.1" />
   <Restrict client="127.0.0.1" />
</Status>
```

# Optimizing Director performance

A Web server with Director on the front-end, configured with a cluster of Enhydra servers, provides a fairly scalable solution. The following information should help you gain performance benefits when you use Director and Enhydra.

## Load balancing HTTP requests and Enhydra

Director provides a form of loosely coupled clustered Java application servers. The Web server, using Director, distributes requests to a number of Enhydra servers.

HTTP request load balancers, like Cisco System's Local Director, Radware's Web Server Director, Alteon's ACEdirector, and similar hardware, are designed for high volume sites that need to distribute HTTP requests to a number of logical Web servers. They also provide some basic IP-based session affinity, so session requests always go back to the server on which the session was created. In general, these

HTTP load balancers perform best when distributing typical high-volume HTTP traffic: static HTML and files. This is because Web servers are typically limited by file I/O, rather than processing power. Most of the work of a Web server is copying files from the local file system and sending them to the network.

Application servers, however, are generally limited by processing power, not file I/O. This is particularly true with Java application servers. Producing dynamic content usually requires more processing power, and application servers are best utilized when processing requests than serving files.

Director solves these problems by moving the dynamic processing to one or more separate machines and distributing the dynamic request load among these machines. It allows the Web server to process HTTP requests, and passes application requests to the application servers, and it does all this transparently.

## Separating static and dynamic content

When you write your Enhydra applications, it is best to separate the static Web content (HTML pages and graphics) from the dynamic content created by your application. This allows the Web server to serve the static content and your application servers to create the dynamic content, which plays to the strengths of both servers.

## Scaling up

You may, however, reach a point where the load of the Web server will be too great. This situation may be caused by the limitations of your HTTP server (for instance, Apache's multi-process model described in "Increasing performance for Director and Apache on Linux" on page 173) or because the heavy load is pushing the limits of your HTTP server's hardware.

One solution is to purchase a more powerful Web server. A second solution is to have more than one Web server serving requests. The second solution is more desirable when the cost of purchasing a more powerful Web server exceeds the combined cost of an HTTP request load balancer (like Cisco System's Local Director, Radware's Web Server Director, Alteon's ACEdirector, and similar hardware), a number of less expensive Web servers, and a larger number of Enhydra application servers. This second solution scales up well, and Director is well suited for this set-up.

Most HTTP request load balancers provide for session affinity to the server that began the user session based on the user's IP address.

**Note** AOL clients, due to their proxying mechanism, break this type of session affinity because the user's IP address changes for each request.

Director is more reliable than IP-based session affinity because it recognizes session cookies and URL-encoded session IDs for Enhydra applications, and will attempt to preserve session affinity by routing request to the originating server. If the originating server is unavailable, Director will route requests to the next available server, according to "no affinity" load-balancing rules.

### Limitations of Director and session affinity

There are two areas that Director does not currently deal well with: mid-request failover and multi-host session management.

If a back-end Enhydra server fails without sending the requested data to the user, Director will send the request to the next available Enhydra server, and mark the first server as "failing" for a user-configurable amount of time. "Failure" here means that the Enhydra server has stopped responding due to a machine crash. Director never retries requests due to application-specific errors. If HTTP request data has been sent to the original server, Director has no way of knowing how much progress was made for that session, and fails that request, sending request to the next available application server. The user's session data would be lost, and they would have to begin another session on the new application server.

The other situation is multi-host session management. Director simply routes requests to Enhydra application servers. The application itself must deal with request data storage. The default request handler creates sessions that apply only to the originating application server. For applications that use the standard session manager, leave "session affinity" enabled for the application's Director connections.

You could write your own session manager that stores session data in a database for every request, but this is likely to severely limit performance (due to the large number of database accesses) and to provide a single point of failure, which clustering is designed to avoid. Though a session-database solution to this limitation in Director is not recommended, that is not to say it won't work. It would require a sophisticated, replicated, high-performance database management system with its own load-balancing schema. Most applications work well with Director's session affinity, so before attempting a session-database solution you should examine the complex issues involved with implementing such a system.

## Increasing performance for Director and Apache on Linux

A common Director configuration consists of an Apache Web server running on i386 hardware under the Linux operating system. The Apache server sends static content to the client, and passes requests for dynamic content to a number of Enhydra application servers using Director. The following tips will help increase performance on such a configuration.

### Hardware considerations

Because Web servers are file I/O bounded (see "Load balancing HTTP requests and Enhydra" on page 171), the amount of available memory and the speed of file I/O are the most important considerations for serving static content.

The amount of memory is the more important hardware factor, assuming your Web server has a reasonably fast processor. Ideally you'd have enough RAM to cache all frequently-used static data to speed up requests and bypass the file system.

**Note** If your network bandwidth is low (below 15 MB/second), adding RAM usually doesn't help performance. The network is the bottleneck in such a circumstance.

The second most important hardware factor is file I/O speed. If you cannot cache all your static content, having a high performance file system, such as a SCSI RAID array, will help you retrieve files faster.

## Tuning the Linux OS

Changing some default settings on your Linux distribution can help improve performance for your Web server and your Enhydra servers.

### Kernel versions

Use the most recent stable production Linux kernel. Considerable effort has been expended to improve the performance of networking in the kernel.

### Remove non-essential daemon

Run only the most essential system daemons. The more daemons running on the server, the less virtual memory space and kernel table space is available for your Web server or Enhydra application server.

### Increase the maximum number of open files and inodes

Increase the maximum number of files and inodes that may be open at once. These values are set in `/proc/sys/fs/file-max` and `/proc/sys/fs/inode-max`. Set the maximum number of files to at least 16348, and the maximum number of inodes to at least 49152.

### Improve file access time

Place the static data on a separate partition, and use the `noatime` option when mounting the file system in `/etc/fstab`. This prevents the system from updating file access times, which you don't need (or can get from the Web server's log files).

**Note** Some benchmarks have shown limited improvement from this change. It has other benefits, however, such as better `fsck` times and improved data integrity.

### Increase the number of available TCP sockets

The default Linux configuration will limit Director to about 260 connections per second. Even sites that do not usually reach this limit may experience load spikes that could cause denial of service errors. If connection rates are likely to exceed 500 connections per second, you need to recompile your kernel to increase the number of available TCP sockets. See "TCP TIME_WAIT problem" on page 176 for instructions on increasing your TCP sockets.

## Tuning the Apache Web Server

The Apache Web Server is focused on flexibility, extensibility, and portability rather than raw performance. Some Apache features can be disabled to reduce the server's overhead and improve performance. If you do need a particular feature, enable it only for the portions of your site that require it, and leave it disabled for the other parts. This section discusses only those features that directly relate to Director and Enhydra performance, but you may need to tune other features to maximize

performance. See the Apache documentation for information on tuning features not discussed here.

### Design your applications to separate static content from dynamic content

Your servers will perform best when Apache is used to server static content and Enhydra is used to serve dynamic content. Enhydra, like all application servers, is not designed for efficient file I/O, whereas Apache is designed to do just that.

### Use the latest stable release of Apache

Use the most recent stable production release of Apache. Similar to the Linux kernel, considerable effort has been made to tune the most recent versions for increased performance. In addition, the default values of many tuning parameters were changed in the 1.3 release of Apache.

### Disable hostname lookups

Disable DNS lookups for Web server requests. This increases performance because Apache doesn't need to query DNS to resolve the client IP address before responding. This feature is disabled by default in Apache 1.3 or greater. If you are running Apache 1.2, set `HostnameLookups` to `Off` in `httpd.conf`.

Additionally, if you use the `Allow from domain` or `Deny from domain` directives, specify IP addresses rather than hostnames.

### Allow Apache to follow symbolic links

For security reasons, Apache is often configured to not follow symbolic links (because an intruder might replace a file with a symbolic link to another file). This check may not be worth the performance cost if other, stronger security measures, such as firewalls, are in place.

You should enable `FollowSymLinks` in all your directories in `httpd.conf`, and never enable `SymLinksIfOwnerMatch`.

### Disable AllowOverride

Apache allows you to dynamically fine-tune file access via the `AllowOverride` directive in `httpd.conf` and the placement of `.htaccess` files in your Web server directories. This is not needed for many sites, and should be disabled by setting `AllowOverride` to `None`.

### Monitor the number of server processes

Apache spawns child processes to handle incoming requests. It dynamically adjusts the number of child processes, and some parameters used in this algorithm can be adjusted. The defaults are reasonable for most circumstances. However, you should check the Apache error log, typically `<Apache_root>/logs/error_log`, to see if there are frequent messages regarding spawning more than four child processes per second. If this occurs, increase the `MinSpareServers` setting in `httpd.conf`.

### Ensure browser caching

Typically, static content files change infrequently, and when they do, they change in predictable ways (for example, files that change daily or weekly). You can reduce the

number of client requests by making sure that client browsers will cache these items instead of wasting resources retrieving them from the Web server each time the user accesses the site.

By default, Apache does not specify expiration times. Client browsers generally cache items without a specific expiration time, but it is preferable to set an explicit time.

The easiest way to set the expiration times is to use the Apache expires module, `mod_expires`. You can use this module to set blanket expiration times for your files, including, for example, an expiration time for all GIF files. For information on `mod_expires`, see http://www.apache.org/docs/mod/mod_expires.html.

Even if you log page hits, and therefore don't want clients to cache the static HTML pages, you can improve performance by caching static graphics, which are often larger than the HTML pages that call them.

**Note** If you do set long expiration times for these files, you will encounter problems if the files change before your expiration times. In this situation, you should give the changed files different URLs.

## TCP TIME_WAIT problem

The TCP protocol includes a "cooling off period" for connections that have recently been closed. This is the TIME_WAIT state. It is required by the TCP specification to ensure that stray data from a defunct connection on the same port does not find its way to the new connection. RFC 793, "Transmission Control Protocol," states that the wait period should be four minutes.

The problem is that this four-minute wait is far too conservative for high-performance transaction-oriented server implementations. For example, we have a server that is in steady state receiving and processing 200 requests per second. Director opens a new connection for each transaction to the back-end Enhydra server. This means that 200 connections per second on average are going into the TIME_WAIT state when they finish. After four minutes, older TIME_WAIT connections begin to disappear, as expected. This means that there will be, on average, about four minutes worth of TIME_WAIT connections at any given time. Doing the math, we get 200 * 4 * 60 = 48000 connections in the TIME_WAIT state.

The maximum number of connections that any one server can have in any state is 65535. This is because RFC 793 specifies that the port that defines the local endpoint for the connection is only a 16-bit unsigned integer, and the largest such number is 65535. Worse, ports 0-1024 are usually reserved for well-known services and are not available as dynamically allocated, or ephemeral, ports. In the best of all cases, there are 64511 ports available.

With a four minute TIME_WAIT, we can compute 64511 ports /(4 minutes * 60 seconds per minute) = 268 connections per second. This means that no matter how fast your server is, connections will begin failing at 268 connections per second average load. Actually, the failures will usually start just before that threshold is reached.

## Solving the TIME_WAIT threshold

The current solution to this problem is to reduce the amount of time that old connections spend in TIME_WAIT. This is technically in violation of RFC 793, but the trade-off in performance gain well offsets this violation.

The purpose of TIME_WAIT is to prevent any lingering duplicates from old connections from finding their way into an existing connection, where they could cause data corruption. Back in the days of 9600-baud X.25 networks, where a packet could very well circulate around many tens of seconds among remote routers, four minutes was a good number to serve the purpose of TIME_WAIT. The Web had not been invented at this point, and even if it had, nobody expected the current hardware to be capable of sustaining 200 or more transactions per second.

On a typical Enhydra high-throughput server, all the back-end Director connections take place on a fast 100Base-T or Gigabit Ethernet subnet. In such a situation, a packet is either going to arrive at its destination or die within milliseconds—if not microseconds—unless server routing misconfigurations are present.

On the Internet side of the server, connections are coming from all locations. It is quite possible that stray packets could linger for quite some time. For a single client-server pair, the four-minute TIME_WAIT is not going to be a problem and may be reasonable if the connection is being reestablished for each transaction. However, a Web server is a server with many different clients. HTTP requests from a single client are in most cases piggy-backed into one TCP session using HTTP "Keep-alive" semantics. Most new connections are coming from different clients, which prevents the stray segment problem.

For a stray segment to make it into a new connection, the following conditions must all be met:

1. The local port of the new connection must match the local port of the old connection.
2. The local IP address of the new connection must match the local IP address of the old connection.
3. The remote port of the new connection must match the remote port of the old connection.
4. The remote IP address of the new connection must match the remote IP address of the old connection.
5. The TCP sequence number in the old segment packet must be such that the new connection will think it is valid. This is highly unlikely, even in the already unlikely event that the other four conditions have been met.

Given these conditions, it should be very safe to set a TIME_WAIT interval that is quite low. Choose a TIME_WAIT value that keeps the average number of outstanding TIME_WAIT connections below 30000. That is, TIME_WAIT_SECS * CONNECTIONS_PER_SECOND < 30000. A value of 60 seconds will allow a theoretical maximum of 1072 connections per second, and up to 500 connections per second without going over 30000 TIME_WAIT connections. If you need more, lower TIME_WAIT even more.

### Changing TIME_WAIT on Linux

On Red Hat 6, run the following command:

```
/sbin/sysctl net.ipv4.ip_local_port_range
```

If you haven't already changed this value, it is 1024 to 4999. This is the range of ephemeral ports available by default, and is too low for a high performance server. Use this command to change the range from 1024 to 65535:

```
/sbin/sysctl -w net.ipv4.ip_local_port_range="1024 65535"
```

By default the TIME_WAIT interval on Red Hat Linux is one minute. This allows up to 1072 connections per second, or 500 per second without exceeding 30000 TIME_WAIT connections. If you need more, you have to change a kernel header file and recompile your Linux kernel. If `/usr/src/linux` is a symbolic link to your current kernel source, change the file `/usr/src/linux/include/net/tcp.h`.

Change the line

```
#define TCP_TIMEWAIT_LEN (60*HZ)
```

to

```
#define TCP_TIMEWAIT_LEN (<TIMEWAITSECS> * HZ)
```

where `<TIMEWAITSECS>` is the number of seconds for connections to remain in the TIME_WAIT state.

For example, if you are changing the TIME_WAIT period to 15 seconds, the line will be:

```
#define TCP_TIMEWAIT_LEN (15 * HZ)
```

Once you have saved `tcp.h`, completely rebuild and install the kernel.

### Changing TIME_WAIT on Solaris

On Solaris, the TIME_WAIT interval is a tunable parameter and can be changed using `ndd`. The value is specified in milliseconds, so to specify a value of 30 seconds, use 30000 milliseconds.

For Solaris 2.6 and earlier:

```
ndd -set /dev/tcp tcp_close_wait_interval 30000
```

For Solaris 2.7 and later:

```
ndd -set /dev/tcp tcp_time_wait_interval 30000
```

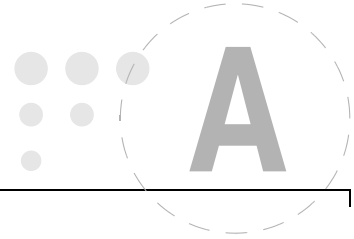
The change takes place immediately on all new connections. Old connections will still wait for the old interval until they expire. You should put this command into a system start-up file so it is run each time the system is rebooted.

### Changing TIME_WAIT on Windows

The TIME_WAIT interval is set in the registry key under `HKEY_LOCAL_MACHINE`:

```
SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\TcpTimedWaitDelay
```

If this key does not exist you can create it as a `DWORD` value. The numeric value is the number of seconds to wait; it can be set to any value between 30 and 240. If it is not set, Windows defaults to 240 seconds for TIME_WAIT.



# Appendix A

# Using SSL with Enhydra

This appendix is a guide to developing Enhydra applications that use Secure Sockets Layer (SSL) encryption. Although this is not a tutorial, if you follow these instructions you can modify the Golf Shop example application to use SSL.

## System requirements

You must have the following to use SSL with Enhydra:

- JDK 1.2.2
- Your JDK must include `keytool` (located in `<JDK_HOME>/jre/bin`)
- Sun's Java Secure Socket Extension Kit (available at http://java.sun.com/products/jsse/)

The Java Secure Socket Extension Kit contains implementations of cryptographic algorithms, and is subject to U.S. export restrictions: You cannot download it in countries that the United States currently has an embargo against. Sun has made a separate version available for export outside the United States and Canada. For more information, see the JSSE website.

## Background

There are two ways to use SSL with Enhydra:

- You can associate Enhydra with a Web server using Enhydra Director. This is the recommended method, as any serious use of encryption will take a large amount of CPU cycles. It is far better to do this with native code than with Java. The preferred setup would consist of Enhydra with Apache and the `mod_ssl` module. See http://www.modssl.org. For information on integrating Director with Apache, see Chapter 9, "Using Enhydra Director."
- If you need a pure Java solution, you can build SSL support directly into Enhydra.

This document discusses the pure Java option. There are a variety of commercial SSL tool kits for Java, but we recommend the Sun Java Secure Socket Extension Kit (JSSE). JSSE combines SSL features found in Sun's Java Web server with the security API of JDK 1.2. This is likely to be a defining standard for other commercial Java SSL vendors, though it may take some time for the vendors to release their own implementations of JSSE.

JSSE is reasonably full featured. Sun's implementation is a reference release: they do not plan on supporting JSSE further. JSSE implements SSL 3.0 and TLS, although it is currently unclear how secure this implementation is.

# Installation and configuration

Here are the basic steps to get SSL working with Enhydra.

1. Install a version of Enhydra as usual, using the built in SSL hooks.
2. If building from source, edit `config.mk`.
3. Download and install the JSSE JAR files.
4. Edit your Java security policy file.
5. Generate or install X509 certificates.
6. Modify Enhydra configuration files to add the SSL connection method.

Each step is described in more detail in the sections that follow.

## Step 1: Install Enhydra

Make sure that you have the version of Enhydra with SSL support. Enhydra has no implementations of cryptographic algorithms, so its export outside the U.S. is not restricted.

## Step 2: Download and install JSSE JAR files

JSSE is currently available only from Sun at http://java.sun.com/products/jsse/. Once you have downloaded the ZIP file, unzip the three JAR files and place them in their target locations.

JSSE is a Java extension kit, so if you place it in `<JDK_root>/jre/lib/ext`, the Java compiler and virtual machine will be able to find the JSSE classes without changing the CLASSPATH.

If you do not have access to this directory, you must run the Enhydra configure script with the `-jsse` option. This flag takes as an argument the directory containing the JSSE JAR files.

## Step 3: Configure Make

If you are building Enhydra from source, set `HAVE_JSSE=YES` in your top-level `config.mk`. Make sure that this environment variable is set after include `$(ROOT)/lib/make/stdrules.mk`, or it will not work.

## Step 4: Edit the Java security file

You can find your Java security file at `<JDK_root>/jre/lib/security/java.security`. Find the list of security providers. The default is:

```
security.provider.1=sun.security.provider.Sun
```

To add the default JSSE security provider, add this line:

```
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
```

The numbering refers to the order in which the security providers are used. If you are using a vendor's implementation of JSSE, the security provider will be something else. Refer to your vendor documentation for more information.

## Step 5: Generate or install your X509 certificates

If you are testing your setup, you will want to generate your X509 certificate yourself. If you are building a production site, you will need to purchase a certificate from a certificate authority such as Thawte or Verisign.

### Generating your private key

You can use the JDK `keytool` utility to generate your own X509 certificates, but be aware that this is a memory-intensive operation. Also, when you generate a certificate you will need to give information to the `keytool` utility. Do not lose this information.

At the command prompt, presuming that `<JDK_root>/jre/bin` is in your path, enter:

```
keytool -genkey -alias name -keyalg RSA
```

**Note** The program will not work without the `-keyalg` RSA option. Netscape uses RSA encryption, but the keytool uses DSA by default. At this point you will be prompted to confirm that the information is correct. If it is then the program will proceed to generate a self-signed certificate and key. This may take some time. Finally, you will be prompted for a password for the certificate. Make a note of the password because you will not be able to use the certificate without it.

Do not attempt to run this command until you have changed your `java.security` file as described previously. If you do, you will get the following error:

```
keytool error: KeyPairGenerator not available
```

An RSA-enabled provider is not provided with the default JDK1.2.x.

When you run the `keytool` command, you will be prompted for the following information:

- **keystore password**: If this is the first time you are running `keystore`, it creates a keystore in your home directory, and you will be prompted to create a keystore password. You will need this password every time you use any key management. The alias is the name that will identify the key in the keystore (you can have several keys in your keystore). If you do not specify an alias, the default name is `mykey`.

- **First and Last name**: The fully qualified domain name of the host from which you'll be running your server (for example, www.mycompany.com). It is important for you to enter the correct name because the certificate authority will not issue a certificate if the name is incorrect.
- **Name of organizational unit**: This is not the company name, but rather the name of an internal department (for example "Engineering").
- **Name of your organization**: The full name of your company (for example "MyCompany Incorporated").
- **City or location**: For example, "Santa Cruz."
- **State or Province**: For example, "California."
- **Country code**: For example, "USA."

To verify that the key was properly created in the keystore, you can verify it with:

```
keytool -list
```

## Generating a certificate request

If you want a certificate, either your own self-signed certificate or one from a recognized certificate authority, you will need to generate a certificate request. You can submit the certificate request to your certificate authority or issue your own self-signed certificate using either `keytool` or OpenSSL.

To generate a certificate request, type the following at the command prompt:

```
keytool -alias <name> -certreq -file <filename>
```

where *<name>* is the alias of the key in the keystore that you are generating the request against. The specified file is where the certificate request will be written to. If no file is specified, the request will be output to standard output.

A successful certificate request should look like this:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBuzCCASQCAQAwezELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExEzARBgNVBAcT
CnNhbnRhIGNydXoxDzANBgNVBAoTBmx1dHJpczEbMBkGA1UECxMSc2VjdXJlIGRldmVsb3BtZW5O
MRQwEgYDVQQDEwtzdGV2ZSBsYXRpZjCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAuunCyGrr
wCCZeUAJrCvoN/n82k8IF1OwH7KNzAyaPgMU6L7CcawvWqVQY/TncHZmy5tvIlNaEJR3OOHa8Keo
TxWIG7T/GHgwqBcjmt/reZbvKdKxBnT7ocoWx2G5BjHoN8RxMLQtZIc/vd9QUrelfw3WMTSLoT4A
QJiAOQpcSvECAwEAAaAAMAOGCSqGSIb3DQEBBAUAA4GBAEkeC/6FzrLOOEUAgOzaIDHazB7kKqZH
cbxpAhPC1/x9ow5D/BOwUf114lMK1L8rNDSv9UlTrRJfG36cNFHIiomu42kUovFV774AdOup6IFO
AFFOXitH2CiZVM5458NVECGdlNauRvjpwQvsRcRHC2rEpfTDOdb9ISH/lNOJmDUz
-----END NEW CERTIFICATE REQUEST-----
```

The first and last lines (which start with `BEGIN` and `END)` are part of the certificate request and should not be removed.

To validate the certificate request, use OpenSSL or `keytool`. Even though OpenSSL is not Java-based, it is more robust and has a more refined set of command-line tools than `keytool`. OpenSSL is an open-source implementation of the SSL and TLS protocols, and is widely used with the Apache `mod_ssl` package to provide SSL servers. If you have any doubts about open-source cryptographic packages, note that OpenSSL and `mod_ssl` are used by three commercial Apache vendors as the basis for

their secure servers. Refer to http://www.openssl.org and http://www.modssl.org for downloads and documentation.

When OpenSSL is installed, run the command

```
openssl req -noout -text -in <csr>
```

where <csr> is the name of the file containing your certificate request.

## Submitting your certificate request

To submit your certificate to a recognized certificate authority, consult the instructions on their Web page. Two well known certificate authorities are: http://www.thatwte.com and Verisign.

If you are doing development and are creating your own self-signed certificate, run the command

```
keytool -sefcert -alias <keyname>
```

where <keyname> is the alias of the key you want to associate with the certificate. The certificate is stored in the keystore.

To validate the certificate, you first need to export the certificate from the keystore:

```
keytool -export -alias <name> -file <filename>
```

where <name> is the alias of the associated key, and <filename> is the name of the file that the certificate will be written to.

Now read the certificate information:

```
keytool -printcert -file <file>
```

where <file> is the name of the file with the exported certificate.

## Importing a certificate

If you are using a certificate authority to issue your certificate, you will receive a file that looks like this:

```
-----BEGIN CERTIFICATE-----
MIIC3DCCAkWgAwIBAgIDATZXMA0GCSqGSIb3DQEBBAUAMIHEMQswCQYDVQQGEwJa
QTEVMBMGA1UECBMMV2VzdGVybiBDYXBlMRIwEAYDVQQHEwlDYXBlIFRvd24xHTAb
BgNVBAoTFFRoYXd0ZSBDb25zdWx0aW5nIGNjMSgwJgYDVQQLEx9DZXJ0aWZpY2F0
aW9uIFNlcnZpY2VzIERpdmlzaW9uMRkwFwYDVQQDExBUaGF3dGUgU2VydmVyIENB
MSYwJAYJKoZIhvcNAQkBFhdzZXJ2ZXItY2VydHNAdGhhd3RlLmNvbTAeFw0wMDA3
MjQyMjMyNDBaFw0wMTA4MDcyMjMyNDBaMHoxCzAJBgNVBAYTAlVTMRMwEQYDVQQI
EwpDYWxpZm9ybmlhMRMwEQYDVQQHEwpTYW5OYSBDcnV6MSIwIAYDVQQKExlMdXRy
aXMgVGVjaG5vbG9naWVzLCBJbmMuMROwGwYDVQQDExRiZWVsemVidWIubHVOcmlz
LmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA4pMbXgVDOjBrQHW5Xqpj
jfSQ7OHzCwagrUyHPtV5LbvLffInJ2mAhihlqwPxCmrOHnYIioDxtJgr/3gqfL9C
IC1/L1xlExO6IKBkFs9X4XVXPay2DzFFGnpvCvSlEjCYobHpK+QqwF8bJrnEa9Bd
oyLyxkGBGthaQkxUJARus+MCAwEAAaMlMCMwEwYDVR0lBAwwCgYIKwYBBQUHAwEw
DAYDVR0TAQH/BAIwADANBgkqhkiG9w0BAQQFAAOBgQC6xEHb6Is9jUJUfO6XfWiD
wrZ4/IOYnA52bg54NVTTyjjl3qxcQpanAwajp6aAnWUYb34MuRZ8dpsYVu3TUjNF
xxgvOMWQByb4LIjv+l2JcTO4a5ZmFp7Kqp6U2XgdgcS2YYxG+mMQmTdJ3PjCB4Od
g3TILQ8TdSHnSG4YaQgNPw==
-----END CERTIFICATE-----
```

You can verify that the certificate works by using OpenSSL:

```
openssl x509 -noout -text -in <crt>
```

where <*crt*> is the name of your certificate file.

Or, you can use `keytool`:

```
keytool -printcert -v -file <crt>
```

where <*crt*> is the name of your certificate file.

**Note** `keytool` and `openssl` handle certificates in different ways. `keytool` complains that a certificate is unreadable if it contains a new line at the end of the file, but `openssl` does not have this problem.

Once you have verified your certificate, you can import it into your keystore by issuing the command:

```
keytool -import -alias <name> -file <crt> -trustcacerts
```

where <*crt*> is the name of your certificate file, and <*name*> is the name of the alias that you want to associate with the certificate. The `trustcacerts` option tells `keytool` to look in the `cacerts` file found in the <*JDK_root*>`/jre/lib/security` directory. This file contains the root certificates for Thawte and Verisign, and `keytool` uses them to verify the certificates you input into the keystore.

**Note** If you are using a certificate authority other than Thawte or Verisign, you will have to import that certificate authority's root certificates into the <*JDK_root*>`/jre/lib/security/cacerts` file. To do this, download the root certificate files from your certificate authority, then run `keytool`:

```
keytool -import -alias <name> -file <filename> -keystore cacerts
```

where <*name*> is the alias that you want to associate with the certificate and <*filename*> is the name of the file containing the root certificate.

# Modifying your application

Now you can alter your Enhydra application configuration file so that it can find the certificates and keys. For example, here is the configuration file for the GolfShop demo that is shipped with the Enhydra source code. This configuration file is in the <*GolfShop_root*>`/output` directory.

This example uses the XMLC implementation of the GolfShop demo.

Add the following to the configuration file:

```
# begin --------------------------------
Connection.golfPortSSL.Type = https
Connection.golfPortSSL.Port = 8443
Connection.golfPortSSL.SecureRandomAlgorithm =SHA1PRNG
Connection.golfPortSSL.SecureRandomProvider = SUN
Connection.golfPortSSL.SSLContextProvider = SunJSSE
Connection.golfPortSSL.SSLContextProtocol =TLS
Connection.golfPortSSL.KeyStoreLocation="/home/steve/.keystore"
Connection.golfPortSSL.KeyStoreProvider=JKS
Connection.golfPortSSL.KeyManagerAlgorithm = SUNX509
Connection.golfPortSSL.KeyManagerProvider = SunJSSE
Connection.golfPortSSL.TrustManager=JSSE
Connection.golfPortSSL.Password = <your password here>
Connection.golfPortSSL.ClientAuthentication=false
#
```

```
# Connect the port to the application
#
Channel.golfPortSSL.golfChannel.Servlet = GolfShopSSL
Channel.golfPortSSL.golfChannel.Url= /
Channel.golfPortSSL.golfChannel.Enabled = yes
#
# Specify applications (no admin).
#
Application.GolfShopSSL.ConfFile = GolfShopXMLC.conf
Application.GolfShopSSL.Description = "Enhydra Demo Secure Shopping Cart Application (SSL)."
Application.GolfShopSSL.Running = yes
#end ----------------------------------------
```

## Configuration file in detail

This section explains each line you added to the configuration file in detail.

`Connection.golfPortSSL.Type = https`

Defines the connection method. This line is required if you want to use SSL.

`Connection.golfPortSSL.Port = 8443`

Defines the port to connect to with the HTTPS method. The default port for SSL is 443, but that is a privileged port on UNIX and you will need to be root to use it. The HTTP alternative ports are in the 8000 range. If you are testing your application on a port other than 443, Internet Explorer will not be able to connect to it. Netscape does not have a problem with SSL on nonstandard ports.

A workaround is to use SSH port-forwarding to bind port 44 on your local machine to the port on which Enhydra is running. For example, if your Enhydra application is running on `foo.bar.org` on port 8443 where you are user `steve`, then your invoke ssh: `ssh -x -l steve -L 443:foo.bar.org:8443 steve@foo.bar.org`.

`Connection.golfPortSSL.SecureRandomAlgorithm =SHA1PRNG`

Java security provides a cryptographically strong Pseudo Random Number Generator (PRNG). This specifies the algorithm.

`Connection.golfPortSSL.SecureRandomProvider = SUN`

The provider refers to the providers in the `java.security` file, in this case SUN or SSL.

`Connection.golfPortSSL.SSLContextProvider = JSSE`

The SSLContext Provider currently defaults to JSSE. The SSLContext holds the state of the SSL implementation. SSLContext is used to generate the factories for the sockets.

`Connection.golfPortSSL.SSLContextProtocol =TLS`

This currently has two defaults: SSL or TLS. TLS is a protocol that is a likely replacement for SSL 3.0.

`Connection.golfPortSSL.KeyStoreLocation="/home/steve/.keystore"`

The keystore is generated and managed by the `keytool` utility. The default is to have it in your home directory.

`Connection.golfPortSSL.KeyStoreProvider=JKS`

The keystore provider.

`Connection.golfPortSSL.KeyManagerAlgorithm = SUNX509`

Currently `SUNX509` is the only legal value.

`Connection.golfPortSSL.KeyManagerProvider = JSSE`

This is currently the only provider. The name may change to SunJSSE in the future.

`Connection.golfPortSSL.TrustManager=JSSE`

The Trust manager.

`Connection.golfPortSSL.Password = <your password here>`

When you generated the key and certificate or the certificate request you had to specify a password.

At this point, assuming that everything is correctly configured, you should be able to start up Enhydra and connect on port 8443.

# For more information on Java and SSL

The following books, newsgroups, and Web pages provide additional information about Java security issues and SSL.

- For general Java security, see *Java Security by Scott Oakes* (published by O'Reilly & Associates).
- For an interesting and topical book, see *Java2 Network Security* by Marco Pistoia (published by Prentice-Hall). Note that the sample code has many bugs.
- The best source for JSSE information is the Javadoc files that come with the JAR files. In particular, see `overview.html`, `API_users.html`, and `addtional.html` files, which are very useful.
- The `comp.lang.java.security` newsgroup has occasional discussions on JSSE.
- Sun has a mailing list archive at http://java.sun.com/security/hypermail/java-security-archive, which has many interesting posts related to JSSE.
- An excellent resource to learn about SSL is the open-source OpenSSL libraries at http://www.openssl.org. OpenSSL forms the basis of most of the commercial SSL versions of Apache when combined with mod_ssl (http://www.modssl.org).
- For an excellent paper on SSL scalability issues, see http://www.awe.com/mark/apcon2000/. The authors are two of the main OpenSSL developers.

# Appendix B

# XMLC metadata file schema

This appendix describes the schema of XMLC metadata files. For general information on XMLC metadata files, see “Using XMLC metadata files” on page 84.

## Structure

The hierarchy of tags in an XMLC schema file is as follows:

```
<xmlc>
   <compileOptions/>

   <parser>
      <xcatalog/>
   </parser>

   <html>
      <htmlTagSet/>
      <htmlTag/>
      <htmlAttr/>
      <compatibility/>
   </html>

   <domEdits>
      <deleteElement/>
   </domEdits>

   <documentClass>
      <implements/>
   </documentClass>

   <javaCompiler>
      <javacOption/>
   </javaCompiler>

</xmlc>
```

## Tag reference

This section contains an alphabetical reference of all the tags allowed in the XMLC schema file. Each entry corresponds to an XML tag, and contains the subsections:

- **Content:** Tags that the tag can contain.

- **Attributes:** Attributes the tag can have.
- **Context:** Tags within which the tag can appear, in other words, the tags that can contain it.

So, for a tag `<sampleTag>`, whose attributes are `attribute1`, `attribute2`, and so on, whose context is `<contextTag>`, and that can contain `contentTag`, its general syntax would look like:

```
<contextTag>
   <sampleTag attribute1 attribute2 ...>
      <contentTag/>
   </sampleTag>
</contextTag>
```

## <compatibility>

Enables compatibility with the way older version of XMLC handled HTML.

**Content** None.

**Attributes** The `<compatibility>` tag has the following attributes:

#### oldClassConstants

Older versions of XMLC generated HTML class attribute constants as all upper case (e.g. CLASS_DELETEROW) with values being case-preserved. If a true value, this option will reproduce the old behavior. This option is intended to aid in the porting of existing applications; it may not be support in future releases.

#### oldNameConstants

Older versions of XMLC generated HTML name attribute constants as all upper-case (e.g. NAME_INPUT) with values being case-preserved. If a true value, this option will reproduce the old behavior. This option is intended to aid in the porting of existing applications; it may not be support in future releases.

**Context** `<html>`

## <compileOptions>

Specifies options for the document compiler.

**Content** None.

**Attributes** The `<compileOptions>` tag has the following attributes:

#### printVersion

Print the XMLC version number (boolean value).

### keepGeneratedSource
Keep the generate Java source, do not delete it (boolean value).

### printDocumentInfo
Print useful information about the contents of the document, such as ids and URLs (boolean value).

### printParseInfo
Print detailed information about the page parsing (boolean value).

### printDOM
Print out the DOM tree for the page (boolean value).

### printAccessorInfo
Print the signature of each generated access method and constant (boolean value). This also lists the methods and access constants that were not generated due to being invalid Java identifiers.

### compileSource
If true, the generate source will be compiled, if false don't compile the source. Default is true. (boolean value).

### inputDocument
URL of the document to compile.

### processSSI
If true, process server-side include directives. If false, pass them through as comments. Default is false.

### documentFormat
The format of the document. Value is one of xml, html, or unspecified. If unspecified, then first line of the file is checked for an XML header. If an XML header is found, xml is assumed, other html is assumed. The default is unspecified. This attribute is only required for parsing XML files that don't have a XML header as the first line.

### sourceOutputRoot
Specifies the root directory for the generate source files. A full package hierarchy is created under this directory. If not specified, the file is created in the current directory. If `-keep` is specified, the generate source files will be saved under this directory.

### classOutputRoot
Specifies the root directory for the compiled class files. A full package hierarchy is created under this directory. If not specified, the file is created in the current directory. The metadata file for recompilation is also created in this directory.

### compileSource
If false, the source code will not be generated or compiled. This is useful with the documentOutput option, for validating documents, and for print information about the documents. Default is true. (boolean value).

### documentOutput
Write a the document, after DOM editing, to file. This is useful for pages where the URL's must be mapped to reference dynamic pages, but there is no other dynamic content. Normally used with `createCode="false"`.

**Context** `<xmlc>`

## <deleteElement>

Delete all matching elements. This is often used to specify the element class of mock-up data that is to be deleted from the document.

**Content** None.

**Attributes** The `<deleteElement>` tag has the following attributes:

### elementIds
Restrict replacement to the elements matching any of the ids in the space-separate list.

### elementClasses
Restrict replacement to the elements any of the class names in the space-separate list. These are class attribute values, as specified by the HTML CLASS attribute, not Java class names.

### elementTags
Restrict replacement to the elements matching any of the tag names in the space-separate list. Case is ignored for HTML.

**Context** `<domEdits>`

## <document>

Not yet implemented.

## <documentClass>

Specify properties of the XMLC document class to generate.

**Content** `<implements>`

**Attributes** The `<documentClass>` tag has the following attributes:

### name

The fully-qualified name of the class to generate. If generate specifies class, then this is the name of the class. Otherwise, this is the name of the interface to generate, with the generated implementation will have "`Impl`" appended to this name.

### generate

Specifies what kind of classes should be generated. Normally either `class` or `both` is used. Valid values are:

- **class:** Generate a simple class (default).
- **interface:** Generate just an interface but not the implementation.
- **implementation:** Generate an implementation of the interface, named in the form "`nameImpl`," but not the interface.
- **both:** Generate both an interface and an implementation.

### delegateSupport

Generate code for delegate support used by for XMLC document class reloading. Boolean value, default is false.

### createMetaData

Create a XMLC meta-data XML file in the same directory as the class file. This is used by the XMLC recompilation factory. Boolean value, default is false.

### recompilation

Set all options required for XMLC recompilation. Boolean value, default is false. Setting this to true results in:

- `generate="both"`
- `delegateSupport="true"`
- `createMetaData="true"`

### extends

Specify the class that the generated class will extend. This class must extend org.enhydra.xml.xmlc.XMLObjectImpl for XML documents or org.enhydra.xml.xmlc.html.HTMLObjectImpl for HTML documents.

### domfactory

Specify the Java class for creating DOM documents. This class must implement org.enhydra.xml.xmlc.dom.XMLCDomFactory. The option is not supported for HTML documents. The DOM factory must have a constructor that takes no arguments. This class serves as a factory for Document objects, giving a mechanism for creating DTD-specific DOMs. Element classes that correspond to a specific element types. The specified class must be available on the CLASSPATH.

### dom

Specify one of a predefined set of DOM factories. This is a short-cut for the domfactory attribute. These are the valid values along with the XMLCDomFactory they map to are:

- **lazydom:** The LazyDOM, derived from the Xerces DOM.
  - **XML:** org.enhydra.xml.xmlc.dom.lazydom.LazyDomFactory
  - **HTML:** org.enhydra.xml.xmlc.dom.lazydom.LazyHTMLDomFactory
- **xerces:** The Xerces DOM.
  - **XML:** org.enhydra.xml.xmlc.dom.xerces.XercesDomFactory
  - **HTML:** org.enhydra.xml.xmlc.dom.xerces.XercesHTMLDomFactory

The default value, if neither the domfactory or dom attributes are specified is `lazydom`.

### recompileSourceFile

Specifies the classpath-relative path of the source file that the recompilation factory will use. This is compiled into the class as the value of the XMLC_SOURCE_FILE field. If not specified, the file path is generated by converting the package name into a file path and merging with the base name of the source file.

### elementAccessorReturnType

Specifies the class or interface to use as the return type for the all generated `getElementXXX()` methods. If not specified, the return type is obtained for each element using `XMLCDomFactory` associated with the document.

The type specified must implement `org.w3c.dom.Element` as well as be assignment compatible (without casting) to the actual return type. This is used to make the class more generic and make it possible to write interfaces to which multiple documents conform. One of the following value maybe used instead of a class or interface name:

- **Element:** a short-cut for `org.w3c.Element`.
- **class:** the actual class name of the element.
- **interface:** the value obtained from the `nodeClassToInterface()` method in the `XMLCDomFactory` object being used to compile the document.

**Context** <xmlc>

## <domEdits>

tag containing DOM editing specifications. These are modifications done to elements in the DOM during the compilation of a document.

**Content**
- <urlEdit>
- <urlMapping>
- <urlRegExpMapping>
- <deleteElement>

**Attributes** None.

**Context** <xmlc>

## <html>

Tag containing HTML-specific options.

**Content**
- <htmlTagSet>
- <htmlTag>
- <htmlAttr>
- <compatibility>

**Attributes**

**encoding**
Specify the encoding to use when reading a HTML document. This is a HTML encoding name, not a Java encoding name

**Context** <xmlc>

## <htmlTagSet>

Add a predefined set of proprietary HTML tag and attributes to the list of valid HTML tags. This option is only used by the tidy parser.

**Content** None.

**Attributes** The <htmlTagSet> tag has the following attributes:

**tagSet**
The name of the tag set. The following tag sets are defined:

- **cyberstudio:** Tags added by Adobe CyberStudio.

**Context** <html>

## <htmlTag>

Add a proprietary tag to set of valid HTML tags.

**Content** None.

**Attributes** The `<htmlTag>` tag has the following attributes:

**name**
The name of the tag.

**inline**
Tag is a character level element.

**block**
Tag is a block-like element, e.g. paragraphs and lists.

**empty**
The tag does not have a closing tag.

**optclose**
The closing tag is optional.

**Context** `<html>`

## <htmlAttr>

Add a proprietary attribute the list of valid HTML attributes. The attribute will be allowed on all tags. This option is only used by the tidy parser.

**Content** None.

**Attributes** The `<htmlAttr>` tag has the following attributes:

**name**
The name of the attribute. It will be allowed for all HTML tags.

**Context** `<html>`

## <implements>

Specifies the name of an interface the document class will implement.

**Content** None.

**Attributes** The <implements> tag has the following attributes:

**name**
The fully qualified class name of the interface that will the generated class will implement.

**Context** <documentClass>

## <javaCompiler>

Specify information about the Java compiler.

**Content** <javacOption>

**Attributes** The <javaCompiler> tag has the following attribute:

**javac**
Specify the command name of the Java compiler to use.

**Context** <xmlc>

## <javacOption>

Specifies an option to pass to the Java compiler.

**Content** None.

**Attributes** The <javacOption> tag has the following attributes:

**name**
The name of an option understood by the specified Java compiler. The flag argument should contain the leading minus (-) or plus (+) characters.

**value**
The value to associate with the option. Omitted if the option doesn't take a value.

**Context** <javaCompiler>

## <parser>

Specifies the parser and parsing options.

**Content** <xcatalog/>

**Attributes** The <parser> tag has the following attributes:

#### name
Name of the parser. The valid values are:

- **tidy:** Use the Tidy parser for parsing HTML. This is the default for HTML. Validation is always done by this parser.
- **swing:** Use the Swing parser for parsing HTML. Limited validation is always done by this parser.
- **xerces:** Use the Xerces parser for parsing XML. This is only XML parser and is the default. Validation is optional with this parser. The default is to validate.

#### validate
Changes the default document validation mode of the parser. An error is generated if the value isn't supported by the parser. Boolean value.

**Context** <xmlc>

## <urlMapping/>

Specifies the literal replacement of URLs in element attributes. When used in the <domEdits> tag, it applies globally to elements. When use in a element or sub-document, it applies only to that context.

**Content** None.

**Attributes** The <urlMapping> tag has the following attributes:

#### editAttrs
List of attributes that are to be edited. If not specified, defaults to the attributes that the XMLCDomFactory object for the document defines as containing URLs.

#### elementIds
Restrict replacement to the elements matching any of the ids in the space-separate list.

#### elementClasses
Restrict replacement to the elements any of the class names in the space-separate list. These are class attribute values, as specified by the HTML CLASS attribute, not Java class names.

#### elementTags
Restrict replacement to the elements matching any of the tag names in the space-separate list. Case is ignored for HTML.

**url**
The existing URL. If not specified, all URLs for the matching elements will be replaced.

**newUrl**
The replacement URL.

**Context** `<domEdits>`

## <urlRegExpMapping/>

Specifies the regular expression replacement of URLs in element attributes. If the regular expression matches, it is substituted using a replacement pattern. POSIX extended regular expressions are used implemented uses the gnu.regexp package. Set the documentation on this package for details of the regular expression and substitution syntax. When used in the `<domEdits>` tag, it applies globally to elements. When use in a element or sub-document, it applies only to that context.

**Content** None.

**Attributes** The `<urlRegExpMapping>` tag has the following attributes:

**editAttrs**
List of attributes that are to be edited. If not specified, defaults to the attributes that the XMLCDomFactory object for the document defines as containing URLs.

**elementIds**
Restrict replacement to the elements matching any of the ids in the space-separate list.

**elementClasses**
Restrict replacement to the elements any of the class names in the space-separate list. These are class attribute values, as specified by the HTML CLASS attribute, not Java class names.

**elementTags**
Restrict replacement to the elements matching any of the tag names in the space-separate list. Case is ignored for HTML.

**regexp**
The POSIX regular expression to match against URL tag attributes.

**subst**
The substitution expression to use to the generate the replacement URL.

**Context** <domEdits>

## <xcatalog>

Specifies a XCatalog file to use in resolving external entities when parsing XML files. This element maybe specified multiple times. The catalogs will be searched in the order specified.

**Content** None.

**Attributes** The <xcatalog> tag has the following attribute:

**url**
The URL of the XML catalog file.

**Context** <parser>

## <xmlc>

Root element for XMLC metadata documents.

**Content**

- <compileOptions>
- <parser>
- <html>
- <domEdits>
- <document>
- <documentClass>
- <javaCompiler>

**Attributes** None.

**Context** None (top-level tag).

# Appendix C

# DOML file syntax

This appendix describes the syntax of DOML files, which are used by the Data Object Design Studio (DODS) to generate data access code for Enhydra applications.

## Structure

The hierarchy of tags in a DOML file is as follows:

```
<doml>
   <database>
      <package>
         <package>
         ....
         </package>
         <table>
            <column>
               <type/>
               <referenceObject/>
               <initialValue/>
               <javadoc/>
            </column>
         </table>
      </package>
   </database>
</doml>
```

## Tag reference

This section contains an alphabetical reference of all the tags allowed in DOML files. Each entry corresponds to an XML tag, and contains the subsections:

- **Content:** Tags that the tag can contain.
- **Attributes:** Attributes the tag can have.
- **Context:** Tags within which the tag can appear, in other words, the tags that can contain it.

So, for a tag <sampleTag>, whose attributes are attribute1, attribute2, and so on, whose context is <contextTag>, and that can contain contentTag, its general syntax would look like:

```
<contextTag>
   <sampleTag attribute1 attribute2 ...>
      <contentTag/>
   </sampleTag>
</contextTag>
```

## <doml>

Root element of DOML files. This tag contains a database hierarchy that contains all the packages.

**Content**

- <database>

**Attributes** None.

**Context** None.

## <database>

Contains the package hierarchy, and specifies the database.

**Content**

- <package>

**Attributes** The following attributes are valid for <database>:

### database

The database attribute specifies the database vendor. The valid types are as follows:

**Table C.1** Valid database attributes

| Attribute | Description |
|---|---|
| Standard_JDBC | Generated SQL code will conform to standard JDBC SQL. This is the default attribute. |
| Oracle | DODS will generate SQL optimized for Oracle databases. |
| Informix | DODS will generate SQL optimized for Informix databases. |
| MSQL | DODS will generate SQL optimized for MS SQL Server databases. |
| Sybase | DODS will generate SQL optimized for Sybase databases. |
| PostgreSQL | DODS will generate SQL optimized for PostgreSQL databases. |

**Context** <doml>

## <package>

Each package can contain a sub-package or a table structure.

**Content**

- <package>
- <table>

**Attributes** <package> has the following attributes:

### id

The name of the package. The format for the name includes the parent package's id value. For example, if I had a package myPackage, and a sub-package of it called mySubPackage, mySubPackage's id value would be myPackage.mySubPackage.

**Context** <database>

## <table>

<table> describes a table in a database.

**Content**

- <column>

**Attributes** <table> has the following attributes:

### id

Similar to the id attribute in <package>, <table>'s id contains the value of the table name located in a package. For example, if I had a package myPackage, a sub-package mySubPackage, and a table myTable, the id value is myPackage.mySubPackage.myTable.

### dbTableName

The actual SQL table name. By default this is the same as the id value, minus the package information. For example, myPackage.mySubPackage.myTable's dbTableName is myTable.

### isFinal

The possible values for isFinal are:

- true
- false

### isLazyLoading

This attribute specifies whether the DO will use lazy loading. If a DO uses lazy loading, when you supply a known ObjectId to create a DO instance, the DO instance is created but the corresponding row in the table is not retrieved until the first get() or set() method call is made. It delays the hit on the database until

the moment the data is actually needed. The possible values for `isLazyLoading` are:

- `true`
- `false`

### isAbstract

This attribute specifies whether the data object is abstract. In DODS, all data objects that are extended must be abstract. This is because of how DODS handles database tables. Only non-abstract leaf classes have tables in the database.

The possible values for `isAbstract` are:

- `true`
- `false`

### isEJB

This attribute is not currently used by DODS.

### isView

This attribute is not currently used by DODS.

### extensionOf

The name of another table of which this table is an extension. The value of `extensionOf` must match the `id` of a previously defined table.

### caching

This attribute specifies whether the DO is fully cached, partially cached, or not cached. Cached DOs are stored locally, and queries for the cached DO do not go to the database table. The query instead looks up the information in the cache. The values for `caching` are:

- `none`
- `partial`
- `full`

**Context** `<package>`

## <column>

`<column>` describes a column in a database table.

**Content**

- `<error>`
- `<javadoc>`
- `<referenceObject>`
- `<type>`
- `<initialValue>`

**Attributes** <column> has the following attributes:

#### id
The name of the column in the database table.

#### isConstant
Specifies whether the column contains a constant value.

The possible values for isConstant are:

- true
- false

#### isIndex
This attribute specifies whether the table column will be an index. If isIndex is true, an index is created on this column in the table.

The possible values for isIndex are:

- true
- false

#### isUnique
Specifies whether the column must contain unique values.

The possible values for isUnique are:

- true
- false

#### usedForQuery
Specifies whether the values of the column will be used for queries.

The possible values for usedForQuery are:

- true
- false

**Context** <table>

## <javadoc>

The <javadoc> tag contains the text for Javadoc entries for the column.

**Content** None.

**Attributes** None.

**Context** <column>

## <referenceObject>

If the column is a reference to another table, <referenceObject> specifies the table.

**Content** None.

**Attributes** <referenceObject> has the following attributes.

**constraint**
Specifies whether the specified table row must exist.

The possible values for constraint are:

- true
- false

**reference**
Specifies the ID of the referenced table.

**Context** <column>

## <type>

<type> dictates the form of the data stored in the column. If no <type> is specified, the column contains all default values.

**Content** None.

**Attributes** <type> has the following attributes.

**canBeNull**
Specifies whether the column can contain null values.

The possible values for canBeNull are:

- true
- false

**dbType**
Specifies the internal SQL data type the database will use for this column. The default value of dbType is VARCHAR.

**javaType**
Specifies the Java data type returned by the DO to the user when querying this attribute of the DO. The default value of javaType is String.

**size**
Specifies the size of data types that are commonly measured in width, like `VARCHAR`. `size` must be an integer.

**Context** `<column>`

### <initialValue>

`<initialValue>` is used to specify a default initial value for the column.

**Content** None.

**Attributes** None.

**Context** `<column>`

## Sample DOML file

The following snippet is the contents of a DOML file, `sample.doml`, that will create tables containing data about cars, car dealers, and car owners.

```
<?xml version="1.0" encoding="UTF-8"?>
<doml>
   <database database="Standard">
      <package id="sample">
         <table id="sample.Dealer">
            <column id="Name">
               <type dbType="VARCHAR" javaType="String"/>
            </column>
         </table>
         <table id="sample.Owner">
            <column id="Name">
               <type dbType="VARCHAR" javaType="String"/>
            </column>
            <column id="Age">
               <type dbType="INTEGER" javaType="int"/>
            </column>
         </table>
         <table id="sample.Car">
            <column id="LicensePlate">
               <type dbType="CHAR" javaType="String"/>
            </column>
            <column id="Dealer">
               <referenceObject reference="sample.Dealer"/>
               <type dbType="none" javaType="sample.DealerDO"/>
            </column>
         </table>
         <table id="sample.CarOwner">
            <column id="Car">
               <referenceObject reference="sample.Car"/>
               <type dbType="none" javaType="sample.CarDO"/>
            </column>
```

```
                    <column id="Owner">
                       <referenceObject reference="sample.Owner"/>
                       <type dbType="none" javaType="sample.OwnerDO"/>
                    </column>
                    <column id="IsCurrent">
                       <type dbType="BIT" javaType="boolean"/>
                    </column>
                 </table>
              </package>
           </database>
        </doml>
```

# Index

## Symbols



## A

## B



## C

## D

## E

## F

## G



## H

## I



## J

## K



## L

## M



## N

## O



## P

## Q



## R

## S

# T

## U



## V



## W

## X



## Y